Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros Industriales

A General Architecture for Autonomous Navigation of Unmanned Aerial Systems

Ph.D. Thesis

José Luis Sánchez López
M.Sc. in Automation and Robotics
Industrial Engineer

2017

Departamento de Automática, Ingeniería Eléctrica y Electrónica
e Informática Industrial
Escuela Técnica Superior de Ingenieros Industriales
Universidad Politécnica de Madrid

A General Architecture for Autonomous Navigation of Unmanned Aerial Systems

A thesis submitted for the degree of
Doctor of Philosophy in Automation and Robotics

Author: **José Luis Sánchez López**
M.Sc. in Automation and Robotics
Industrial Engineer

Advisors: **Pascual Campoy Cervera**
Prof. Dr. in Automation and Robotics

Martín Molina González
Prof. Dr. in Computer Science
and Artificial Intelligence

Madrid, May 2017

Título:
A General Architecture for Autonomous Navigation of Unmanned Aerial Systems

Autor:
José Luis Sánchez López
M.Sc. in Automation and Robotics, Industrial Engineer

Directores:
Pascual Campoy Cervera
Prof. Dr. in Automation and Robotics
Martín Molina González
Prof. Dr. in Computer Science and Artificial Intelligence

Tribunal nombrado por el Mgfco. y Excmo. Sr. Rector de la Universidad Politécnica de Madrid
el día ..... de ..... de 2017

Tribunal

Presidente : ....., ...

Secretario : ....., ...

Vocal : ....., ...

Vocal : ....., ...

Vocal : ....., ...

Suplente : ....., ...

Suplente : ....., ...

Realizado el acto de lectura y defensa de la tesis en ..... el día ..... de.....de.....

Calificación de la Tesis.....

El Presidente: Los Vocales:

El Secretario:

A mis padres, por haberme enseñado que puedo conseguir cualquier cosa,

A Miriam, por haberme dado siempre constante apoyo y cariño.

José Luis.

Acknowledgements

It is a pleasure to thank all those who made this Ph.D. thesis possible.

First of all, and foremost, I am deeply grateful to my advisor, Prof. Pascual Campoy, for his encouragement, guidance, and support gave during the last seven years. Thanks to his perfect combination of freedom and guidance, together with his constant aim to achieve the highest results, I had the opportunity to learn a lot of a variate number of fields of knowledge, to use my creativity as much as I wanted, and to develop my own thesis as I believed it was convenient, allowing me to achieve the excellence in the process. Furthermore, he always considered me as an essential part of the research group he leads, happily funding my research without having the need to ask for it.

Second, I am very thankful to my advisor Prof. Martin Molina. His essential guidance in the artificial intelligence based design of the proposed system architecture led to the final design proposed in this thesis. His perseverance contributed to having the presented software framework fully functional, being the name of *Aerostack*, cleverly proposed by him. Moreover, he helped me to significantly grow as a scientific in my final stage of the doctorate, thanks to his constant discussions and reviews that allowed me to clearly organize my thoughts and to formalize all my ideas.

It is specially important to mention the support of all the current members of the Computer Vision and Aerial Robotics (CVAR) group, including Jesús Pestana, who has contributed in half of this thesis; Prof. Paloma de la Puente, who has constantly taught me and guided me, never asking for anything in return; Carlos Sampedro, whose competitive spirit made the best of me; Adrian Carrio, whose capabilities of achieving several projects at the same time allowed me to think that impossible is nothing; Ramon Suarez, who helped me to minimize my stress and annoyances when appeared; Hriday Bavle, whose constant predisposition made many experiments possible.

Other members of the group, like Prof. Sergio Dominguez, Alejandro Rodriguez, Zorana Milosevic, and all the Master and Undergraduate students that have ever been part of the group, have as well helped me in many aspects to make this thesis possible.

Additionally, former members of the group, including Dr. Miguel Olivares, Prof. Ivan Mondragon, Prof. Carol Martinez, Dr. Changhong Fu, Ignacio Mellado, Dr. Aneesh Chauhan, and Dr. Jean-François Collumeau, among others, helped me to become what I currently am.

A special mention requires the help and the support of the other permanent and temporal staff of the Centre for Automation and Robotics (CAR), including Prof. Manuel Ferre, Prof. Rafa Aracil, and Prof. Enrique Pinto, for their patience with me while collaborating in their subject; all others professors for their valuable lessons; Angel Martinez, because his creativity and skills helped me while developing plenty of hardware components; and Carlos Sanchez, Rosa Ortiz, and Teresa Olmos for constantly helping me with all the paperwork, saving me from multiple troubles.

I also want to thank the Autonomous Systems Technologies Research & Integration Laboratory (ASTRIL) of the Arizona State University (ASU) - Arizona (USA), and specially to Prof. Srikanth Saripalli, for giving me the opportunity to do a research stay under his supervision, that gave me wonderful experiences in both personal and professional level; and Patrick McGarey and friends for adopting the little Spanish kid that once traveled to America.

Not less important is the acknowledgment to the Robotics and Interactions (RIS) group of the Laboratory

of Analysis and Architecture of Systems (LAAS) - Toulouse (France), and specially to Prof. Simon Lacroix, who made possible the two research stays there; Dr. Antonio Franchi, whose guidance during the stays helped me to learn a lot in fields like state estimation and control; Marco Tognon, Nicolas Staub, Dr. Markus Ryll, Dr. Anthony Mallet, Davide Bicego, Victor Arellano, and all the other members for their valuable help.

I would like to thank all the institutions that have funded this thesis. Specially, the Consejo Superior de Investigaciones Científicas (CSIC) for doctoral scholarship through the program "Junta de Ampliación de Estudios. Becas predoctorales 2010"; and the Campus France for the research stay scholarship thanks to the program "Eiffel Scholarship 2015".

I need to specially thank my parents, my girlfriend, my brother and sister, and my friends. I will be grateful forever with all of them for their constant support, patience, and confidence.

I also thank all the persons that, while not specifically mentioned, have supported the realization of this thesis.

Resumen

La navegación totalmente autónoma de una flota de robots aéreos realizando misiones dinámicas complejas en entornos desafiantes no estructurados, es necesaria para simplificar el uso de los vehículos aéreos y extender su utilización a un mayor número de aplicaciones, es necesaria . El desarrollo de un sistema inteligente multi-robot completamente autónomo sigue siendo un problema abierto con soluciones parciales e incompletas en robótica aérea, habiéndose sólo desarrollado algunas arquitectura abiertas para sistemas aéreos, las cuales presentan limitaciones en su nivel de autonomía y versatilidad.

Esta tesis presenta una arquitectura de sistema versátil para robótica aérea que permite el funcionamiento totalmente autónomo de un sistema multi-robot aéreo, cumpliendo con los requisitos de ser agnóstico a nivel de misión, plataforma y entorno. Se ha caracterizado a nivel abstracto y general, definiendo sus siete subsistemas, sus funcionalidades, y sus interfaces a alto nivel, garantizandose la versatilidad y flexibilidad. Los siete subsistemas propuestos son: Sistema de Extracción de Características, Sistema Motriz, Sistema de Conciencia Situacional, Sistema Ejecutivo, Sistema de Planificación, Sistema de Supervisión y Sistema de Comunicación. Esta arquitectura de sistema proporciona a los diseñadores una arquitectura inicial de la que partir cuando desarrollan sus propios sistemas inteligentes totalmente autónomos multi-robots aéreo. La validación de la arquitectura propuesta del sistema es una tarea compleja ya que el rendimiento del sistema completo depende de la misión, el entorno, la configuración del hardware, los algoritmos empleados y su implementación software. Tres tipos de escenarios se utilizaron con éxito para proporcionar una evaluación global del sistema completo, validando el desempeño de la arquitectura de sistema propuesta: (1) competiciones internacionales de robótica aérea, (2) desafíos auto-propuestos y (3) demostraciones públicas.

Además, la tesis presenta varios algoritmos que permiten un aumento del nivel de autonomía de los sistemas robóticos aéreos, desarrollados en el contexto de una aplicación particular. Estos algoritmos se pueden agrupar en: (1) percepción, (2) control, (3) planificación y ejecución de tareas, (4) inteligencia y cognición, y (5) comunicación e interacción. Todos estos componentes han sido evaluados individualmente, demostrando su desempeño. Sin embargo, su importancia aparece cuando se integran en el sistema completo. Los componentes propuestos más destacados, presentados con un alto nivel de detalle, son los siguientes: (1) detección y reconstrucción de helipuertos, (2) percepción basada en odometría y marcadores visuales con reconstrucción del entorno, (3) percepción basada en odometría y visión por computador para mapas con cuadrícula, (4) percepción basada en la fusión de múltiples sensores con reconstrucción del entorno, y (5) planificación de trayectorias libre de colision para entornos dinámicos.

La tesis presenta también un entorno de software de código abierto, llamado Aerostack, que facilita la implementación eficiente de la arquitectura de sistema y los algoritmos desarrollados por medio de componentes de software. Este software se organiza de forma modular mediante paquetes agrupados por su funcionalidad, sus dependencias y su ciclo de vida. El entorno de software propuesto se basa en el middleware ROS para la comunicación interprocesos y utiliza un paradigma asíncrono multiproceso en el que cada funcionalidad elemental se implementa como un solo proceso, facilitando el desarrollo y permitiendo un procesamiento distribuido. El entorno de software propuesto ha demostrado ser versátil y escalable, siendo los desarrolladores capaces de reutilizar tantos módulos de software como sea necesario, y de modificar o desarrollar nuevos módulos sin necesidad de adaptar otros componentes.

Abstract

Achieving a fully autonomous navigation of a fleet of aerial robots when performing complex dynamic missions in challenging unstructured environments is an essential requirement to simplify the use of micro aerial vehicles and to extend their utilization to a greater number of applications. The development of a multi-robot fully autonomous intelligent system is still an open problem with partial and incomplete solutions in aerial robotics, and only some open source architecture frameworks for aerial systems have been developed so far, which present limitations in their autonomy level and in their versatility.

This thesis presents a versatile system architecture for aerial robotics that enables the fully autonomous operation of an aerial multi-robot system and fulfills the requirements of being mission, platform, and environment agnostic. It has been characterized in an abstract and general level, defining its seven subsystems, their functionalities, and their interfaces in a top-level way that guarantees the versatility and flexibility. The seven proposed subsystems are: Feature Extraction System, Motor System, Situation Awareness System, Executive System, Planning System, Supervision System, and Communication System. This system architecture provides system designers the initial architecture for developing their own fully autonomous intelligent aerial multi-robot systems. The validation of the proposed system architecture is a complex task since the performance of the complete system is highly dependent on the mission, the environment, the hardware setup, the employed algorithms, and their software implementation. Three kinds of scenarios were successfully used to provide a global evaluation of the complete system, validating the performance of the proposed system architecture: (1) international aerial robotics competitions, (2) self-proposed challenges, and (3) public demonstrations.

In addition, this thesis presents several algorithms, with different level of detail, that yield to an increased level of autonomy of the aerial robotic systems, developed in the context of particular applications. These algorithms can be gathered in the following five groups: (1) perception, (2) control, (3) planning and task execution, (4) intelligence and cognition, and (5) communication and interaction. All these components have been evaluated isolatedly, demonstrating their individual performance. Nevertheless, their importance stands out when integrated into the complete system. The most important components presented in this thesis, analyzed with a high level of detail, are the following: (1) helipad detection and reconstruction for shipboard landing, (2) perception based on odometry and visual markers with environment reconstruction, (3) perception based on odometry and computer vision for gridded maps, (4) perception based on multi-sensor fusion with environment reconstruction, and (5) collision-free path planning for dynamic environments.

This thesis presents as well, an open-source software framework, called *Aerostack*, that facilitates a cost and time effective implementation of the designed system architecture and the developed algorithms by means of software components. This framework is modularly organized in software packages gathered by their functionality, their dependencies, and their life state. The proposed software framework relies on the widely used ROS middleware for interprocess communication and uses an asynchronous multiprocess paradigm where every elementary functionality is implemented as a single process, easing the development and allowing a distributed processing. The proposed software framework has demonstrated to be versatile and scalable, being the developers capable of reusing its software modules as needed, and modifying or developing new modules without adaptations of any other components.

Nomenclature

The conventions used in this document are defined as follows:

Vectors and Matrices

- Vectors are represented by bold lower case letters, such as $\boldsymbol{x}$ or $\boldsymbol{z}$.
- Vectors are represented as column vectors, unless indicated otherwise.
- $\boldsymbol{\nu}_A \times \boldsymbol{\nu}_B$ represents the cross product of the vectors $\boldsymbol{\nu}_A$ and $\boldsymbol{\nu}_B$.
- $\boldsymbol{\nu}_A \circ \boldsymbol{\nu}_B$ represents the dot product of the vectors $\boldsymbol{\nu}_A$ and $\boldsymbol{\nu}_B$.
- $\|\boldsymbol{\nu}_A\|$ represents the norm of the vector $\boldsymbol{\nu}_A$.
- Matrices are represented by bold capital letters such as $\boldsymbol{R}$ and $\boldsymbol{H}$.
- The transpose of a vector $\boldsymbol{v}$ and a matrix $\boldsymbol{R}$ is represented by $\boldsymbol{v}^T$ and $\boldsymbol{R}^T$ respectively.
- The inverse of a matrix $\boldsymbol{R}$ is represented as $\boldsymbol{R}^{-1}$.
- The pseudo-inverse of a matrix $\boldsymbol{R}$ is represented as $\boldsymbol{R}^{+}$.
- $\left[\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{v}}\right]$ represents the Jacobian Matrix of a the vector $\boldsymbol{u}$ with respect to the vector $\boldsymbol{v}$.
- $\left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{v}}\right]$ represents the Jacobian Matrix of a matrix $\boldsymbol{R}$ with respect to the vector $\boldsymbol{v}$.
- $\dot{\boldsymbol{v}} = \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t}$ represents the time derivative of the vector $\boldsymbol{v}$.

Points of the space

- Points of the space are represented by capital letters such as P and Q.

Reference frames and poses (Appendix A)

- Reference frames are represented by capital letters such as W and R.
- The pose of frame A in coordinates of frame B is represented by: p_A^B.
- $p_A^B \oplus p_B^C$ represents the pose composition of p_A^B and p_B^C.
- $p_B^A = \ominus p_A^B$ represents the inverse transformation of the pose p_A^B.
- All the reference frames are right-handed.

Vectorial quantities in different reference frames (Appendix A)

- $\boldsymbol{\nu}_{A|B}^C$ represents the vectorial quantity $\boldsymbol{\nu}$ (e.g. velocity or acceleration) of the frame A with respect to the frame B in coordinates of frame C.

Quaternions (Appendix B)

- Quaternions are represented by bold lower case letters, such as $\boldsymbol{q}$ or $\boldsymbol{p}$, following the Hamilton notation.
- $\boldsymbol{p} \otimes \boldsymbol{q}$ represents the quaternion product of $\boldsymbol{p}$ and $\boldsymbol{q}$.
- $(\boldsymbol{q})^*$ represents the conjugate quaternion of $\boldsymbol{q}$.

Estimated and measured variables (Appendix D)

- Estimated variables are sometimes represented with a hat, such as $\hat{\theta}_e$.
- Measured variables are sometimes represented using a tilde, such as $\tilde{d}$

Acronyms

Following is a description of most commonly acronyms referenced on this work:

AFM: Artificial Field Maps
ANN: Artificial Neural Network
API: Application Programming Interface
ArUco: Augmented Reality from University of Cordoba
CAR: Centre for Automation and Robotics (in Spanish, Centro de Automatica y Robotica)
CAR UPM-CSIC: Centre for Automation and Robotics, joint venture Technical University of Madrid, Spanish National Research Council (in Spanish Centro de Automatica y Robotica, centro mixto Universidad Politecnica, Consejo Superior de Investigaciones Cientificas)
CLI: Command Line Interface
CSIC: Spanish National Research Council (in Spanish, Consejo Superior de Investigaciones Cientificas)
CV: Computer Vision
CVAR: Computer Vision and Aerial Robotics Group
CVG: Computer Vision Group, former name of the Computer Vision and Aerial Robotics Research Group
EKF: Extended Kalman Filter
EKF-SLAM: Simultaneous Localization And Mapping based on an Extended Kalman Filter
ESC: Electronic Speed Controller
FCS: Flight Control System
FOV: Field Of View
GA: Genetic Algorithm
GCS: Ground Control Station
GNSS: Global Navigation Satellite System
GPS: Global Positioning System
GS: Guidance System
GUI: Graphical User Interface
HMI: Human Machine Interface
HMM: Hidden Markov Model
HRI: Human Robot Interaction
HSV: Hue-Saturation-Value
HW: Hardware
IARC: International Aerial Robotics Competition
IMAV: International Micro Air Vehicle competition
IMU: Inertial Measurement Unit
JCR: Journal Citation Report
KF: Kalman Filter
LIDAR: LIght Detection And Ranging
LTI: Linear Time Invariant
MAV: Micro Air Vehicle
MLP: Multi-Layer Perceptron
MOCAP: Motion Capture System
MSF: Multi-Sensor Fusion
MTOW: Maximum Take-Off Weight
NUI: Natural User Interface
NS: Navigation System

PD: Proportional-Derivative
PF: Particle Filter
PFM: Potential Field Map
PID: Proportional-Integral-Derivative
PRM: Probabilistic Road Map
RANSAC: RANdom SAmple Consensus
RGB: Red-Green-Blue
RGB-D: Red-Green-Blue-Depth
ROS: Robot Operating System
RPA: Remotely Piloted Aircraft
RPAS: Remotely Piloted Aircraft System
SLAM: Simultaneous Localization And Mapping
SW: Software
UAS: Unmanned Aerial System
UAV: Unmanned Aerial Vehicle
UPM: Technical University of Madrid (in Spanish, Universidad Politecnica de Madrid)
UI: User Interface
VSLAM: Visual Simultaneous Localization And Mapping
XML: Extensible Markup Language

Contents

List of Figures

List of Tables

Part I

INTRODUCTION AND STATE OF THE ART

Chapter 1

Introduction and Objectives

1.1. Introduction to aerial robotics

Robotics is a multidisciplinary branch of engineering and science that deals with the design, construction, operation, and use of robots[1]. The recent advances in robotics have duplicated the size of the robotics market in the last few years. The last reports in robotics forecast an exponential growing of the robotics market during the following years[2]

Robots are classified according to the environment in which they move in Unmanned Ground Systems (UGS), Autonomous Underwater Systems (AUS), and Unmanned Aerial Systems (UAS). Hybrid robots designed to move in different environments are starting to appear as a very versatile solution, but they still represent a negligible percentage.

Unmanned Aerial Systems (UAS), also called Remotely Piloted Aerial Systems (RPAS) or simply drones, gathers all the robots that mainly perform their operation in the air (nevertheless, aerial robots might include a small part of their operation in other environments like the ground when performing, for example, the taking off and the landing).

UAS are formed by three main parts: (1) the aerial platform, (2) the ground control station, and (3) the communication segment. The aerial platform (also called Unmanned Aerial Vehicle, UAV; or Remotely Piloted Aircraft, RPA) represents the vehicle itself, including the lifting system, the mechanical structure, the power system, the electronics (sensors and onboard computers), and all the payload. The ground control station (GCS) is a land- or sea-based control center that provides the facilities for human control of UAVs. The communication segment (CS) allows the exchange of information between the aerial platform, the ground control station, and the other agents (i.e. vehicles) at the system.

The UAVs can be classified, according to (Cuerno Rejado et al., 2015), and (Gupte et al., 2012), depending on their density in: (1) lighter-than-air (e.g. zeppelins and balloons), and heavier-than-air. Heavier-than-air aircrafts are divided according to its lifting system in (1) fixed-wing, (2) rotatory-wing, (3) flapping-wing, and (4) hybrid. There exist four kinds of rotatory-wing vehicles according to their rotor(s) configuration: (1) helicopter-type, (2) cyclogyro-type, (3) autogyro-type, and (4) gyrodyne-type. Helicopter-type aircraft include multirotor UAVs.

Other classifications of the aerial vehicles might be done according to their range (short, large), altitude (low, high), endurance (short, long), or Maximum Take-Off Weight (MTOW) and size (micro, small, medium, large).

[1] Online: `https://en.wikipedia.org/wiki/Robotics`

[2] According to the International Federation of Robotics (IFR). Online: `http://www.ifr.org/`

This thesis focuses on micro and small size (< 5 Kg, < 2 m), low flight altitude (< 300 m), short range (< 10 Km) multirotor-type UAVs, but a great number of the presented results are easily extensible to other kinds of robotic systems (including ground and underwater).

The level of autonomy of a robot is another essential indicator of the robot's capabilities. All the works by Huang, (Huang et al., 2005a), (Huang et al., 2005b), (Huang, 2007), (Huang, 2008a), and (Huang, 2008b), propose a framework, called Autonomy Levels for Unmanned Systems (ALFUS), with specific metrics for the determination of the level of autonomy of Unmanned Systems. Some other works, like (Beer et al., 2014), propose a framework, called Levels of robot autonomy (LORA), with a special focus on the relationships between the robots and the humans depending on the level of autonomy of the robots.

This thesis targets the highest level of autonomy, where the robots are able to complete, without human intervention, a complex dynamic mission in a changing environment with interactions between all the robots and the humans.

1.2. Motivation of the thesis

The market of the autonomous micro aerial vehicles is growing, and many challenges and opportunities are appearing. According to (Kumar and Michael, 2012), the four main fields of interest are (1) design, including mechanical, new materials, electronics, scaling of size, weight, and power, etc; (2) control, including adaptation and learning; (3) planning; and (4) state estimation and perception. Being, the boundaries of the four topics clearly defined. In addition to these four fields of interest, (Kumar and Michael, 2012) proposes two challenges that involve the first four fields: (1) physical interaction, including grasping and manipulation; and (2) adaptation to complex environments with changing dynamics.

Most of the existing research groups focus their effort on solving one or two of these challenges at the same time. Thanks to this concentrated effort, the state of the art in the previously enumerated topics has been strongly pushed, turning most of them into a reliable and fully functional truth, although still with some limitations.

Nevertheless, there is still a remainder open challenge that has not been deeply tackled for aerial robots. This challenge requires the integration of all the previously cited topics to create a safe and reliable fully autonomous system that might be formed by one or more robots performing a complex dynamic mission interacting between them and the elements of the environment, and moving in an unstructured changing environment.

This is a big challenge that cannot be avoided and requires to be tackled to push the research on unmanned aerial systems and to evaluate the real needs and the current limits of every of the previously defined individual fields of research. Having this functional fully autonomous system of aerial robots will make the market of the aerial robotics to grow without boundaries, enabling their usage in many new applications.

The current applications of aerial robotics, apart from the military ones, range from: aerial photography and filming, industrial inspection (power line; windmill; structures such as bridges or pipelines; building; etc.), crop survey and precision agriculture, firefighting, environment and geological monitoring, surveillance and patrolling, wildlife survey, etc.

Most of these applications use the aerial robot with a low level of autonomy, mostly teleoperated or commanded through waypoints. Additionally, the environment is usually outdoors (what enables the use of the GPS), and the environment is structured (wide open areas without any moving obstacle), being the interaction between the aerial robot and the environment only non-physical.

Other more advanced applications that are still under research, include for example: package delivery, search and rescue, medical assistance, usage in smart cities (Mohammed et al., 2014), creation of mobile networks (Sahingoz, 2013), etc. These applications require the aerial robot to have a higher level of autonomy (taking decisions, and solving complex problems in a safe and reliable way), being able to move in GPS-denied environments with unstructured environments (small previously unknown areas with moving obstacles). The physical interaction between the aerial robot and the environment is still very limited (only package dropping), although the non-physical interaction is very challenging.

Future challenging applications like construction, require the highest level of autonomy, moving in complex unstructured environment, with multiple physical and non-physical interactions.

1.3. Problem statement

The safe and intelligent fully autonomous navigation of a robotic system formed by one or more aerial agents requires a multidisciplinary work that has to successfully integrate and combine all the following topics that are tightly related with many interdependencies:

System design

In the first place, the complete system has to be properly designed. This design has to take into account the objectives that the system has to achieve, the environment where the system is going to be developed and the technological, economical and temporal restrictions.

The system design is an iterative process that requires creating different designs that evolve in the time. The initial designs are more abstract and general, responding to top-level functionalities. These initial designs describe the subsystems and the general interfaces between the subsystems. The final designs concrete the subsystems, their functionalities, and their interfaces.

Hardware

The aerial platforms (also called aerial vehicles) together with their payload are required to be properly selected, designed, mounted and configured for a particular operation. The payload onboard an aerial platform range from the platform frame, its batteries and power system, the onboard sensors, the onboard computers, and the onboard actuators. Offboard elements like ground sensors (e.g. a motion capture system) might be used, but highly limit the operation capabilities of the aerial robots and therefore, their usage is highly discouraged and omitted in this thesis. In addition to the aerial platform, the ground control station (GCS) where the human operators monitor and interact with the aerial robotics system is required. Other indispensable part of the hardware is the communication system that allows the communication between aerial platforms and human operators.

Perception: feature extraction, state estimation and environment reconstruction

The environment where the aerial robots are operating requires being properly perceived. The perception of the environment is done through the sensors, and therefore the perception algorithms are highly dependent on the sensor setup.

This thesis does not limit the sensors used for perceiving the environment, but a predilection on cameras is evident in all the solutions adopted. This preference for cameras is given by the fact that cameras are cheap, lightweight, and provides a very rich information, making them perfect for their use onboard aerial vehicles. The main drawbacks of cameras are that they require a computational expensive processing of the provided information, and require the environment to have some particular characteristics (enough but not too much light, some specific features, smoke is not allowed, etc.).

Typically, sensors provide a noisy information that does not represent completely the environment or the robot state, or this information is not in a format that can be easily usable. State estimation algorithms where the information coming from different sensors is combined to obtain the complete state of the robot are therefore needed. Additionally, mapping algorithms that compute a usable representation of the environment are required in order to be able to operate in such environment.

Control

The stabilization of the aerial platform, together with the transformation of references such as trajectories, positions, or velocities into actuator commands, ensuring that the commanded reference is followed by the aerial platform constitutes the control problem. These control algorithms are highly dependent on the aerial platform (because of its actuators) but also on the perception algorithms that indicates the feedback used to ensure that the controller is following the commanded reference.

Planning and task execution

The creation of optimum plans for achieving different goals is essential to allow the aerial robot to execute complex missions in complex environments. The division of a complex mission in smaller tasks and actions and its posterior sequencing into control references is as well needed to connect the high-level planning and reasoning with the controllers and the actuators. Planning and task execution algorithms are highly dependent on the application, the admissible control references, and the available information of the environment.

Intelligence and supervision

To allow the adaptation of the aerial robot to environment changes, being able to react to them by performing different actions, a higher level intelligence and environment supervision is required. Without this feature, the robot will only be able to work in structured environments, executing sequential missions. Additionally, the aerial robot requires being capable of detecting performance failures and reconfiguring the system to safely complete the commanded mission or to abort it in the event of a critical failure. A self-supervision is therefore essential.

Multi-agent interaction

The previous paragraphs enumerated the requirements to obtain a fully autonomous intelligent robot that is able to operate in a dynamic unstructured environment executing complex dynamic missions.

A still missing requirement is the capability to operate in a multi-robot environment, where the robots have to interact between them in a safe, intelligent and reliable way.

The robots have to interact as well with human agents because they have to share with them the same environment safely, and they have to perform collaborative tasks with them.

Software implementation

The realization of the designed and developed algorithms is mostly done by means of software components. An appropriate software implementation is required to be able to efficiently execute these algorithms. Besides the efficiency, the software implementation has to fulfill other criteria such as modularity, versatility, and scalability, to enable the sustained usage over the time of the software components, together with the ease of its maintenance.

System integration, testing and evaluation

Last but not least, the complete system (including hardware and software components) needs to be properly integrated, proven, and easily replicable for different applications, and for different kinds and number of agents, reducing the development time and ensuring the usability and versatility of the system.

The assessment of the achieved performance enables the iterative improvement process of the system.

1.4. Objectives

The main objective of this thesis is summed up in one single sentence: enable and facilitate the fully autonomous intelligent navigation of a heterogeneous aerial multi-robot system for different complexity and kinds of missions and environments.

In order to fulfill such general objective, three more specific objectives can be considered:

- Design of a novel versatile system architecture for aerial robotics that allows the fully autonomous operation of an aerial multi-robot system. This system architecture has to be flexible and therefore has to be specified as mission, platform, and environment agnostic. It has to be characterized in an abstract and general level, defining only to top-level functionalities. To do so, it requires:
  - Definition of its main subsystems and their functionalities. Subsystems should be defined along natural boundaries. Subsystems should be defined to minimize the amount of information to be exchanged between the subsystems. Well-designed subsystems send finished products to other subsystems. The subsystems and their functionalities have to be defined in a top-level way, to guarantee the versatility and flexibility of the system architecture.
  - Specification of the interfaces between subsystems. These interfaces have to be unequivocally defined, avoiding ambiguity that might lead to errors and malfunctioning. Nevertheless, they have to be general enough to respond to the requirements of mission, platform, and environment agnostic.

  This system architecture must provide system designers the initial architecture for developing their own fully autonomous intelligent aerial multi-robot systems, guiding them through this complex process and speeding up the development of aerial robotic systems.
- Development of different algorithms to increase the autonomy level of the aerial robotic systems. To enable the autonomous operation of aerial multi-robot systems, concrete functionality algorithms, that aim to solve a particular problem, are needed. These algorithms might range from:
  - Perception algorithms, including feature extraction, state estimation and mapping algorithms for different applications and robots ensuring the adequate estimation of the state of the robot and the environment.
  - Control algorithms for different applications and robots, that ensure that the commanded references are correctly tracked and transformed into actuator commands.
  - Planning and task execution algorithms for different applications and robots that permit the development of complex missions in complex environments.
  - Intelligent and cognitive algorithms that ensure the adaptation of the robots to changing environment, system failures, or dynamic missions.
  - Social and interaction capabilities between robotics agents and human agents, guaranteeing the safe and efficient coexistence and cooperation.
- Development of a software framework that facilitates the efficient implementation of the designed system architecture and the developed algorithms by means of software components. A good software framework

has to be modular to allow the creation of independent modules with specific functionalities which can be exploited once connected to the rest of the components. Versatility is desired to enable developers to reuse as many software modules as possible, reducing the development time and cost. The software framework needs, as well, to be scalable, permitting to modify or to develop new modules without the need of changing any other components. Finally, to reach the largest possible number of people, contributing to the scientific community, the software framework is desirable to be open-source.

1.5. Methodology

The methodology followed in this thesis is based on an iterative process that comprises the following stages:

- *Challenge*: in this first stage, a particular application related to aerial robotics is proposed to be solved. The complexity of the challenge increases with the number of iterations. The challenges can be given externally as international aerial robotic competitions or internally as proposed experiments.
  The aerial robotic competitions present the following features: Their objectives are determined by the competition organizers and therefore they cannot be modified or adapted. Additionally, international aerial robotic competitions do not admit flexibility in the deadline, requiring to have a reliable working solution by the competition day.
  The internally proposed challenges have the following features: Their objectives might be adapted to the particular research line that is wanted to be investigated, with an unlimited level of difficulty. Nevertheless, they do not present a hard deadline, being the reliability a minor requirement.
- *Proposal*: the second stage comprises the design, development, implementation and integration of a solution capable of solving the existing challenge. The initial solution must be based on the solution provided by the previous iteration, trying to reuse the larger amount of existing modules. This stage is, as well, iterative, where the final solution adopted is obtained gradually following a top-down strategy.
- *Evaluation*: once the final solution that solves the proposed challenge is obtained, the evaluation stage starts. This stage consists of put down the proposed solution to different tests to obtain different metrics of its performance. These tests are two kinds: simulations and real experiments.
  Simulations are carried out in the first steps of the evaluation and provide a general overview of the performance of the proposed solution under controlled conditions and being capable of isolated some elements. This kind of tests allows ignoring some technological and economical problems, to concentrate on the difficulties of the adopted solution.
  Real experiments are required on every system that is supposed to work in the real world. This kind of tests evaluates the proposed solution in the presence of technological and economical limitations.
- *Analysis*: To determine the degree of success in the achievement of the proposed challenge, the analysis of the results of the evaluation has to be done. This stage has to provide objective opinions on the performance of the proposed solution with an exhaustive exam of the data provided by the evaluation stage. To do so, it is essential to fully understand the evaluation methodology and the challenge objectives.
- *Generalization*: The last stage requires reformulating the proposed solution in a way that the performance of the system would be improved, by taking into account the analysis of the evaluation results. This reformulated solution is going to be the initial solution to the proposed challenge in the following iteration.

This methodology will eventually end up, after several iterations, on a refined model capable to successfully cover a wide range of applications.

The methodology used in this thesis is based on a system engineering approach[3], that comprises the following tasks, summarized with the acronym SIMILAR, (Bahill and Gissing, 1998): Statement of the problem, Investigation of alternatives, Modeling of the system, Integration, Launching the system, Assessment of the performance, and Reevaluation.

1.6. Contributions

The objectives of this thesis are very wide and challenging. To successfully achieve them, the collaboration of the complete multidisciplinary team of people that belong to the research group of the thesis author was indispensable. The contributions of this thesis are therefore multiple, included in a wide variety of fields of knowledge, with a different nature, and with a different level of importance. The contributions of this thesis are gathered in the following three groups:

[3]Online: `http://www.incose.org/AboutSE/WhatIsSE`

Versatile system architecture for aerial robotics

First of all, this thesis presents a versatile system architecture for the multiple heterogeneous aerial robots operation ensuring its fully autonomous intelligent performance in different complexity and kinds of missions and environments without committing to any specific hardware or platform.

The system architecture defines the top-level subsystem of the robotic system, avoiding loss of generality. A top-down design methodology was used, breaking down the system in several simpler subsystems, defining the specifications and functionalities of every subsystem and guaranteeing the compatibility and relationships between subsystems.

To achieve the present version of the system architecture, an iterative process has been carried out. Every iteration of the system architecture was done to solve a more complex challenge that was faced through its usage in several international competitions, experiments, and public demonstrations.

This system architecture provides system designers the initial architecture for developing their own fully autonomous intelligent aerial multi-robot systems, guiding them through this complex process and speeding up the development of robotic systems.

Algorithms to increase the autonomy level of aerial robots

The second contribution of this thesis is the development of several algorithms that contribute to increasing the autonomy level of aerial robotic systems. These algorithms implement concrete functionalities to solve a particular problem and they are therefore highly dependent on the hardware setup, the environment, and the mission.

These developed algorithms implement functionalities related to perception, state estimation, and mapping; control; planning and task execution; intelligent and cognitive behavior; and social behavior and multi-modal interaction. In this thesis, several perception, state estimation, and mapping algorithms, and a path planning algorithm are deeply presented.

After the theoretical development of every algorithm, a following software implementation stage is carried out. The software components are tested by means of simulations and real experiments, evaluating its performance to iteratively improve them.

Open-source software framework for aerial robotics

The last contribution of this thesis is an open-source software framework for aerial robotics that efficiently implements the proposed system architecture and the developed algorithms.

The presented software framework includes features such as modularity, versatility, and scalability, ensuring the sustained usage over the time of the software components, easing their maintenance. Moreover, the fact of being open-source benefits the whole scientific community, contributing to speed up the development of the aerial robotic systems.

1.7. Outline

The present thesis document is organized into four parts, thirteen chapters, and five appendices, as follows:

- Part I, formed by two chapters, introduces and motivates the thesis, doing a review of the related state of the art.
  - Chapter 1, the present one, does an introduction to aerial robotics, motivates the thesis and sums up the main objectives and contributions.
  - Chapter 2, does a review of the state of the art related to this thesis.
- Part II describes in three chapters the proposed system architecture and software framework for aerial robotics.
  - Chapter 3 describes the system architecture proposal for aerial robotics, together with its general features, indicating its historical evolution to arrive at the current design.
  - Chapter 4 provides deep details of the seven subsystems of the previously introduced system architecture for aerial robotics.
  - Chapter 5 introduces the proposed software framework, called Aerostack, as an implementation of the previously presented system architecture, created to be modular, reusable and open-source.
- Part III covers several proposed solutions by the author of the thesis to increase the level of autonomy of aerial robots. It is organized in the following five chapters:
  - Chapter 6 proposes a computer vision based algorithm for the detection and reconstruction of helipad marks for the shipboard landing.

    - Chapter 7 proposes a complete perception solution based on odometry and visual markers with environment reconstruction.
    - Chapter 8 proposes a complete perception solution based odometry and computer vision, specifically designed for gridded maps.
    - Chapter 9 proposes a complete versatile perception solution based on multi sensor fusion with environment reconstruction.
    - Chapter 10 proposes a collision free path planning solution for dynamic environments.
- Part IV evaluates the proposals of the thesis, showing the achieved results that validate them. This part also concludes the thesis and enumerates some future works.
    - Chapter 11 briefly presents other algorithms to increase the level of autonomy of aerial robots that were implemented for the validation of the proposed system architecture and software framework, and that can also be seen as illustrative examples of the abstract components of the proposed system architecture.
    - Chapter 12 discusses the results achieved by the thesis, including qualitative and quantitative results.
    - Chapter 13 enumerates the conclusions and the open future works.
- The five existing appendices are:
    - Appendix A has a general overview of the existing methods for representation of rotations and poses.
    - Appendix B provides a detailed analysis of the algebra of quaternions.
    - Appendix C gathers the kinematic equations of a rigid body expressed by means of different representation methods.
    - Appendix D describes the equations of the very well-known extended Kalman filter for both direct and indirect formulations, including simultaneous localization and mapping capabilities.
    - Appendix E includes a list of the scientific contributions as a consequence of this thesis.

Chapter 2

State of the Art

This chapter does a small review of the state of the art related to this thesis. First of all, in Section 2.1, the most famous aerial robotics architectures and frameworks are analyzed. Section 2.2 briefly reviews some available hardware components for aerial systems. The main information related to the software middlewares and simulators is cited in Section 2.3. The following sections sum up the state of the art of the main components required on the proposed system architecture, including: feature extraction from sensor information (Section 2.4), state estimation, mapping, and sensor fusion (Section 2.5); control (Section 2.6); planning (Section 2.7); and human-robot interaction (Section 2.8). Section 2.9 shows some examples of complete unmanned aerial systems. Finally, Section 2.10 discusses the limitations of the reviewed state of the art.

2.1. Aerial robotics architectures and frameworks

Most of the research works on aerial robotics are concentrated in solving individual challenges such as control, state estimation and perception, or planning, as analyzed in the following sections. To test their researches, they provide custom system architecture whose objective is limited to the validation of the work they are proposing and they are not aiming to develop a versatile fully autonomous system architecture and framework for complex missions and environments.

Some years ago, the development of a complete system architecture and framework for unmanned aerial systems was tightly related to the hardware used and therefore none general-purpose system architectures existed. (Lim et al., 2012) does a survey, analyzing eight very well known open-source projects (OSP) on unmanned aerial systems. These OSP present a complete solution that forced their users to adopt their hardware setup, to follow their system architecture design, and to employ the components they have developed (e.g. their own controllers and state estimators) with their own software implementation. Despite being open-source projects, the user freedom when aiming to modify and interchange elements, was very limited.

In the recent years, this tendency has been inverted and two very well-known open projects are leading the market of Unmanned Aerial Systems, where they provided hardware and software elements that are fully exchangeable:

- "PX4" project [1] (Meier et al., 2015)
- "ArduPilot Mega (APM)" project [2]

[1] Online: `http://px4.io/`
[2] Online: `http://ardupilot.com/`

Both projects provide the following similar elements:

- The hardware of the autopilot, and some extra sensors and communication elements.
  The open hardware autopilots proposed by these two projects are, respectively: "PX4 FMU / PixHawk" [3]; and "ArduPilot Mega (APM)"[4].
- The software executed onboard, called "Flight Stack", that includes:
  - Firmware of the autopilot, sensors and other electronics. Including the basic components that do not require a high computational power and are run on limited embedded computers.
  - Other software executed onboard. Here are included complex algorithms executed on more powerful (onboard) computers, such visual SLAM algorithms, or collision avoidance algorithms.
  - Middleware and communications. This group of software includes all the interprocess communication protocols.

  The open-source flight stack proposed by these two projects are called, respectively: "PX4 Flight Stack", see (Meier et al., 2015); and "APM Flight Stack".
- The Flight Control and Mission Planner, executed on the GCS.
  The open-source flight controllers and mission planners proposed by these two projects are, respectively: ArduPilot Mega Mission Planner; and QGroundControl [5].
- Simulators.

The interchangeability of the components of these flight stacks, and flight controllers and mission planners is due to the creation of a (semi-) standard communications protocol, called MAVLink[6]. This protocol is not only followed by the PX4 and APM projects, but also for many other open and commercial projects and platforms, becoming nowadays in a de facto standard.

The "Linux Foundation" is leading a project, called "Dronecode" project[7], that has the objective of joining all the open-source projects on UAS, support them, and to create standards ensuring the future collaboration.

The level of autonomy that could be achieved by these two projects is not theoretically limited, but the current status of their flight stack provides a low level of autonomy, being indicated only for manual, and semi-autonomous missions. One of the limitations are due to the need of having a human operator has to supervise the flight to take the control in case of failures or unexpected changes. The second limitation is the kind of missions that are targeting, being mostly a navigation in outdoor environments (GPS-enabled) following the actions (e.g. take-off, or land) and the waypoints commanded by the human operator (provided by the mission planner components).

Nevertheless, many companies and research groups are working to create aerial systems with a higher level of autonomy by developing new components of these flight stacks.

Focusing now on the open system architectures and frameworks for fully autonomous UAS provided by research centers, their number is also very limited, highlighting the following:

- "Pixhawk" project[8] from CVG - ETHZ. This project started as a research project but evolved in the previously mentioned PX4 project.
- The "Paparazzi" project [9] with specific hardware and open-source software, was developed and used by ENAC and MAVLAV - TUDelft (see (Brisset et al., 2006)). Being one of the first that appeared, this project includes not only the software framework but also the hardware autopilot and sensors and it is not compatible with many available commercial hardware components. It does not use, as well, many widely used features and components like ROS, what limits its capabilities.
- The "asctec_mav_framework" [10], developed by ASL - ETHZ, has a special focus on autonomous navigation of Ascending Technologies aircrafts and it is not compatible with any other aircraft platforms.
- The "hector_quadrotor" [11] framework, developed by HECTOR - TU Darmstadt (see (Kohlbrecher et al., 2014)), focuses on heterogeneous cooperation for search and rescue tasks.
- The "telekyb" [12] framework, developed by HRI - MPI (see (Spica et al., 2013; Grabe et al., 2013)). It focuses on the development of bilateral teleoperation systems between human interfaces (e.g., haptic force feedback devices or gamepads) and groups of quadrotor Unmanned Aerial Vehicles (UAVs). Although it is very powerful, the main drawback is its rigid architecture that even allowing to exchange for modules

[3]Online: `https://pixhawk.org/`
[4]Online: `http://www.ardupilot.co.uk/`
[5]Online: `http://qgroundcontrol.com/`
[6]Online: `http://mavlink.org`
[7]Online: `https://www.dronecode.org/`
[8]Online: `https://pixhawk.ethz.ch/`
[9]Online: `http://wiki.paparazziuav.org/wiki/Main_Page`
[10]Online: `http://wiki.ros.org/asctec_mav_framework`
[11]Online: `http://wiki.ros.org/hector_quadrotor`
[12]Online: `http://wiki.ros.org/telekyb`

with similar functionalities depending on the user's need, it does not allow the user to easily change the architecture design for new capabilities.
- "Twirre" (Van de Loosdrecht et al., 2014) architecture, developed by NHL Computer Vision proposes a hardware and software design. It is focused mainly on hardware and it does not report a high level of autonomy.

2.2. Hardware: aerial platforms, autopilots and sensors

Sensors

There exist multiple kinds of sensors that can be used in Unmanned Aerial System.

Depending on where the sensors are located with respect to the aerial vehicle, they can be classified in, onboard, if they are attached to the aerial vehicle; and offboard, if they are not attached to the aerial vehicle, and they are normally fixed to the ground. Due to the limited size, payload, and energy available on the aerial vehicles, specially multirotors, onboard sensors have to be small, lightweight and with a limited energy consumption. Offboard sensors limit the working area of the aerial robot, and therefore, to support the autonomy level, onboard sensors are preferred along this thesis.

According to the kind of measurement, the sensors can be classified in: proprioceptive, which measure values internal to the robot; and exteroceptive, that acquire information from the robot's environment.

Depending on the energy measured, the sensors can be classified in: passive, that measure the environmental energy entering in the sensor; and active, that emit energy into the environment and measure the environmental reaction.

The following lines list the most widely used sensors on aerial robotics:
- Inertial Measurement Units (IMU), formed by accelerometers and gyros. These proprioceptive devices allow measuring their linear acceleration and the angular velocity. Without any exception, every aerial robot is equipped with at least one IMU. There exist different technologies for IMUs, but the cheap and small electro-mechanical ones are the dominant technology for micro and small aerial robots. These sensors, specially accelerometers, are characterized by their large noise and variable drift in the measurement.
- Magnetometers measure the magnetic field of their surroundings. Without the presence of any other source of magnetic interferences, they are used onboard to measure the Earth magnetic field to be able to have a global estimation of their attitude. Some elements, such a metal beams or electric motors, create different magnetic fields that can corrupt the measurement of the Earth magnetic field.
- Pressure sensors or barometers. They measure the air pressure, and they are used onboard to have an estimation of the altitude of the aerial robot with respect to the ground level. When the pressure changes due to the wind, or in indoor environments, their altitude estimation might be affected.
- Global Navigation Satellite System, such as GPS, GLONASS, or GALILEO provide an estimation of the position of the robot with respect to the Earth reference frame. It is only working outdoors, and their precision and rate are very limited.
- Ranging sensors such as Light Detection and Ranging (LIDAR) sensors or ultrasound sensors. These active exteroceptive devices measure the distance from these sensors to the objects of the environment. Depending on the information received by the environment, they can be 1D, 2D or 3D. The Hokuyo Range Finders[13] are lightweight 3D LIDARs that widely used on small aerial robots.
- Cameras are passive exteroceptive sensors that measure the visible light intensity of the environment. They are called RGB cameras if they measure the intensity of each of the red, green and blue visible light spectrum. The provided information is a discrete 2D array with integer values. These sensors are cheap and lightweight and provide a very rich information that normally cannot be used directly without processing it.
- Thermal and infra-red (IR) cameras are special kinds of cameras that measure other ranges of the electromagnetic spectrum.
- Depth cameras (also called RGB-D cameras or Ranging cameras) are a relatively new kind of sensors that using different technologies like structured pattern and time-of-flight, are able to obtain an RGB image with an associated depth value for each pixel. Examples of this special kind of ranging sensors (Cruz et al., 2012) are the Microsoft Kinect (Zhang, 2012), or the Intel RealSense.
- Motion Capture Systems, such as Vicon[14] and OptiTrack[15] are active offboard sensors formed by IR

[13]Online: `http://www.hokuyo-aut.jp/02sensor/index.html#scanner`
[14]Online: `https://www.vicon.com/`
[15]Online: `http://optitrack.com/`

cameras that are able to fast and accurately recover the position of markers that reflect a projected infrared light.

- Optical-flow sensors. These sensors are able to recover, by means of a camera, the existing optical-flow. Some sensors, like the px4flow (Honegger et al., 2013), include an IMU and an ultrasound sensor, being able to estimate the ground linear velocity by assuming that they are aiming to a flat static floor.

Many other less common sensors like event-based cameras or force sensors have been omitted from the previous list.

Aerial Platforms and autopilots

Different solutions might be adopted when selecting an aerial platform, particularly multirotors, for a specific application, depending on the user's needs and budget.

In the one hand, there exist some specialized companies that manufacture custom small scale production aerial vehicles, depending on the client needs. These companies typically provide a complete high-cost solution, normally designed and manufactured by them. These companies were very common at the early days of the aerial robotics, but now, other more cost effective solutions are dominating the market.

To reduce the cost of the aerial vehicles, some companies design and manufacture aerial vehicles on a large scale production. These aerial vehicles are designed for a particular application (e.g. aerial photography, precision agriculture, toy, etc.) and only a small customization using their own components is possible. These platforms work out of the box with simple software applications, extending their usage to non-expert users. Thanks to these companies, the aerial robotics has experienced a big growing in the last years.

Some other companies have developed mid-scale production aerial robots for research. These platforms allow an easy and flexible customization and provide APIs to develop custom applications.

Finally, the new trend in aerial robotics is that the final users design and mount their aerial robots customizing them for their particular needs. This is possible thanks to new cheap manufacturing techniques like 3D printing, and thanks to the development of interchangeable and easy to use components for the aerial platforms. This allows the development of cost effective aerial platforms for exploring a wide range of new applications.

The two companies leading the market of multirotors are the Chinese DJI[16] and the French Parrot[17].

DJI has focused on the large-scale production of aerial platforms for professional applications such as aerial photography, precision agriculture or inspection, providing closed solutions with some customization possibilities. Nevertheless, DJI has now a set of products targeting the market of aerial robots for research, allowing a complete customization of their platforms and releasing an API, allowing the development of custom applications.

Parrot has concentrated their efforts in developing large scale produced cheap and safe personal drones and toys. Parrot does not allow a wide range of hardware customization to their platforms, but they have released several APIs permitting the users to develop applications for their aerial robots. The two most famous aerial robots from Parrot are the following:

- Parrot AR.Drone (Bristeau et al., 2011). This 250 €aerial platform has two cameras (a front camera and a bottom camera), together with an IMU, a magnetometer, and a down-looking ultrasound sensor. It includes an autopilot that estimates the aerial robot attitude (by means of the IMU and the magnetometer), its flying altitude (thanks to the ultrasound sensor), its horizontal linear velocity (calculating the optical flow with the bottom camera), and its angular velocities (filtering the IMU). The autopilot allows to command the vehicle in attitude mode for the horizontal angles (pitch and roll), and in velocity mode for the vertical angle (yaw) and for the vertical movement (z). The autopilot includes as well some actions (and the required controllers) like take-off, hover and land, and can be connected to other devices by means of a Wi-Fi connection.
- Parrot Bebob. This 400 €aerial platform is an upgraded version of the Parrot AR.Drone with better performance and sensors.

The German company Ascending Technologies Gmbh (AscTec)[18], currently owned by Intel, initially developed platforms for research such as:

- AscTec Pelican (quadrotor), or AscTec Hummingbird (hexarotor). Both equipped with the Asctec Autopilot, an autopilot board that stabilizes the vehicle using the information from IMU, pressure altimeter and magnetometer fused by means of a Kalman Filter, providing as well an estimate of the aerial robot attitude, its angular velocities, and its vertical altitude. This autopilot, includes two different processors, the Low Level (LL) and the High Level (HL), being the last one connectible to a companion computer by means of a serial cable. The embedded controllers are executed on the LL processor and are closed

[16]Online: `http://www.dji.com/`
[17]Online: `https://www.parrot.com/en/Drones`
[18]Online: `http://www.asctec.de/`

and unmodifiable but their gains are tunable. The autopilot allows, among other modes, to command the vehicle in attitude mode for the horizontal angles (pitch and roll), in velocity mode for the vertical angle (yaw), and the average thrust command given by its motors.

- AscTec Neo (hexarotor). This hexarotor is a new version of the AscTec Hummingbird. It is equipped with a new autopilot, called Asctec Trinity with better performance and characterized for having triple redundancy.

In the last years, AscTec has developed products for professional applications such as the AscTec Falcon (equipped with the Asctec Trinity autopilot).

Another German company, Mikrokopter[19], has focused on the development of out of the box multirotor solutions that allow certain customization for the user applications.

The PixHawk[20] autopilot, (Meier et al., 2011) and (Meier et al., 2012), is an autopilot board specifically designed to allow the users to design and develop their own aerial vehicles. This autopilot counts with a customizable estimation of the state of the vehicle provided by its IMU, magnetometer, and barometer. It includes the possibility to directly connect other extra sensors, such as GPS or the px4flow. Finally, this autopilot has an embedded processor for executing its fully customizable controllers, that can be connected to a companion computer by means of a serial cable.

Microcomputers

Normally, aerial vehicles come equipped with an autopilot that has an embedded microcomputer. This microcomputer is used for executing the state estimators and controllers for basic navigation tasks. They normally have limited computational power and they do not allow to process computational expensive programs such as computer vision, or planning algorithms. Additionally, they have a limited number of inputs and outputs.

To overcome these limitations, aerial robots are equipped with more powerful microcomputers. Some examples of these small size and lightweight computers are the following:

- AscTec Atomboard[21], with an Intel Atom processor and 1 GB of RAM.
- AscTec Mastermind[22], with an Intel i5 or i7 processor and 1-4 GB of RAM.
- Intel NUC[23], with an Intel i5 or i7 processor and 1-8 GB of RAM.
- Odroid[24] U3/XU3/XU4, with and ARM processor and 2GB of RAM.

2.3. Software: simulators and middlewares

Middlewares

A robot middleware is an abstraction layer that resides between the operating system and software applications, designed to manage the heterogeneity of the hardware, improve software application quality, simplify software design, and reduce development costs. (Elkady and Sobh, 2012) provides a literature survey on robotics middlewares.

ROS (Robotics Operating System), see (Quigley et al., 2009), is one of the most widely used robotics middlewares, used for interprocess communication (using TCP/UDP protocols) and process management, that requires a name/parameter server (the ROS master).

There exist multiple middlewares with different interprocess communications approaches, such as MIRA (Einhorn et al., 2012), that claim to outperform ROS. Nevertheless, as a large part of the scientific community has adopted ROS as their robotics middleware, and the "Open Source Robotics Foundation" is actively maintaining it, the usage of other middlewares different than ROS is merely testimonial.

Simulators

Robot simulation is an essential tool in every roboticist's toolbox. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios.

There exist multiple commercially available or open source simulators with different features. (Craighead et al., 2007) does a survey on simulators of Unmanned Vehicles, analyzing the features of the 14 more well known.

[19] Online: `http://www.mikrokopter.de`
[20] Online: `https://pixhawk.org/`
[21] Online: `http://www.asctec.de/asctec-atomboard/`
[22] Online: `http://www.asctec.de/asctec-mastermind/`
[23] Online: `http://www.intel.com/content/www/us/en/nuc/overview.html`
[24] Online: `http://www.hardkernel.com/`

Some other new simulators have been recently developed, such as the MORSE simulator, (Echeverria et al., 2011), a realistic modular simulator that allows a software in the loop simulation.

Gazebo[25] is a realistic open source simulator that is compatible with ROS. (Meyer et al., 2012) provides the needed information to simulate a complete Unmanned Aerial System using Gazebo and ROS.

Both the PX4 and the APM project (see Section 2.1) provide software in the loop simulations for their software stacks.

2.4. Feature extraction from sensor information

The information coming directly from the sensors is often very complex and cannot be directly used to read the state of the robot of the state of the environment. To simplify the measurements given by the sensors, converting them into a more tractable information, feature extraction algorithms are developed.

There exist a huge amount of feature extraction algorithms, that depend on the sensor source. For example, for extracting 3D features on point clouds, the PCL (Rusu and Cousins, 2011) can be used. Another example is the Perspective-n-Point (PnP) problem, that tackles the problem of estimating the pose of a calibrated camera given a set of n 3D points in the world and their corresponding 2D projections in the image. This problem has been studied in multiple works such as: (Wu and Hu, 2006), (Leng and Sun, 2009), (Li et al., 2012), (Hesch and Roumeliotis, 2011), (Lepetit et al., 2008).

This thesis deeply analyzes two feature extractors whose state of the art is reviewed in the following paragraphs: the fiducial visual markers, and the helipad detection for shipboard landing.

Fiducial Visual Markers

Visual Markers are elements of the environment that can be detected by a computer vision algorithm. The most common type of visual markers are fiducials, that is, artificial visual features added in the environment. As they normally provide a unique id tag, they are sometimes called coded visual markers.

These types of fiducials were first developed and popularized by augmented reality applications, for example, (Zhang et al., 2002) can be consulted for a survey of visual markers for AR applications.

Nevertheless, they have been widely adopted by the robotics community. Their uses range from ground truthing to object detection and tracking, where they can be used as a simplifying assumption in lieu of more sophisticated perception.

Many works propose different kinds of fiducial visual markers, claiming for different properties. (Fiala, 2005) presented the famous ARTag. reacTIVision are introduced in (Kaltenbrunner and Bencina, 2007). The very efficient AprilTag and their second version are presented in (Olson, 2011) and (Wang and Olson, 2016) respectively. The simpler Whycon visual markers are covered on (Krajník et al., 2014) with UAS applications. (Lightbody et al., 2017), propose the Whycode, an improvement of the Whycon visual markers.

Despite the big amount of the available visual markers, the preferred in this thesis are the ArUco visual markers (Garrido-Jurado et al., 2014; Garrido-Jurado et al., 2016), since they have an open source library[26], that is currently maintained and updated, presenting a very good performance.

Helipad detection for shipboard landing

Visually-guided autonomous landing is a topic that has been extensively studied in the recent years. (Kong et al., 2014) does a complete survey on this topic.

Focusing only in visually-guided autonomous landing for vertical take-off and landing, the number of available works is limited. (Sharp et al., 2001), (Xu et al., 2006a) propose computer vision algorithms for the landing problem, based on coded visual markers that result very unrealistic. (Fan et al., 2008) proposes a computer vision algorithm based on realistic helipad marks but assuming that some of the movements of the helicopter are limited. (Yang et al., 2013) developed a computer vision algorithm that relies on inertial sensors to detect and recover the 3D pose of a realistic helipad for the landing of a multirotor.

Some authors have gone further, and they have completed the full task of the visually-guided autonomous landing on static and moving platforms. (Saripalli et al., 2003) demonstrated the visually-guided autonomous landing of a helicopter on a static surface with realistic helipad marks, while (Saripalli, 2009), landed on a 2D moving platform. (Lee et al., 2012) demonstrated the visually-guided autonomous landing of a multirotor on a 2D moving platform with coded visual markers.

Nevertheless, to the knowledge of the author of the thesis, no authors have achieved the fully autonomous visually-guided landing on a ship deck using realistic helipad marks.

[25] Online: `http://gazebosim.org/`

[26] Online: `http://www.uco.es/investiga/grupos/ava/node/26`

2.5. State estimation, mapping, and sensor fusion

2.5.1. State estimation

State estimation is the process of calculating the state of the robots and their environment based on some calculated models and the measurements given by their sensors. There exist multiple algorithms used for state estimation.

Some state estimators assume that the state of the environment is previously known, and only the state of the robot require to be estimated, this is sometimes called the localization problem. Simultaneous Localization and Mapping (SLAM) refers to the process of estimating at the same time the state of the robot and the state of its environment. Estimating only the state of the map is called mapping.

There are multiple algorithms used for state estimation, but they can be classified into three main groups:

- Kalman filters and variety
- Particle filters and variety
- Graph-based

The most widely used state estimators in robotics are the Kalman Filters (and all their variety), although other approaches are becoming more popular thanks to the growing available computational power.

Robot state estimation with Kalman filters

Extended Kalman filters (EKF), deeply described in Appendix D, are the preferred choice when working with non-linear systems.

An important aspect needed to analyzes when working in state estimation is the estimation of the state (*direct*) or the estimation of the error state (*indirect*). (Panich, 2010) justifies the usage of indirect approaches, for increasing the stability of the filter.

As extended Kalman filters are linearized versions of Kalman filters, they suffer from linearization errors. (Shojaie et al., 2007) suggest iterating the EKF to improve its accuracy and robustness against linear error propagation. (Xu et al., 2014) goes a little further proposing the adaptive iterated EKF.

A very widely use a variant of the EKF is the unscented Kalman filter (UKF)(Julier and Uhlmann, 1997). Although some works claim that the UKF outperforms the EFK, (Wan and Merwe, 2000) (Kandepu et al., 2008), this affirmation is not completely accepted by the research community.

Involving 3D attitudes in the state estimation problem is not an easy task due to the complexity on representing 3D attitudes (see Appendix A). (LaViola, 2003) recommend using EKF (better than UKF) when using quaternions for estimating attitudes, due to its simplicity. When using quaternions for attitude representation, two possible estimation approaches appear: Additive EKF, proposed in (Bar-Itzhack and Oshman, 1985); and Multiplicative EKF, proposed in (Lefferts et al., 1982) and (Markley, 2003). (Markley, 2004b) and (Markley, 2004a) analyze how to handle quaternions in EKF state estimation, comparing the Multiplicative and Additive approaches, and finally recommending the multiplicative one for being theoretically consistent. Another important issue that needs to be taken into account when estimating attitudes based on quaternion representations is how to deal with the quaternion covariance intersection, analyzed in (Crassidis et al., 2009). The attitude estimation problem using quaternions has been a recurrent problem and (Ling-juan et al., 2002) and (Barfoot et al., 2011) revisited this topic, reconfirming the advantages of the multiplicative approach.

(Sola, 2016) proposes a solution based on an indirect multiplicative EKF using quaternions for full state estimation based on IMU measurements.

Simultaneous localization and mapping with Kalman filters

Extended Kalman filters have been widely used also for simultaneous localization and mapping (EKF-SLAM), (Sola, 2013) and (Matsebe et al., 2010). Their limit appears when the size of the map grows significantly and therefore the computational performance of the EKF decreases drastically.

Some Kalman filters variants try to increase the performance of the EKF during the mapping, like the sparse extended information filters, (Thrun et al., 2004) (Eustice et al., 2005) (Walter et al., 2007).

Particle filters

Particle Filters (PF), also called Sequential Monte Carlo (SMC), (Gustafsson, 2010), are a very popular alternative to Kalman Filters. The represent non linear systems and non Gaussian noises more accurately than Kalman Filters, but their main limitation is that they become intractable when the dimension of the state grows a little. A remarkable property that they present is that they are easily parallelizable.

Graph-based

Graph-based state estimators keep a record of all the states and measurements, representing them as a graph and optimizing them as the state estimation algorithm. Graph-based SLAM, (Grisetti et al., 2010), is an example of this kind of algorithms. They were initially used offline, but currently, its online usage is a reality.

General frameworks for graph optimizations, like g2o, (Kümmerle et al., 2011), are a powerful tool that eases the usage of these algorithms.

2.5.2. Vision-based state estimation

Using vision-based algorithms for state estimation has become a trend in the last years because of the low-cost, and the light weight of cameras, and therefore it worths to mention the last advantages on these algorithms.

Purely vision-based algorithms can be classified depending on the information used as input in: *dense*, using all pixels of the images and therefore requiring intensive computations *semi-dense*, using image patches *keypoints* and *visual markers*, using salient objects in the scene. The three first approaches require a textured environment with proper visibility conditions, suffering in very common indoors environments like walls or shining well-polished floors. Additionally, they do not deal properly with non-static scenes, specially when the moving objects are predominant in the image, critical for physical interaction tasks. On the other hand, visual markers based approaches require the environment to have salient objects. These approaches are able to handle untextured environment (e.g. a big white wall with a small window) or non-static scenes (e.g. an aerial robot pushing a box in a big storehouse). The movement of these objects might be modeled and included in the estimator, allowing to work in dynamic environments. The critical stage on these algorithms is the accurate detection of these salient objects. The use of natural visual markers (such as windows, doors, boxes, etc.) is not yet possible due to the lack of accuracy on the detection or the data association problem. Nevertheless, the last advantages on deep learning applied to object detection, are showing promising results (see (LeCun et al., 2015)).

Another possible classification of vision-based algorithms is attending to the number of cameras employed, in *monocular* or *stereo*. Monocular algorithms use a single camera and they are unable to recover the scale of the movement (unless there is any object in the scene with known dimensions). Stereo algorithms use two or more synchronized cameras, that is, two or more simultaneous points of view of the scene. Stereo algorithms are able to recover the full 3D pose, including the scale.

The last classification can be done attending to the main objective of the algorithm in:

- Structure from Motion (SfM) algorithms are in charged of reconstructing, normally offline, the poses of some discrete frames, creating an absolute map of the environment.
- SLAM algorithms, that try to estimate the absolute drift-free pose of the camera, together with an absolute map on an image stream.
- Visual Odometry (VO) algorithms, estimate the incremental pose of the camera between frames of the image stream, suffering from drift when estimating the absolute pose of the camera.

Parallel Tracking and Mapping (PTAM), presented in (Klein and Murray, 2009), is the most famous and successful keypoint based monocular SLAM algorithms. As opposite, Dense Tracking and Mapping (DTAM), (Newcombe et al., 2011), is the most famous dense monocular SLAM algorithms. (Forster et al., 2014) presented the Semi-dense Visual Odometry (SVO), a very successful semi-dense Visual Odometry algorithms.

All the previous three cited algorithms, have a common structure in their state estimation algorithm, firstly introduced in the PTAM. They have two parallel threads, the first one is performing a fast tracking and pose reconstruction from a 3D map; while the second thread is online building this map performing a graph-based optimization (as an online structure from motion) of some representative keyframes. The importance of the mapping thread is analyzed in (Polok et al., 2015).

(Lim and Lee, 2009) developed a pure visual SLAM using fiducial visual markers for the autonomous navigation of a robot.

Finally, recovering the camera pose from a known (vision-based) map, and improving this map with different past experiences, are two new fields of vision-based state estimation:

- Visual place recognition consist on recovering the camera pose in a known map. Examples of this are: the FAB-MAP, presented in (Cummins and Newman, 2008), (Paul and Newman, 2010), (Cummins and Newman, 2011); the SeqSLAM, presented in (Milford and Wyeth, 2012); and (Stumm et al., 2015).
- Visual experience-based localization is an extension of visual place recognition that tries to incorporate into the visual map the different experiences (e.g. daylight, night light, rain, fog, ...) when navigating in an environment to improve its performance. Examples of this are: the experience-based navigation presented

in (Churchill and Newman, 2012a), (Churchill and Newman, 2012b), (Churchill and Newman, 2013), (Linegar et al., 2015); the visual teach and repeat in (Dequaire et al., 2016); and the metric localization from (Linegar et al., 2016).

2.5.3. Mapping

As stated before, the mapping is the part of state estimation that targets the creation of a map of the environment. All SLAM algorithms create a map of the environment, nevertheless, this map is highly dependent on the nature of the sensors used, and on the algorithm. An efficient and usable map representation is required to be used by other components such as planners.

The generation of maps based on a 3D mesh is a possible solution. (Weiss et al., 2011) generate a 3D mesh map from a keypoint map given by a monocular SLAM algorithm to use onboard an aerial robot.

Other solutions, like grid maps are widely used, specially in 2D map representations. 3D grid maps are extremely inefficient and more adequate solutions are used. Octrees, and its Octomap implementation (Wurm et al., 2010), as well as its multiple variations such as (Hornung et al., 2013) and (Krajnık et al., 2014) are a trend.

More compact representations of the map using geometric primitives are starting to be explored.

2.5.4. Multiple sensor fusion

Multiple Sensor Fusion (MSF) for state estimation is a common feature in all works related to navigation of robotics, and in particular of Unmanned Aerial Systems (UAS).

Using the information coming from different sensors for the state estimation provides reliability in the estimation. Additionally, sometimes is the only way to get the complete state at the required rate.

There are two main paradigms to fuse the measurements coming from different sensors: (1) *tightly-coupled*, where the raw measurements are directly incorporated to the sensor fusion algorithm, and (2) *loosely-coupled*, where the raw measurements are preprocessed and after that incorporated to the sensor fusion algorithm. Despite showing poorer results, loosely-coupled approaches are preferred due to their versatility.

(Bancroft, 2009) show a loosely-coupled algorithm to fuse multiple IMUs and GPS applied to ground vehicles using an EKF, while (Zhang et al., 2005) fuse the same information for a ground robot using an UKF. (Caron et al., 2006) fuses in simulation measurements coming from IMUs and GPS with a Kalman Filter approach, but tackle the sensor failure detection using a fuzzy approach. (Kingston and Beard, 2004) shows the IMU and GPS fusion for a fixed wing UAV.

The Visual Inertial navigation, what requires to fuse the measurements coming from an IMU and images coming from a camera is a trend in robotics and specially in aerial robotics. (Loianno et al., 2015) propose a tightly-coupled Visual Inertial Odometry (VIO) algorithm used on a smartphone powered UAS that fuses visual keypoints with IMU measurements. (Jones and Soatto, 2011) developed a visual inertial state estimator for a ground robot. (Nützi et al., 2011) and (Weiss and Siegwart, 2011) fused IMU measurements and the output of a PTAM monocular SLAM algorithm using an EKF. (Weiss et al., 2012b) do a similar work, testing on an aerial robot (an AscTec Firefly) (Kelly and Sukhatme, 2011) use an UKF for fusing IMU and camera measurements. (Liu et al., 2016) developed a loosely-coupled algorithm to fuse monocular SLAM and IMU measurements decoupling angular and linear states. (Li et al., 2013) tackle the Visual Inertial problem with rolling shutter cameras.

Combining visual markers with IMU measurements allows to develop a robust state estimator that do not depend on the environment texture.

(Kneip et al., 2011) estimate the position and velocity of the robot relative to a single feature by fusing the information coming from an IMU and a visual feature detector. (Zhang et al., 2009) fuses the measurements coming from an IMU and a visual marker detector for demonstrating the hovering of an UAS. Some works, like (Neunert et al., 2016) fuse IMU measurements and visual markers in a state estimation algorithm that maps the visual markers in the image plane, what makes the algorithm very dependent on the used visual markers.

(Zhang and Singh, 2015) propose a tightly-coupled Visual Lidar Odometry.

Some works go further and fuse the information coming from different sensors. (Sola, 2016) and (Trawny and Roumeliotis, 2005) propose a EKF quaternion-based sensor fusion of an IMU and some other extra sensors. (Achtelik et al., 2011), (Weiss et al., 2012a) and (Lynen et al., 2013) are consecutive works from the same authors where they explore the sensor fusion in a loosely-coupling fashion of IMU and other sensors like a monocular SLAM algorithm, barometers, and generic pose and position sensors for its use on UAS, all based on EKF. (Shen et al., 2014) show a loosely-coupling modular MSF algorithm for UAS based on an UKF. Almost all of the previously cited loosely-coupled algorithms use the IMU sensor in the prediction stage, what limits to one the

number of IMUs that can be used. (Burri et al., 2015) propose for sensor fusion to use a dynamic model of the UAS in the prediction model instead, which increases the complexity of the model and requires more parameters to be identified.

An important issue that need to be taken into account in MSF is the sensor calibration. Camera calibration is a widely studied topic. (Tedaldi et al., 2014) tackle the IMU calibration problem without extra equipment, while (Kim and Golnaraghi, 2004) calibrates an IMU using a motion capture system. The inter sensor calibration is also needed, and for example, (Hwangbo et al., 2013) analyze the simultaneous multi-IMU and camera calibration.

Another important aspect of MSF is the sensor synchronization. Mainly tightly-coupled approaches require that the measurements are perfectly synchronized. Nevertheless, assuming to have all the measurements perfectly synchronized is not real in most of the robotic systems, and the condition is relaxed to have the measurements perfectly timestamped. Having a perfect timestamp is also difficult, and some approaches, like (Olson, 2010), propose some algorithms to accurately timestamp the measurements.

Finally, to incorporate time-delayed measurements, a buffer that keeps track of the measurements and states is the most common option, seen in some of the previously cited works.

2.6. Control

The controllers have the goal to transform the references such as trajectories, positions, or velocities into actuator commands, ensuring that the commanded reference is followed by the aerial platform. Multiple research lines appear within the control of Unmanned Aerial System, including among others: navigation (covering attitude, velocity, position and trajectory tracking or path following), aggressive maneuvers, visual servoing, or physical interaction.

Navigation and aggressive maneuvers

Several works have shown that the navigation control loops must be designed taking into account the non-linear dynamics of the multirotor so that the control actions are approximately decoupled. If the control laws are well designed the multirotor can perform smooth trajectories in position coordinates while orientating its yaw heading in any required direction.

Several labs have used a sub-millimeter accurate motion capture systems to separate the control and the state estimation problems with testbeds such as (Michael et al., 2010; Lupashin et al., 2014). Relying on these motion capture systems has simplified the research problem, which has shown that state estimation is the key to enabling many autonomous applications.

It is worth to highlight the difference between trajectory tracking, where the vehicle is required to track a reference parameterized in time; and path following, where the objective is to make the vehicle follow a desired geometric path, without an explicit timing law assigned to it, as discussed in (Sarras and Siguerdidjane, 2014).

Researchers at the GRASP lab at the University of Pennsylvania, and at the Flying Machine Arena in the ETHZ, have been able to execute precise trajectory following (Mellinger et al., 2014; Michael et al., 2010), and to perform aggressive maneuvers (Michael et al., 2010; Lupashin et al., 2010). Many other researchers such as (Lee et al., 2011) have demonstrated as well the precise aggressive trajectory tracking.

Back in the early years of multirotors, several research groups have shown successful autonomous multirotor navigation capabilities without relying on any motion capture system but using mainly GPS positioning data. The STARMAC project, from Stanford University, has shown several experimental tests on outdoors ranging from trajectory tracking tasks (Hoffmann et al., 2008) to performing flips and other aggressive maneuvers (Huang et al., 2009; Gillula et al., 2010).

As presented in Section 2.9, current research works have demonstrated the precise trajectory tracking for autonomous navigation in both indoor and outdoor environments using different combinations of sensors, being its main bottleneck, the state estimation.

Currently, most of the commercial aerial platforms and autopilots provide an out of the box controller for basic attitude, velocity, position, and path following.

Visual Servoing

Research on visual servoing has shown that the performance of the robot depends on the set of used image features, which should be decoupled (Tahri et al., 2010) or based on computing image moments on a group of points on the target (Tahri and Chaumette, 2005). Recent research has included non-overlapping multi-camera robotic systems (Comport et al., 2011). More specifically, (Hamel and Mahony, 2002) discusses “eye-in-hand” systems where the camera is fixed to a rigid body with actuated dynamics.

Some works have demonstrated the visual servoing of an Unmanned Aerial System, but following simple targets such as blobs of different sizes (Bourquardez et al., 2009), circular markers (Eberli et al., 2011), or balloons (Mondragón et al., 2011; Zhang and Ostrowski, 1999). Other conceptual works (Mahony et al., 2005) demonstrated that incorporating information coming from other sensors like the IMU, the visual sensor problem is simplified.

There exist in the literature multiple visual trackers that can be used for the visual servoing task, but one of the best trackers is the TLD tracker (Kalal, 2011; Kalal et al., 2012a). This algorithm tracks the position and scale of the planar object in the image, recovering therefore only 3 degrees of freedom between two consecutive images. Other tracking algorithms, like (Martínez et al., 2013) are designed to recover all the degrees of freedom of the transformation between the two consecutive images. Recovering the 3D transformation between the two consecutive camera movements is not possible in monocular systems because the depth of the 3D position cannot be recovered. The only possible way to recover this complete transformation is having a previous knowledge (the size) of the target.

2.7. Planning

Mission Planning

Planning dynamic missions for robotics normally consist of an optimization process that typically searches in a graph. One of the biggest problems in mission planning is the definition of the mission by the human operator.

There are a number of available software applications that allow operators to specify a mission for aerial vehicles using a ground control station. Examples of such applications are: MP (Mission Planner)[27], APM Planner 2[28], MAVProxy[29], QgroundControl[30], and PC Ground Station (DJI)[31]. However, the use of waypoint lists to specify a mission presents important limitations (Santamaria et al., 2008; Schwartz et al., 2014). One of the most significant limitations is that they are rigid descriptions that are not able to adapt to mission circumstances. The specification is normally based on a fixed list of waypoints that cannot change dynamically (e.g., in the presence of dynamic obstacles). The specification follows a main sequential workflow and it is difficult to specify alternative flows (e.g., iterations, branches, conditional waypoints, alternative flows).

In order to cope with these limitations, mission specification languages can follow more powerful planning approaches. For example, there are classical planning approaches in artificial intelligence (e.g., with STRIPS-like operators). However, to be used successfully in robotics, they need to be adapted or combined with other solutions to react efficiently to the environment changes, monitor mission execution and integrate with reactive behaviors (Rothenstein, 2002). Some practical planning solutions in robotics are task-based specifications, Petri nets, or behavior-based specifications besides others (rule-based reactive planners or finite state machines).

Path Planning

Path planning for robotic applications, in general, is a very well studied topic since the beginning of the robotics.

Algorithms that tries to search the collision-free path over a graph (discrete search algorithms), initially a grid map, have been used a lot. Examples of these algorithms are the famous single-query $A^\star$ algorithm, (Hart et al., 1968) (Dechter and Pearl, 1985), and multi-query variants like the $D^\star$ algorithm, (Stentz, 1994).

As the discrete search algorithms have high computational requirements, other works proposed to use a potential field map (also called Artificial Field Maps, AFM) approach, where the final point has an attractive potential and the obstacles have a repulsive potential, (Hwang and Ahuja, 1992). The main disadvantage of this method is that the search easily falls into local minima, being unable to find a solution. Many works are based on AFM, trying to avoid their limitations. (Park et al., 2001) proposed a simulated annealing in AFM; (Xu et al., 2006b) combined AFM with genetic algorithms (GA); (bo Chen et al., 2016) merged AFM with optimal control theory; and (Yang and Sukkarieh, 2012) joined the AFM with model predictive control for fixed-wing UAV path planning

Other approaches used visibility graphs to reduce the computational cost. (Huang and Chung, 2004) defined a variant called Dynamic Visibility Graphs (DVG). (Arantes et al., 2016) proposed the combination of a visibility graph with a multi-population genetic algorithm.

Rapidly-exploring random trees (RRT), presented in (Lavalle, 1998), proposed a novel single-query planning method that tries to cover the search space using random samples that are connected to the tree.

[27] Online: `http://planner.ardupilot.org/planner/index.html`
[28] Online: `http://planner.ardupilot.org/planner2/index.html`
[29] Online: `http://dronecode.github.io/MAVProxy/html/index.html`
[30] Online: `http://qgroundcontrol.io`
[31] Online: `http://www.dji.com/es/product/pc-ground-station`

Probabilistic Roadmaps (PRM), presented in (Kavraki et al., 1996), designed a multi-query method that creates a collision-free graph of random nodes on the search space. Multiple improvements over this method have been done (Geraerts and Overmars, 2004): (Bohlin and Kavraki, 2000) presented the Lazy-PRM; and (Song et al., 2001) presented the C-PRM.

Other techniques use optimization and search algorithms like the Particle Swarm Optimization (PSO) algorithm to calculate the optimum path planning. (Zhang et al., 2013) used a multi-objective PSO, whereas (Roberge et al., 2013) fused a genetic algorithm with a PSO for UAV path planning. (Hidalgo-Paniagua et al., 2016) used a multi-objective evolutionary algorithm on a grid map.

(Yu and Zhang, 2015) does a survey on path planning for UAVs. Some recent algorithms have focused on optimizing the path planning task for the specific restrictions of fixed-wing UAVs: (Al-Sabban et al., 2013) proposed a path planning optimization based on wind-energy for a fixed-wing UAV; (Chen et al., 2013) presented an algorithm called T+LVFG for fixed-wing UAV trajectory tracking with obstacle avoidance incorporating wind information; (Yao et al., 2016) adapted the common grid model for the particularities of fixed-wing UAVs.

2.8. Human-robot interaction

In Human-Robot Interaction (HRI) it is crucial for humans to be able to interact and command the robots in natural and efficient ways. To allow effective operation and control of the machine from the human end, interactions between the humans and machines must occur. This interaction takes place in the user interface (UI).

User Interface evolution: CLI, GUI, NUI

The first UIs were simple designs, not intuitive for the users. With newly developed hardware becoming ever so common, new technologies were being developed which meant researchers had to create new forms of interacting with machines. Hence a growing interest in designing new types of interfaces has developed over the decades. As stated in (Wigdor and Wixon, 2011), these types can initially be divided into command line interface (CLI) succeeded by the GUI (Fig. 2.1). Nevertheless, the fact that in CLIs operators have to interact with the systems by typing preprogrammed keywords into a command prompt can lead to novice users feeling completely overwrought when experiencing these interfaces for the first time (Nielsen, 1993).

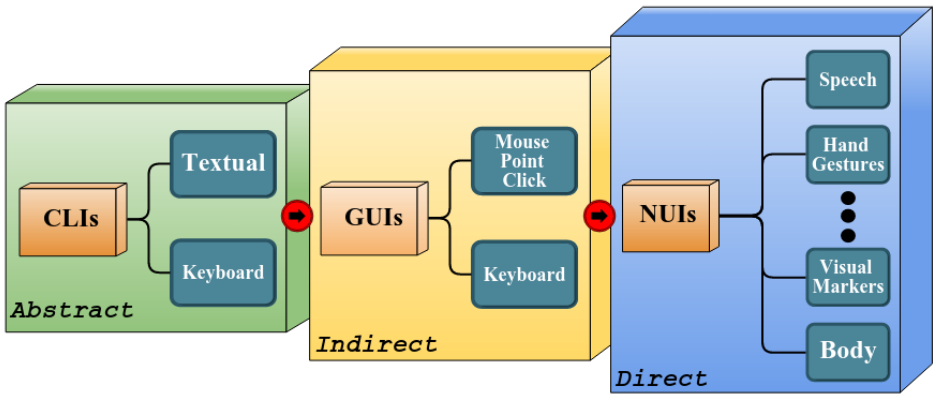


Figure 2.1: The User Interface evolution. Command line interfaces began the UI revolution followed by a more indirect GUI. The most recent user interface is the NUI.

The GUI, that is still used today, produces an indirect but expected mode of interaction by using what is commonly referred to as WIMP (Windows, Icons, Menus, Pointer) (Wigdor and Wixon, 2011) (Turk and Robertson, 2000), a set of user interface elements that serve as user inputs and machine outputs. In comparison to CLIs, these interfaces represent a lower obstacle for users since recognizing and choosing commands is easier than remembering and typing (Wigdor and Wixon, 2011). The properties of WIMP GUIs provide users a clearer idea of what actions and processes are available in the system as well as what their effects might be, this allows users to have a sense of achievement about their interactions with computer programs (Turk and Robertson, 2000). Despite the fact that GUIs have been very prosperous and controlled both human-machine interface (HMI) research and the marketplace for most of three decades (Medioni and Kang, 2004), these interfaces are not the most efficient option, given that there is still a barrier between the communication from human to machine.

Consequently, there is an ever growing demand to create more immerse UIs that take full advantage of modern technologies allowing users to be able to feel fully integrated into the devices they use (Salih, 2015). One step towards achieving this is the Natural User Interface (NUI).

(Preece et al., 2001) describes NUIs as types of interfaces that allow users to engage with machines in a similar way they would interact with the real world through using body movements, hands or even voice. Unlike GUIs where users had to use keys, buttons or a computer mouse, now language, touch, or body movements are used to control a device. Thus, the user can interact directly with the elements on the screen without an additional device, such as the mouse. Thereby a seamless interaction between humans and computers has been created by which you can handle virtual or real objects in a realistic manner. Most NUIs rely on additional equipment for suitable and efficient interaction, however, these devices tend to be so unnoticeable that they appear invisible to the user. Natural user interfaces have been researched since the 1980s, for instance in (Bolt, 1980), the author used gestures and voice commands for control of a graphical user interface (GUI).

Hand gestures are frequently considered the most expressive and in so, the most often used in the literature for new NUIs. These gestures involve (1) a posture: normally expressed by a lack of hand movement with predetermined finger configurations and (2) a gesture: where hand motion is dynamic (Kaushik and Jain, 2014).

The interpreting of gestures by the NUI requires that the configurations (static and/or dynamic) of the human hand and even arm, be measurable by some device. Initial attempts at hand/arm based NUIs were known as *glove-based devices* (Baudel and Beaudouin-Lafon, 1993) (Fels and Hinton, 1993). These works depended on unwieldy sensors that directly measured the spatial position and joint angles. Users found these devices to impede the interaction between the user and the computer controlled environment (Kaushik and Jain, 2014). The need for more natural interaction between the human and the machine has spawned research into the use of other devices.

Human-computer interaction by means of the speech has become more and more common due to the growing number of speech recognition software and toolkits that are readily available. Early examples of speech recognition interfaces were mainly used for speech-to-text applications. Products such as Dragon Naturally Speaking allow the user to dictate and have speech transcribed as written text, have a document synthesized as an audio stream, or issue commands that are recognized as such by the program. These type of applications bridge the gap between the spoken word and its written form and offer hands/eyes-free interfaces that are intuitive and appealing to the user.

Natural Human Robot Interaction

The two main mediums to implement reliable human-robot interaction (HRI) are voice and gesture based NUIs. Since the latter represents direct expression of mental concepts, it is the most preferred in literature (Pavlovic et al., 1997).

Visually recognized natural gestures (Hu et al., 2003) and face pose recognition (Ju and Kang, 2007) have already been used for robotic teleoperation, allowing non-expert users, to interact and operate the robots. Some other works explored the interaction with ground robots by means of a laser gesture interface (Ishii et al., 2009). (Alvarez-Santos et al., 2014) successfully tested in real world scenarios a tour-guide robot that recognized and provided feedback for user hand gesture commands or augmented reality virtual button selection.

Natural Human Aerial Robot Interaction

Some works analyze the ideas of having aerial robots interacting with people, like (Cauchard et al., 2015), that explore how the Natural Human-Aerial-Robot Interaction should be; and (Graether and Mueller, 2012) and (Mueller and Muirhead, 2015), that explore the idea of using a runners body to exercise with an aerial robot companion.

The wide range of hand/arm and body gestures that can be recognized, as well as the verbal vocabulary that can be communicated to the aerial robot, can offer unique opportunities for developing new and captivating types of HDIs. There are multiple works found in the literature, aimed to explore the natural interaction between an aerial robot navigating in GPS-denied environments by using the user's body position, hand gestures, visual markers and/or speech.

Depth cameras like the Microsoft Kinect^{TM} sensor is one of the most widely used NUIs, In (Mashood et al., 2015) and (Sanna et al., 2013), an aerial robot is teleoperated by sending discrete control commands given by static arm gesture recognition techniques. in (del Valle et al., 2014), the authors develop a gesture recognition system based on depth imagery and create a depth-based hand gesture database for control of aerial robots. However, such devices present problems when it comes to accurate recognition of depth-based hand gestures which include, reduced resolution, high noise, and missing data. These deficiencies make it infeasible to extract reliable data of accurate hand/finger poses (del Valle et al., 2014).

On the other hand, by taking advantage of the aerial robot onboard cameras, body and face position estimation and tracking, or gesture recognition can give the user the ability to interact with the aerial robot on a personal

level, like in (Nagi et al., 2014) and (Monajjemi et al., 2013).

Few examples of speech interaction with aerial robots can be found in the literature. In (Quigley et al., 2004) a voice controller was developed that could recognize commands sent to a fixed wing semi-autonomous UAS using a PDA. Real-flight tests with their interface showed that ambient wind noise and conversation can lower the reliability of the voice recognition system. In (Jones et al., 2010) conducted an exploratory study of gesture and speech interfaces for interaction with robots in a simulated environment, which concluded that the test subjects generally preferred using lower-level commands such as *left* or *right* to command the aerial robot.

2.9. Aerial robotics navigation

This section lists a sample of some works related to the autonomous navigation of aerial robots. The works presented here include complete systems for specific applications or environments, describing several integrated components, such as state estimators, controllers, planners, etc.

Single aerial robot navigation

(Kendoul, 2012) presented a complete survey in rotary wing unmanned aerial systems.

(Zingg et al., 2010; Lippiello et al., 2011) are focused on navigation in corridors and obstacle collision avoidance strategies using multiple optical flow sensors.

In outdoors navigation, optical flow sensors were used in a fixed-wing mUAV in (Zufferey et al., 2010) to avoid the collision with trees and other obstacles in a GPS waypoint trajectory.

(Blösch et al., 2010) and (Scaramuzza et al., 2014) explore the autonomous visually-guided navigation of multirotor UAVs, using a monocular visual SLAM fused with other sensors and proposing a trajectory controller.

Some groups have increased the situational awareness of the mUAVs by means of cooperative robots. For instance, in (Rudol et al., 2008), a ground vehicle can estimate the position and attitude of the UAV.

Aerial Multi-robot systems

Researchers at the GRASP lab at the University of Pennsylvania, and at the Flying Machine Arena in the ETHZ, have been able to perform formation control tasks that require synchronization among the flying vehicles (Kushleyev et al., 2013; Sch ollig et al., 2010).

(Doitsidis et al., 2012) tackled the surveillance by a system formed by multiple UAS. These UAS fuse the information coming from an IMU and a monocular visual SLAM algorithm, being able as well to reconstruct a map of the environment. The path that every robot has to follow is optimized for the task of terrain surveillance.

2.10. Discussion

Along this chapter, the state of the art related to the aerial robotics has been analyzed, extracting the following conclusions:

Aerial robotics architectures and frameworks

Even though this line of research has produced important advances, the referred work shows that there are important remaining challenges related to (1) level of autonomy, more complex hybrid architectures able to provide more degree of autonomy; and (2) versatility, more versatile integrated solutions able to be used for different applications and hardware setups.

Hardware components and software packages for aerial robotics

The growing of the market of unmanned aerial systems in the last years has significantly pushed the state of the art of the hardware components useful for aerial robotics applications, making available a large amount of small and light sensors, together with powerful microcomputers. In addition, multiple robust and very stable aerial platforms that can be equipped with cheap and versatile autopilots are ready to be used. These aerial platforms still present some technological limitations, as their limited payload or flight endurance. Nevertheless, they are suitable for a multitude of aerial robotics applications.

Furthermore, the boost experienced by general robotics has propitiated the emergence of multiple software packages, very useful in aerial robotics, such as middlewares and simulators.

Algorithms to increase the autonomy level of aerial robots

Multiple research works are focused on the development of specific algorithms related to perception, control, planning and task execution, intelligence and cognition, and social interaction. They have shown very important advances with highly spectacular results. Nevertheless, these works normally tackle the problems they try to solve isolatedly, without considering the whole variables that come into play in complete robotics applications. In addition, some of them are only valid for some particular limited conditions, environments, or hardware setup. This shows the need of developing versatile aerial robotics specific algorithms whose performance does not decrease when integrated into a complete system.

Solutions for aerial robotics navigation

There exist some works that show fully-autonomous applications related to aerial robotics navigation of single or multiple systems. Nevertheless, these works lack versatility and they are application specific.

Part II

ARCHITECTURE AND FRAMEWORK FOR AERIAL ROBOTICS

Chapter 3

System Architecture for Aerial Robotics

This chapter presents a system architecture for aerial robotics (depicted in Fig. 3.1) as a functional abstract description of its principal layers and systems, together with their relationships.

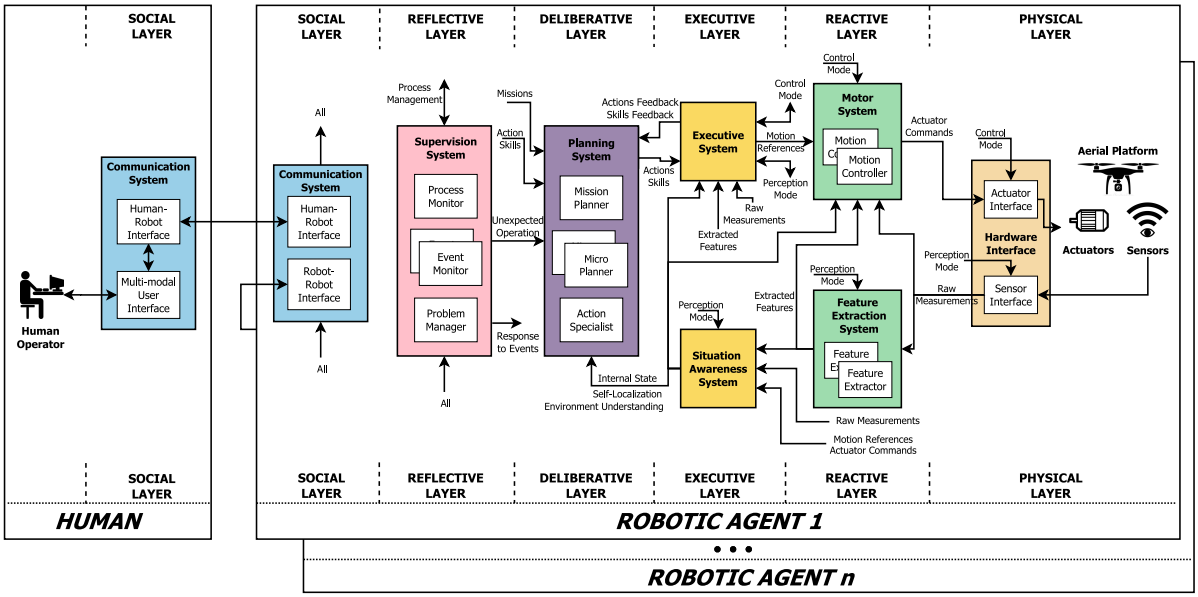


Figure 3.1: Proposed system architecture for aerial robotics.

The proposed system architecture has a uniform hierarchical organization, according to homogeneous blocks at two abstract levels: layer and system. This homogeneity and hierarchical organization facilitates understanding the complexity of the model.

The description of the system architecture is formulated defining the role of its layers and systems and their connections, but independently from specific implementations. Thus, the model is not committed to any specific hardware and software, which facilitates reusability for different platforms. The proposed system architecture is,

therefore, an abstract template that defines functionalities of its systems and their relationships at a general level.

The practical utility of using the proposed system architecture are threefold: First, it allows the designers to quickly design a fully autonomous robotic aerial system for any application, simplifying the systems engineering stage. Second, the architecture allows the user to consider all the modules needed for the correct operation of the complete system, avoiding design errors due to some functionalities are not taken into account or not properly described. Third, it allows the developers to easily focus on the working of one or several components without ambiguity because the component's utility is properly defined.

The remainder of the chapter is organized as follows: The main features of the system architecture are enumerated in Section 3.1. In Section 3.2, the standardized relationships between the systems of the proposed architecture are listed and described. Section 3.3 discusses other tested multi-robot approaches that modify the proposed architecture. Finally, Section 3.4 shows the historical evolution of the proposed system architecture during its more than four years of existence.

3.1. Description and main features

3.1.1. Multi-layered model

The proposed system architecture follows the hybrid reactive/deliberative paradigm, i.e., an architecture that integrates both a deliberative and reactive approaches (Arkin et al., 1987) and (Murphy, 2000). As depicted in Fig. 3.8, the presented design includes the following five layers (plus the physical layer): reactive, executive, deliberative, reflective and social.

The first three layers correspond to the popular hybrid design known as the three layered architecture (Gat, 1998) and (Russell and Norvig, 2003): (1) *reactive layer* with low-level control with sensor-action loops; (2) *executive layer* (or sequencing layer) that accepts symbolic actions from the deliberative layer and generates detailed behavior sequences for the reactive layer; this layer also integrates the sensor information into an internal state representation with self-awareness; and (3) the *deliberative layer* generates global solutions to complex tasks using planning (e.g., planning optimal trajectories). The reactive layer functions in the present while the deliberative layer uses information from the past and projection to the future.

To increase the degree of autonomy of robots, the proposed system architecture includes a *reflective layer* based on cognitive architectures (Sloman, 1999), (Davis, 2002), (Brachman, 2002), and (Singh and Minsky, 2005) to simulate certain self-awareness able to supervise the other layers. The reflective layer helps to see if the robot is actually making progress to its goal and to react in the presence of problems (unexpected events, faults, etc.) with recovery actions.

The proposed system architecture includes also a *social layer* with communication abilities, as it is proposed in multiagent systems and other architectures with social coordination (e.g., (Duffy et al., 2005)). In this level is important to establish an adequate communication with human operators and other robots.

3.1.2. Multi-system model

The system architecture is divided into seven systems (deeply described in Chapter 4) plus the hardware interfaces: Feature Extraction System, Motor System, Situation Awareness System, Executive System, Planning System, Supervision System, and Communication System.

The proposed system architecture specifies only the main functionalities of these systems, together with a standardized generic description of their inputs and outputs. It avoids, therefore, to describe algorithmic or implementation details, leaving these details to the users of the system architecture. As the relationships between the systems are also abstractly described thanks to its ontology (see Section 3.2), the final users are responsible for concreting and implementing them, ensuring that all the involved systems are consistently defined. For example, the Feature Extraction System accepts raw measurements as inputs. For a particular feature extractor implemented, i.e. a computer vision algorithm for the detection of visual markers (as the one presented in Section 7.1), the raw measurements will be concreted, i.e. the images acquired by an RGB camera.

This abstract definition of the system architecture has as well no restrictions on hardware design, allowing the final users to design their hardware applications as it best suits them (for example, using a concrete sensor set-up; or running all the software in a single or in a distributed set of computers). It is important to remark that the system architecture neither imposes restrictions about communications (for example, Wi-Fi might be used).

The proposed system architecture is also consistent with the usual system description related to guidance, navigation, and control of unmanned systems, (Kendoul, 2012). In particular, the Navigation System (NS), also called perception, corresponds to the sensors together with the proposed Feature Extraction System and Situation

Awareness System; the Guidance System (GS) corresponds to presented the Executive System, Planning System and Supervision System and; finally, the Flight Control System (FCS) corresponds to the actuators together with the proposed Motor System.

3.1.3. Autonomy and self-adaptation in complex environments and operations

In general, fully autonomous robots are able to accomplish their assigned mission without human intervention while adapting to operational and environmental changing conditions (Huang, 2008a). Different degrees of autonomy can be identified (Clough, 2002), (Kendoul, 2012) that require to simulate cognitive tasks. This includes dimensions such as the following (Huang, 2008a):

- Human independence. The robot can be operated with simple commands from general human operators and does not require highly specialized operators and technical jargon.
- Dynamic and complex environments. The robot can operate in dynamic and complex environments with unexpected situations where it is required abilities such as: self-adaptation, threat avoidance, self-diagnosis, fault tolerance, etc.
- Complex missions. The robot can perform complex missions (such as search and rescue missions) where situation awareness and complex planning is required.

In particular, the proposed system architecture provides the following systems related to these levels of autonomy:

- The *Situation Awareness System* uses the sensor information to acquire environmental and self-awareness, creating, therefore, a complex internal representation that is richer and more useful than the raw sensor information. This system helps to increase autonomy in terms of complex environments.
- The *Executive System* accepts directives from the deliberative layer and sequences them to be performed by the reactive layer. The Executive System translates requested actions and skills expressed as symbolic descriptions (e.g., take-off, move to a point, etc.) into specific orders for the motion controllers and the activation of certain components. The Executive System helps to increase autonomy in terms of human independence and complex missions.
- The *Planning System* automatically generates the goals in order to accomplish a particular mission. The Planning System helps to increase autonomy in terms of human independence, complex missions and self-adaptation in dynamic environments.
- The *Supervision System* is a key functional system that ensures a correct behavior of the robot. The Supervision System helps to provide self-adaptation to environment and operation changes and fault-tolerance. This typically consists of three steps: event detection, notification, and actuation/recovery.
- The *Communication System* allows the inter-agent communication (including robotic and human agents), enabling the creation of a common network for coordination and cooperation. This system increases the autonomy in terms of complex missions and complex environments.

3.1.4. Multi-robot

The proposed system architecture allows the autonomous operation of a heterogeneous multi-robot system, that is, multiple different robots operating within the same framework.

Section 3.3 deeply analyzes the tested multi-robot approaches. The proposed system architecture is based on a distributed approach where there are no extra agents with a dedicated central multi-robot control functionality.

This bio-inspired approach tries to mimic the behavior of the animals where no external intelligence is organizing them, and only the agents participate in the fleet organization, obtaining, therefore, an emergent multi-agent behavior.

The fact of not having any dedicated central multi-robot control agent in the presented system architecture does not limit to have robotic agents with different roles, like leaders and followers. Thus, this distributed approach enables different multi-robot behaviors like swarming, where all the robotic agents are identical without any implicit multi-robot control components, appearing the multi-robot behavior in an emergent way; or leader-follower (master-slave) approaches where a set of robots have, besides their own functionalities, some multi-robot controllers, and therefore the multi-robot behavior is implicitly defined.

The distributed approach eases the system architecture whose only requirement to allow multi-robot systems is to repeat the individual robotic agent architecture as many times as robotic agents has the system. Its main advantages when compared with a centralized system architecture are the following: (1) it better scales up; (2) communication data is minimized what normally supposes a bottleneck; (3) each agent of the system has a certain level of autonomy being independent of the multi-robot central controller; (4) better fault tolerance, since the central controller would be a critical essential element.

It is important to remember that the proposed system architecture only defines the functionalities of its systems without digging into implementation or algorithmic details and therefore imposes no restriction on the specific features of every robotic agent and therefore is compatible with the use of heterogeneous multi-robot agents.

3.2. Ontology

In order to facilitate the semantic interoperability of the different components, an ontology for aerial robotics has been defined specifically for the proposed system architecture following common terminology found in the research literature about robotics and aerial systems. The ontology defines the formal and explicit specification of shared concepts. The concepts are classified according to the input/output categories that Fig. 3.1 shows. The current formalization of this ontology is based on common data representations. A complete formal specification of this ontology using an appropriate language (e.g., OWL) is a pending task to be done in the future.

- *Raw measurements*: Values corresponding to direct measurements recorded by sensors. The proposed system architecture uses sensor-independent parameters whose values are obtained with the corresponding hardware interfaces (e.g. all the cameras, despite being from different manufacturers, use the same measurement data type for representing the acquired images, allowing, therefore, the interoperability between them).
- *Extracted features*: Single features extracted from measurements of physical quantities. In general, the extracted features can include a partial interpretation of characteristics of the environment such as lines, intersections, visual markers, approximate pose, etc.
- *Self-localization*: Robot localization in the environment together with its kinematic values (e.g., velocities) as they are believed by the robot. For example: pose, velocities, accelerations, forces, torques, etc. This data encodes the situation awareness of the robot.
- *Environment Understanding*: Characteristics of the environment and its elements as they are believed by the robot. For example: walls, pole obstacles, other robots, distance to obstacles, etc. This data encodes the situation awareness of the environment.
- *Internal state*: This encodes the self-awareness of the robot, including the self characteristic of the robot. For example: pose of the sensors in the robot, battery level, etc.
- *Perception mode*: Representation of the perception set-up, including the enabled sensors, the feature extractors, and the situation awareness components, together with their relationship.
- *Actuator commands*: Motion values that are accepted by the actuators. Examples are: voltage, or Pulse Width Modulation (PWM).
- *Motion references*: Motion values to be considered as goals by controllers or planners. Examples of references are: position, velocity, or yaw.
- *Motion mode*: Representation of the configuration of the motor system, including the enabled controllers and their relationships.
- *Actions*: An action express an elementary goal that the aerial robot is able to achieve by itself, using its own actuators. Actions might have a finite duration or infinite duration until they are disabled. An illustrative set of actions are: take-off, go to a point, move forwards, pick up item, drop item, rotate yaw, and land.
  Two categories of actions are considered: Executive actions are the ones accepted by the Executive System and work in present time. Deliberative actions involve planning, and therefore work in future time. These deliberative actions are only accepted by the Planning System.
  Actions might include, not only the symbolic name that expresses the action to be done, but also the complement of the action (called *action complement*). Examples of actions and their action complements are: go to point P, being the point P the action complement of the action go to point; pick up item A, begin the item A the goal of the action pick up item.
  Additionally, actions return a feedback (*action feedback*), that is a performance value about the incremental progress of an action. Similarly, once an action is finished (because it is completed, or because it is unfeasible), a performance value is returned (*Action result*).
- *Skills*: To represent a particular robot's ability the concept of skill is used. An illustrative set of skills is the following: reactively avoid obstacles, self-locate using visual markers, and recognize items. Skills can be active or inactive in a particular robot.
  In general, skills have influence in the behavior of actions. Thus, skills can be understood as global modifiers for sets of actions. Skills have infinite duration until they are disabled. The concept of skill followed in this work is defined in (Molina et al., 2016), in contrast to other meanings defined for skills in the literature of robotics.
  Skills might include, not only the skill by itself, but also complements that add some extra information.

For example, self-locate using visual markers with camera C, being C the complement that defines which camera measurements must be used to detect visual markers and perform the self-localization.
Similarly to actions, skills return a feedback (*skill feedback*), that is a performance value of the enabled skill.

- *Planning references*: They indicate a specific goal given to the planning components with the objective to create a plan formed by motion references. An example of a planning reference is a point P that is required to be achieved given to a trajectory planner.
- *Mission*: Complex goal to be executed by the robot (e.g. search an object in a field, deliver a parcel, etc.). It can be specified by human operators with a set of tasks that describe the different parts of the mission to be done.
- *Task*: It encodes the same concept than the mission, with the only difference of the size of the mission. A task is a complex goal with smaller complexity than a mission. The exact division between mission and task depends on the final user or developer decision.
- *Society knowledge*: This concept includes all the shared information between robotic agents. This shared knowledge might include raw sensor measurements, extracted features, situation awareness information, control and action goals, or in general anything. Depending on the nature of the shared information, it might be included in any component of the architecture, and therefore for clarity, it is omitted in Fig. 3.8.
- *Unexpected operation*: To permit the Supervision System to react to environment or operation changes, several variables require being monitored. In the case that one of the monitored variables acquires a certain value, a flag is enabled and this monitored event message is sent.
- *Process problem*: This includes all the possible problems related to the processes execution that are monitored by the process monitor of the Supervision System.
- *Process management*: To group the common processes messages related to their performance, state, problems, and in general, their management. For example a sensor might report a problem if the sensor stopped working (but the software process is still running); or it might report a performance issue if the measurement rate has decreased because the hardware is too hot and needs to cool down; or it might report an error if the execution failed and the process stopped.

3.3. Multi-robot approaches: decentralized vs. centralized

This section analyzes the multi-robot approaches tested with the proposed system architecture and discusses its possible extension.

According to (Parker, 2008), the types of interactions in multi-robot systems are divided along three different axes - the types of goals, whether entities have awareness of others on the team, and whether an entity's actions advance the goals of others on the team - in:

- *Collective*, in which entities are not aware of other entities on the team, yet they do share goals, and their actions are beneficial to their teammates;
- *Cooperative*, in which entities are aware of other entities, they share goals, and their actions are beneficial to their teammates;
- *Collaborative*, when robots have individual goals, they are aware of their teammates, and their actions do help advance the goals of others;
- *Coordinative*, when entities are aware of each other, but they do not share a common goal, and their actions are not helpful to other team members.

Since this division is a little weak and it might be debatable, the literature does not follow any naming convention and therefore, the previously presented names are mixed and used in a relaxed way.

The proposed system architecture does not limit to any kind of interaction, allowing all of them. The three last mentioned types of interactions might be carried out in a centralized way, where a central controller is performing decisions that are commanded to the robotic agents; or in a distributed way, where the decisions are performed by the robotic agents of the system. As stated before, the proposed system architecture is based on a distributed approach, being its advantages with respect to the centralized one, the following:

- it better scales up;
- communication data is minimized what normally represents a bottleneck;
- each agent of the system has a certain level of autonomy being independent of the central controller;
- better fault tolerance, since the central controller would be a critical essential element.

Moreover, multi-robot interactions might be classified, according to their duration, in (1) punctual; or (2) continuous. Punctual interaction, also called loosely-coupled, appear when a mission, a task, or a goal might be divided and assigned to every robot of the system. An example of a punctual interaction is the distribution of an area between agents for its exploration. Continuous interactions, also called tightly-coupled, appear when a

mission, a task, or a goal cannot be divided and therefore it has to be done simultaneously by several robots of the system in a precise way. An example of a continuous interaction is a flock formation. The proposed system architecture is not limited to any kind of interaction according to its duration. Nevertheless, punctual interactions have been mainly explored.

Despite being designed as a distributed multi-robot interaction system architecture, a centralized approach has been also tested. Fig. 3.2 shows a centralized explored approach for punctual mission level interaction. In this approach, presented in (Sampedro et al., 2016), a Society Coordinator agent acts as the multi-robot central controller. As the interaction is punctual in the mission level, the Society Coordinator is aware of the global mission that needs to be completed, and it generates submissions for every robotic agent, depending on the execution state of the mission, and the state of every robotic agent. The central controller avoids the need of having leader agents, negotiation processes or emergent behaviors.

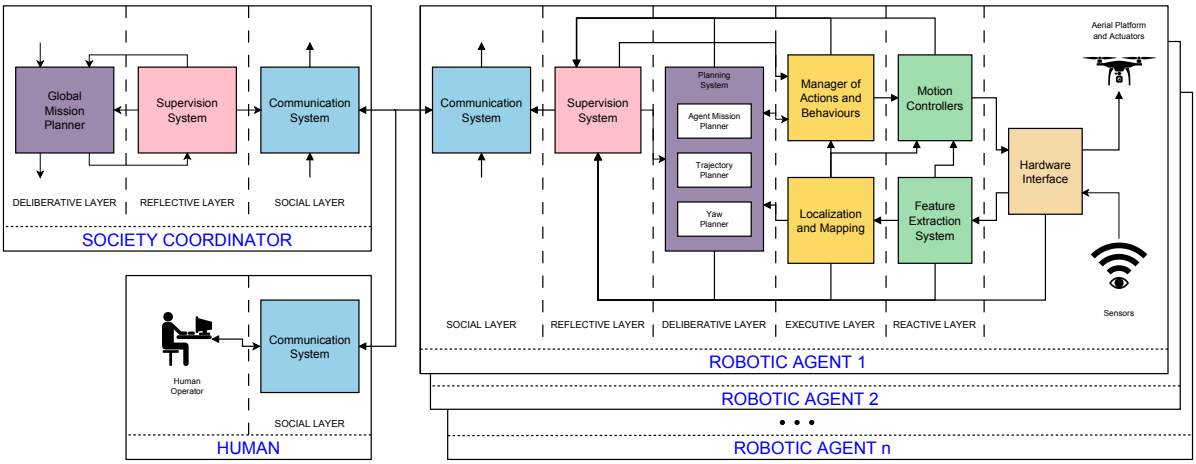


Figure 3.2: Centralized multi-robot approach modification of the proposed system architecture for punctual interactions at the mission level, presented in (Sampedro et al., 2016).

3.4. Historical evolution of the proposed architecture

As mentioned before, the proposed system architecture is the result of more than four years (since February 2013) of evolution and improvements. The methodology followed in this thesis, described in Section 1.5, included several cycles of challenge, proposal, evaluation, analysis, and generalization, until the final solution presented in this chapter has been obtained. This final solution has been demonstrated to be general and versatile enough for a wide range of applications in aerial robotics.

In this section, all the iterations of the proposed system architecture are briefly mentioned, analyzing how the new challenges that appeared were solved, and indicating the main features of every system architecture version upgrade.

IMAV 2012 competition

Previous works in aerial robotics were carried out by the thesis author in the framework of the IMAV 2012 competition (see Section 12.2.1). For this competition, an ad-hoc system architecture (Fig. 3.3) was specifically designed, showing a very limited versatility and a low level of autonomy.

The high difficulty shown to develop a fully autonomous and versatile system, and thanks to the experience acquired, triggered the development of a new system architecture for the versatile fully autonomous operation of aerial robotics systems.

This proposed architecture can be found on (Pestana, 2012), (Pestana et al., 2013a), (Pestana et al., 2014a), (Pestana et al., 2014b)

IMAV 2013 competition

The first time the system architecture was proposed was in the time frame of the IMAV 2013 competition (see Section 12.2.2). This yet very immature system architecture (depicted in Fig. 3.4) was focused on the proposed

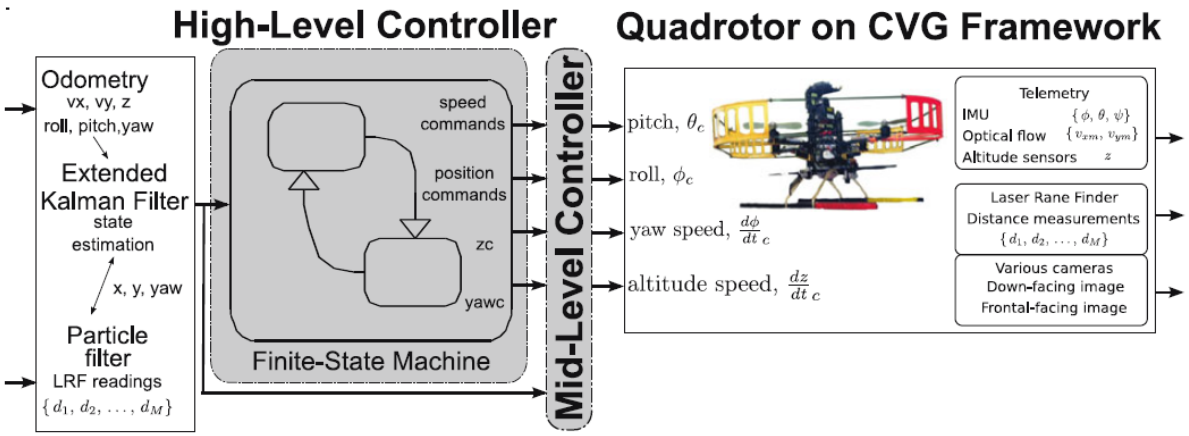


Figure 3.3: Proposed system architecture in the time frame of the IMAV 2012 competition. Figure from (Pestana et al., 2014b).

IMAV 2013 approach, where a multi-robot fully autonomous system formed by Parrot AR.Drone 2.0 aerial platforms were used. These robots were navigating in a partially structured environment using visual markers for localization and environment perception. Every robot had the same mission and one of the important designed emergent interaction between them was the collision avoidance.

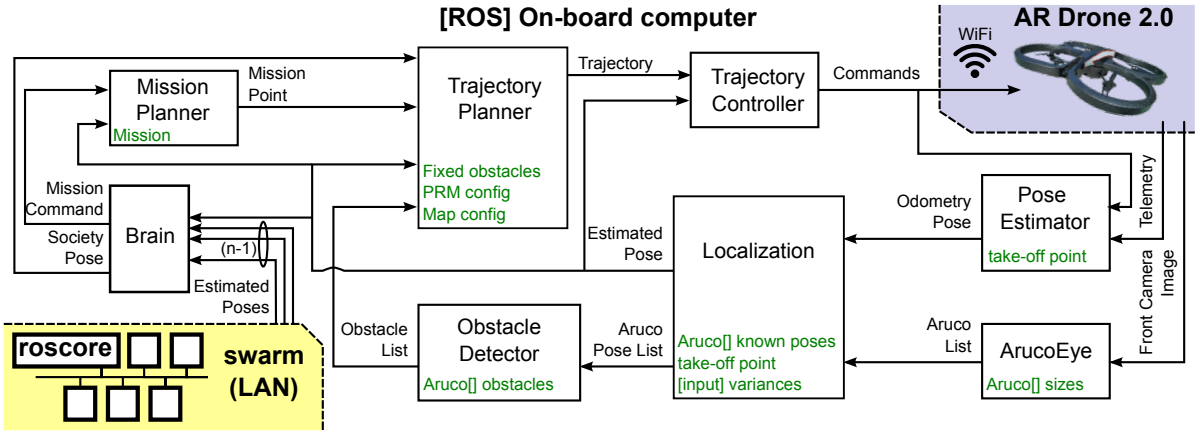


Figure 3.4: Proposed system architecture in the time frame of the IMAV 2013 competition. Figure from (Sanchez-Lopez et al., 2013a).

The Brain component included the functionalities of the current Supervision System, while the current Executive System functionalities were distributed along the Brain and the old Trajectory Controller. The Brain also included some of the current Planning System responsibilities, specifically an incipient Action Specialist. The Brain was also responsible for some of the tasks of the current Communication System. The given name of the Brain tried to exhibit the large number of responsibilities of this component.

This version of the proposed architecture can be found on (Sanchez-Lopez et al., 2013a), (Pestana et al., 2014e), (Pestana et al., 2016).

After IMAV 2013 competition

After the IMAV 2013 competition, and with the motivation of the achieved success in this competition, the previously proposed architecture was slightly refined and extended. The system architecture, depicted in Fig. 3.5, was extended to allow its usage on heterogeneous multi-robot systems.

It is worth to note that the previously proposed Brain was refined and renamed as Hypothalamus in an attempt of demonstrating that it was responsible for sequencing the actions and supervising the components, but was not responsible for the deliberative behavior of the robotic agent (associated with the Planning System).

This version of the proposed architecture can be found on (Sanchez-Lopez et al., 2014a).

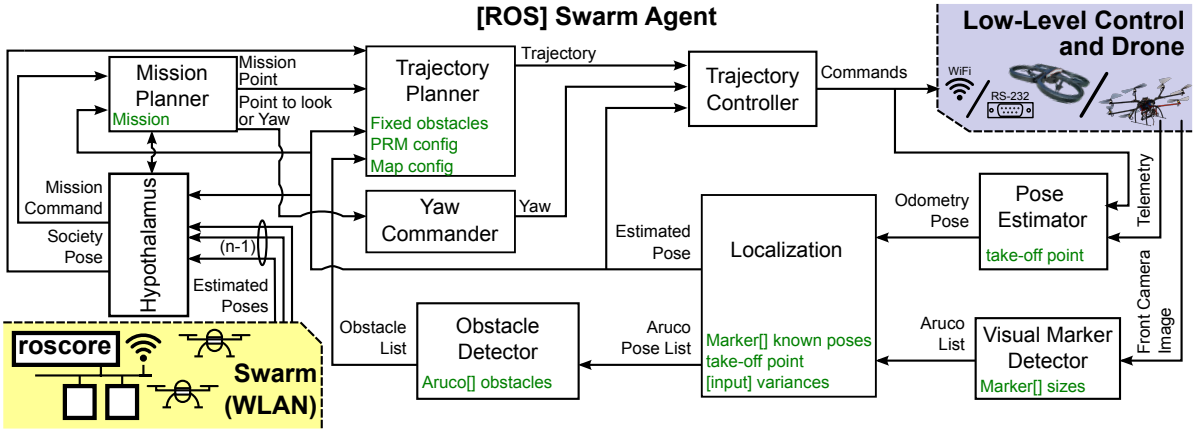


Figure 3.5: Proposed system architecture after the IMAV 2013 competition. Figure from (Sanchez-Lopez et al., 2014a).

IARC 2014 competition

The IARC 2014 competition (see Section 12.2.3) appeared as a new challenge that enforced the system architecture to be improved. Fig. 3.6 presents the system architecture proposed for the this competition. As this competition allowed only the usage of a single robot, the multi-robot features were omitted.

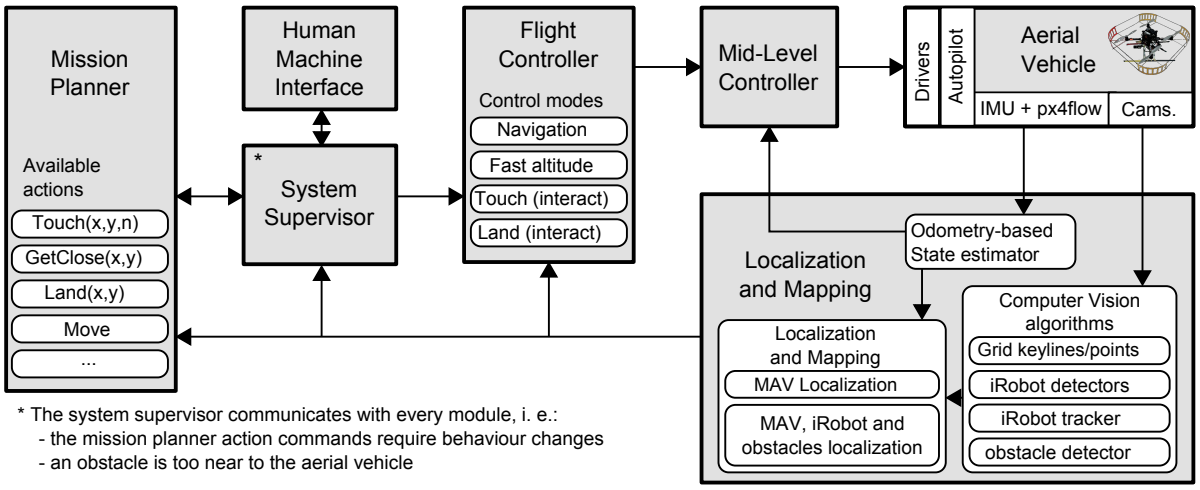


Figure 3.6: Proposed system architecture for IARC 2014 competition. Figure from (Sanchez-Lopez et al., 2015).

New important concepts appeared with this refinement:

- The Mid-level controller accepted four symbolic actions (take-off, land, hover and move), counting with an incipient very basic Executive System to sequence them, together with a basic Motor System.
- The Flight Controller accepted more complex symbolic actions (navigate in trajectory, position or velocity; perform fast altitude movements; or interact with the ground robots by means of touching them or landing in front of them). This component had internally a basic Executive System and a complete Motor System. Additionally, as it included some deliberative actions (like navigate in trajectory avoiding obstacles), part of the Planning System was included here, more concretely, some micro planners.
- The previously defined Brain and Hypothalamus were finally substituted to the System Supervisor. This component still included features of the modern Executive System, Supervision System, and the action specialist of the Planning System.
- The Human Machine Interface was created as the modern Communication System, relaxing the responsibilities of the former Brain.
- The Localization and Mapping component tried to group all the Feature Extraction System and the Situation Awareness System in an attempt of creating functional divisions of the components.

As can be extracted, the important concepts of actions (with different complexity) were firstly introduced in this architecture proposal.

This version of the proposed architecture can be found on (Pestana et al., 2014d), (Sanchez-Lopez et al., 2015).

After IARC 2014 competition

The experience acquired after the participation on the IARC 2014 competition, in addition to the need of having a versatile system architecture for single and heterogeneous multi-robot operation, allowing several kinds of missions in different environments, led to the design of a new version of system architecture. Fig. 3.7 covers this evolution. This architecture created for the first time an abstraction on the algorithmic implementation of the components, focusing on their functionalities.

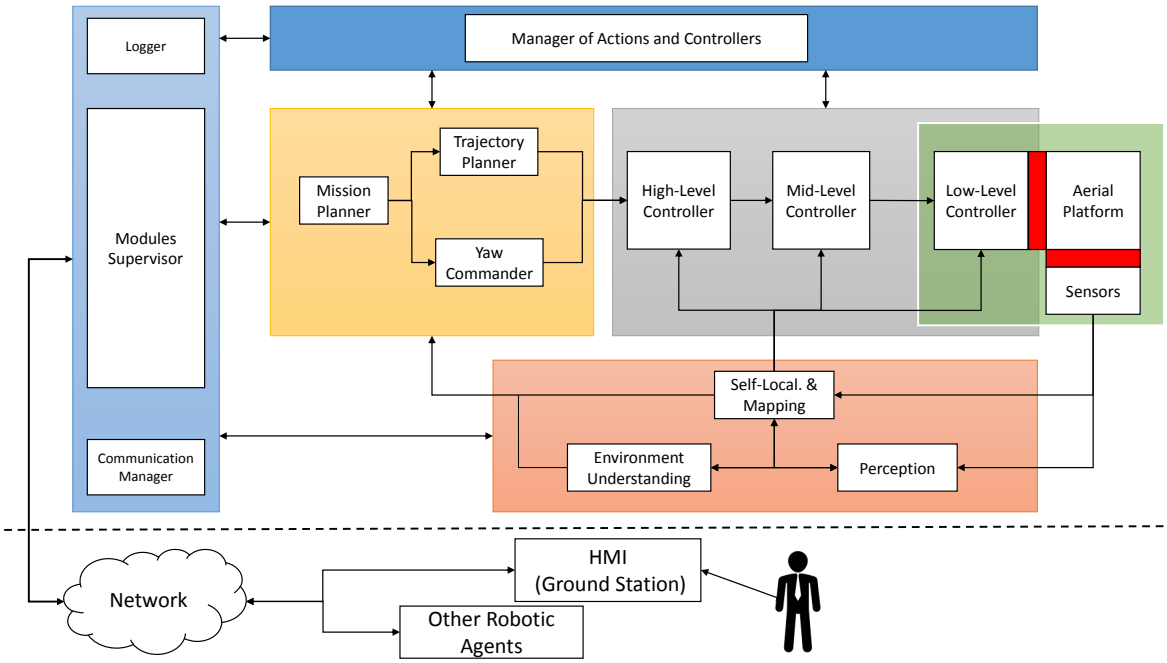


Figure 3.7: First versatile system architecture for single and heterogeneous multi-robot operation, allowing several kinds of missions in different environments. Figure from (Sanchez-Lopez et al., 2016b).

The total amount of the main current components of the system architecture were presented, although their functionalities were not exactly the same than the current ones:

- The Motor System was represented by the High-level, Mid-level and Low-level controllers, and the Yaw Commander;
- The Feature Extraction System was the Perception component;
- The Situation Awareness comprised the Self-Localization and Mapping, and the Environment Understanding;
- The Executive System was represented by the Manager of Actions and Controllers;
- The Planning System included the Mission Planner, and the Trajectory Planner;
- The Supervision System was equivalent to the Modules Supervisor;
- The Communication System was formed by the Communication Manager, the Logger, and the HMI.

This version of the proposed architecture can be found on (Sanchez-Lopez et al., 2016b). To demonstrate the versatility of this version of the proposed system architecture, several use cases, included the IMAV 2013 competition, and the IARC 2014 competition were shown.

Artificial intelligence formalization

A formalization of the system architecture was done, leading to Fig. 3.8. This formalization was in the direction of the main artificial intelligence theories, including concepts like the five layers: reactive, executive, deliberative, reflexive and social. Another contribution was the accurate definition of the functionalities of every component of the system architecture, together with the definition of the ontology that helped to clearly delimit the inputs and outputs of every system.

This version of the proposed architecture can be found on (Sanchez-Lopez et al., 2016c).

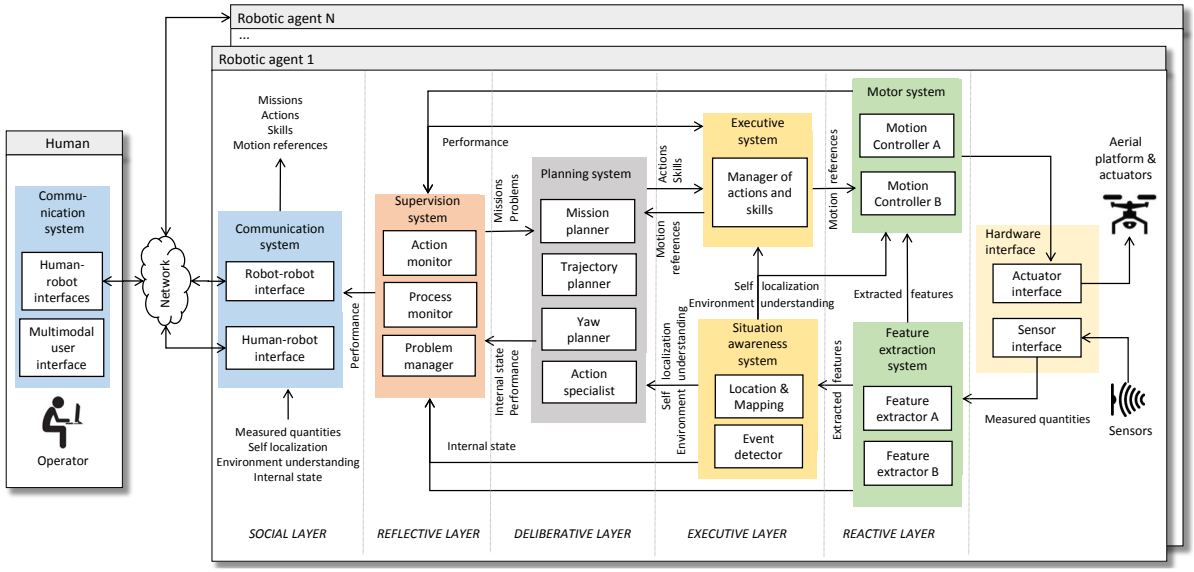


Figure 3.8: Five layered formalization of the system architecture. Figure from (Sanchez-Lopez et al., 2016c).

IMAV 16 competition

For the IMAV 2016 competition (see Section 12.2.4), the previously defined system architecture was barely modified. Fig. 3.6 presents the system architecture proposed for this competition.

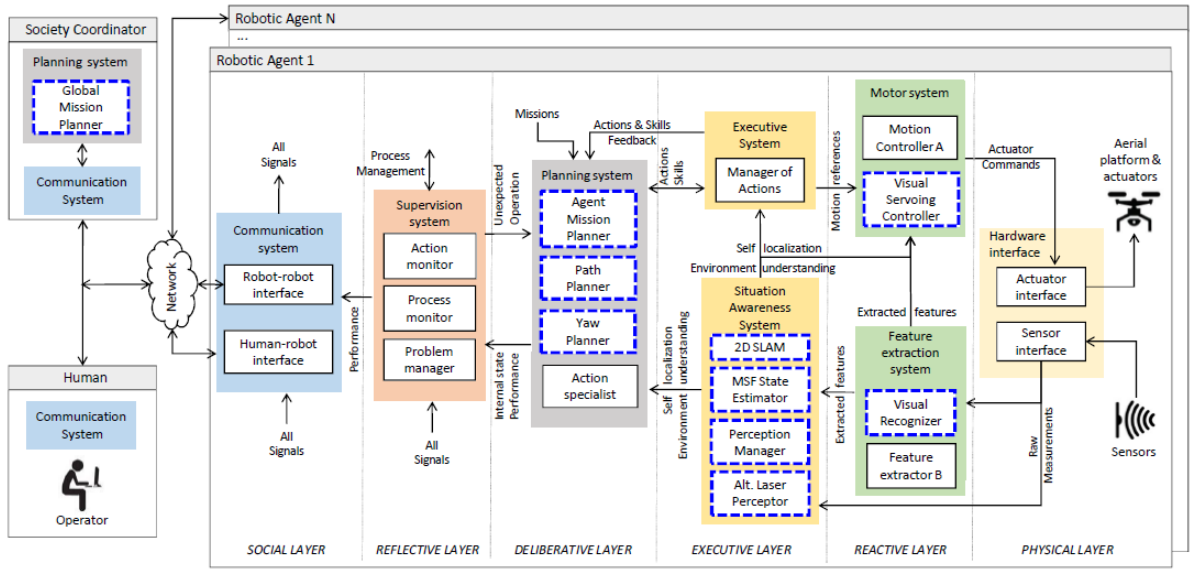


Figure 3.9: Proposed system architecture in the time frame of the IMAV 2016 competition. Figure from (Sampedro et al., 2017).

The only appreciable difference is that for this competition, a centralized approach, as discussed in Section 3.3, was used.

This version of the proposed architecture can be found on (Sampedro et al., 2017).

Artificial intelligence refinement

New minor modifications on the system architecture were done including the experience of the latest projects and the IMAV 2016 competition (Section 12.2.4). This is the version of the system architecture presented in this thesis, and can be found on (Sanchez-Lopez et al., 2017b).

Chapter 4

Subsystems of the Proposed System Architecture

This chapter describes in detail the main subsystems of the proposed system architecture for aerial robotics presented in Chapter 3. For each subsystem, a general description is provided, together with a brief indication of the available components in the current version of Aerostack (see Chapter 5) for illustrative purposes.

4.1. The Feature Extraction System

The goal of the Feature Extraction System is to transform the raw measurements provided by the sensors into a simpler and more usable information (see Fig. 4.1). These extracted features simplify the raw measurements in a way that the components that receive them are able to use them more efficiently.

The inputs of this system are the raw measurements given by the sensors; and the perception mode, given by the Executive System, that encodes the configuration of the complete perception system (including the sensors, the Feature Extraction System, and the Situation Awareness System). Its only output are the extracted features.

Similarly to the other systems, the specification of the Feature Extraction System imposes no prior restriction on the type or quantity of its inputs and outputs and the restrictions will only come with its particular implementation (and its related systems). Thanks to this on purpose open definition, any kind of feature extractor, using any kind of sensor information might be used in the proposed system architecture.

To ensure the correct and efficient operation of the Feature Extraction System, the Executive System enables, disables and connects, by means of the perception mode input, the available components in a way that they will never cause a fault on the system. It is, therefore, a requirement that the available components count with a proper start-up and a shutdown routine.

To illustrate the general design of the Feature Extraction System, the following components, available in the current version of Aerostack, can be considered:

- Signal filters that extract a particular frequency component of a measurement given by a sensor, for example, low-pass, high-pass, band-pass, etc.
- Computer vision based detectors and trackers, such as an helipad detector for shipboard landing (described in Chapter 6), an ArUco visual marker detector (Section 7.1), and a grid points detector (Section 8.1), among others.

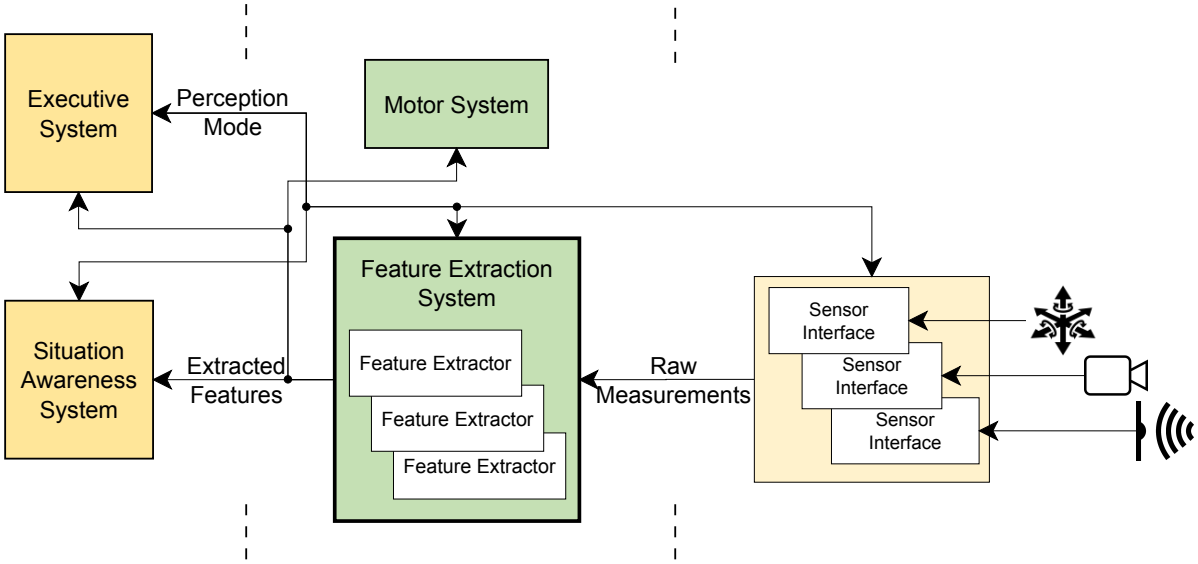


Figure 4.1: General description of the Feature Extraction System, and its relationship with the rest of the components of the proposed architecture.

- Point cloud based detectors.
- Etc.

4.2. The Motor System

The Motor System, represented in Fig. 4.2, has the responsibility to generate the actuator commands for the hardware elements of the robot, ensuring that the commanded motion references are followed, knowing the motion feedback.

The inputs of the Motor System are grouped in:

- *Motion references*, that are motion values to be considered as goals, and are given by the Executive System.
- *Motion feedback*, including the raw measurements given by the sensors; the estimated state of the robot and the environment given by the Situation Awareness System; and the extracted features given by the Feature Extraction System.
- *Motion mode*, that encodes the configuration of the Motor System and modifies its behavior, and that is given by the Executive System.

The only output of the Motor System are the actuator commands, used by the hardware of the robot.

Similarly to the other systems, the specification of the Motor System imposes no prior restriction on the type or quantity of its inputs and outputs and the restrictions will only come with the particular implementation of the Motor System (and its related systems). Thanks to this on purpose open definition, any kind of controller, using any kind of motion feedback might be used in the proposed architecture.

The Motor System comprises, therefore, a set of controllers that can be used independently or in cascade to allow the robot to follow a given motion reference. To ensure the correct operation of the Motor System, the Executive System enables, disables and connects, by means of the motion mode input, the available controllers in a way that they will never cause a fault on the system. It is, therefore, a requirement that the controllers included in the Motor System count with a proper start-up and a shutdown routine.

The Motor System can be implemented as the following controllers included in the current version of Aerostack:

- Low-level embedded controllers (Section 11.1.1).
- Navigation controllers (Section 11.1.2).
- Visual servoing controller (Section 11.1.3).

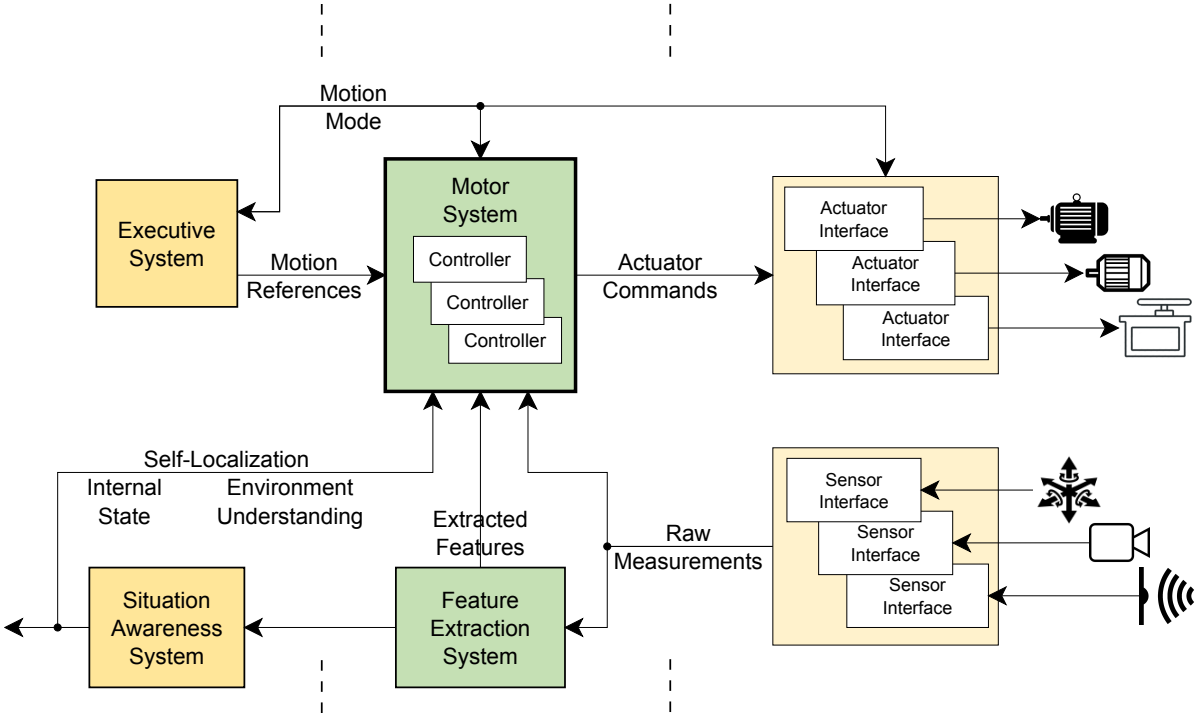


Figure 4.2: General description of the Motor System, and its relationship with the rest of the components of the proposed system architecture.

4.3. The Situation Awareness System

The Situation Awareness System, represented in Fig. 4.3, has the goal to interpret the information provided by the sensors and the Feature Extraction System to create a useful understanding of the current situation to be used in the decision-making and control processes. The Situation Awareness System is highly dependent on the mission, the environment, and the sensors and robot setup.

The inputs of this system are the raw measurements given by the sensors, the processed information given by the Feature Extraction System, and the motion references and actuator commands from the Motor System. It is worth to note that this specification of the Situation Awareness System imposes no prior restriction on the number, type, or nature of the used sensors, or the exploited extracted features. The sensors can as well be located on board the robotic agent or on ground. The restrictions will come only with the particular implementation of the Situation Awareness System.

In general and for completion, as the other subsystems, the Situation Awareness System could also include as inputs, some additional information coming from the Communication System and given by other agents of the multi-robot system (if any), like their estimated state, their estimated state of the map, or other information given by their sensors or Feature Extraction Systems. Including this information provided by other agents of the system allows the creation of a multi-robot Situation Awareness System.

The perception mode input, given by the Executive System, encodes the configuration of the complete perception system (including the sensors, the Feature Extraction System, and the Situation Awareness System).

The outputs of the Situation Awareness System, that define its functionality, are the following:

- *Self-localization*: estimation of the full situation state of the robot, as for example its pose, its velocity and its acceleration.
- *Internal state*: information related to the internal variables of the aerial robot like battery level, or kind, number, pose or range of the sensors equipped on the robot, etc.
- *Environment understanding*: estimate of the map of the environment in a format that is usable for the rest of the components of the proposed system architecture.

It is important to note that there is no restriction about how the Situation Awareness System represents internally the map or environment. This internal model needs to be converted to the appropriate representation, to make it usable for the rest of the components of the proposed system architecture, conferring more flexibility internally but ensuring its performance within the rest of the components.

The general design of the proposed Situation Awareness System can accept a wide diversity of components,

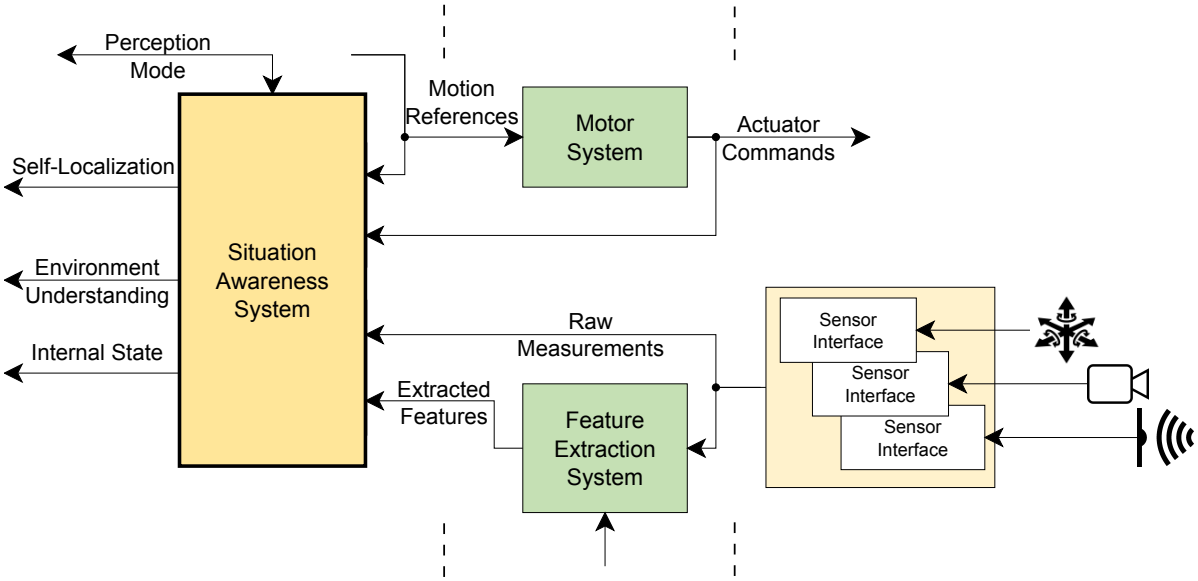


Figure 4.3: General description of the Situation Awareness System, and its relationship with the rest of the components of the proposed system architecture.

included in the current version of Aerostack, such as:

- An odometry based state estimation, described in Section 7.2.
- A visual marker based localization and mapping, analyzed in Section 7.3.
- A grid points based localization and mapping, tacked in Section 8.4.
- A keylines based arena localization, described in Section 8.5.
- A versatile visual markers based multi-sensor fusion state estimation, explained in Section 9.1.
- Two environment map reconstruction based on geometric primitives descriptors, described in Section 7.4 and Section 9.3.
- Etc.

4.4. The Executive System

The main goal of the Executive System (represented in Fig. 4.4) is to accept directives from the deliberative layer (or from a human operator) and to sequence them to be performed by the reactive layer. To be able to do this, the Executive System uses as feedback, the information given by the perception components.

Two different kinds of directives are accepted from the deliberative layer, actions and skills (described in Section 3.2).

An important property of the Executive System is that it creates a clear separation between two representation levels: (1) a symbolic level, where goals are described with linguistic symbols, which is very useful as an operator language for the specification of missions, and (2) a controller or perception level, where goals are described with quantitative values that are used as reference values for controllers, or commands of the components of the perception system.

The Executive System has a symbolic representation of the dynamic state of the aerial robot. For example, the state of the robot might be landed, taking off, hovering, or landing. These states can be divided into (1) states with a finite duration (for example, the taking off state automatically ends when the aerial robot has reached a specific altitude), and (2) states with undetermined duration (for example, the hover state only ends when the Executive System disables it).

These states and their transitions can be described using a finite state machine. The transitions between states describe the feasible actions that the Executive System can accept. The skills are modifiers of these states and transitions. For example, the action take-off, that represents a transition from the landed state to the taking-off state, only exists if a skill for measuring the flying altitude is enabled.

Actions are therefore augmented with skills. Some skills are required for the execution of a particular action, whereas some other skills are optional and only chosen by the operator. More concretely, the essential skills that are needed for the correct execution of a particular action, are marked as required skills of this particular action,

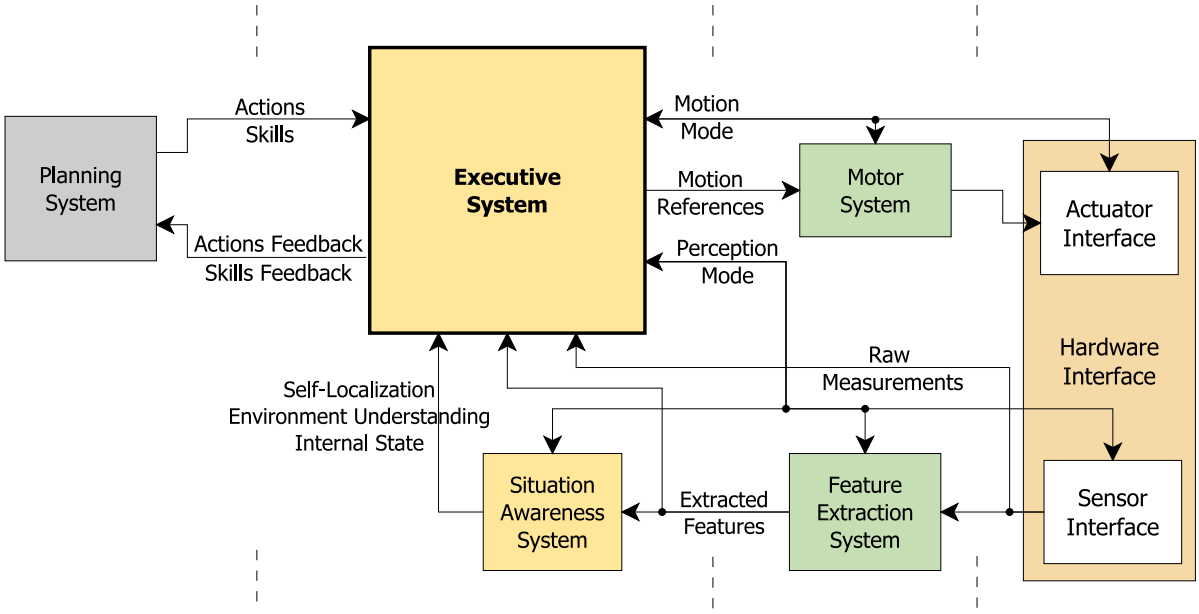


Figure 4.4: General description of the Executive System, and its relationship with the rest of the components of the proposed system architecture.

and this action cannot be carried out if these required skills are not performing properly.

The notion of skill is useful as an intuitive concept to express more easily what complex abilities should be active, without considering low-level technical details. Internally, a skill is automatically supported by a set of running processes. Thus, the activation of skills is associated with the increase of resource consumption (memory space, processing time, battery charge) so it is important to deactivate unnecessary skills when it is possible.

The Executive System has, therefore, a complete knowledge of all the possible actions and skills, together with their possible effects. The Executive System has the responsibility of ensuring that the transitions between states are feasible. For example, a take-off action is only allowed from a landed state, and never from the hovering state.

To carry out a particular action or skill, the Executive System is capable to enable, disable and reconfigure the motion components (Motor System and Actuator Interfaces) by means of the motion mode command; and to enable, disable and reconfigure the perception components (Situation Awareness System, Feature Extraction System, and Sensor Interfaces) by means of the perception mode.

The Executive System is responsible to prepare all the components (motion and perception components) involved in a specific requested action or skill, enabling and disabling them in the proper instant of time in a way that they do not collide with other components, monitoring their state, and in case of being unavailable, generating a response to the requested action or skill.

It is important to note that the Executive System is only checking the feasibility of an action or skill in the present time, unlike the Planning System, that is checking the feasibility in the future time. For example, since a take-off action requires to start the propellers of the aerial robot, if the propellers cannot be started, the Executive System will notice that the action take-off is not feasible, nevertheless, the Executive System will never check for example if the battery level would allow completing a specific mission before taking off.

Different architectures to implement the Executive System have been tested. The tested approaches range from a centralized architecture where a single component performs the complete functionality of the Executive System (an example of this centralized version is shown in (Molina et al., 2016) where the Executive System is implemented as a main process called Manager of Actions and Skills), to a distributed layered architecture (presented in Fig. 4.5), where there is a coordinator component (the Behavior Manager) that interacts with two different levels of specialized components (the first level distinguish between actions and skills, and the second level distinguish between motion components and perception components). A further analysis of the proposed architectures of the Executive System is out of the scope of this thesis, being only worth to mention that distributed layered architectures perform better than centralized ones, being their complexity and the effort to add new states and actions smaller.

The two following sections, illustrate the concepts of actions and skills by means of several examples to help the reader to understand them.

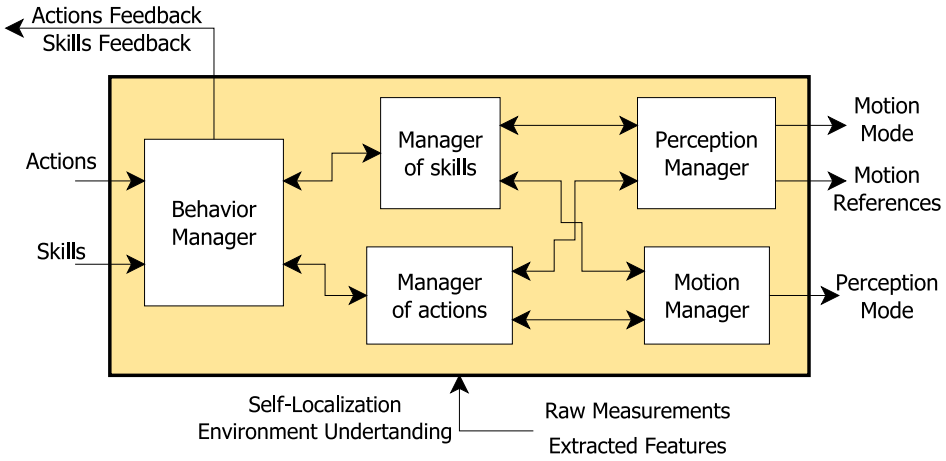


Figure 4.5: Detail of the Executive System.

4.4.1. Example actions

This section briefly describes three essential actions illustratively, the take-off, the landing, and the hover actions.

The take-off and landing actions are finite-time actions that automatically end after they are executed, unlike for example the hover action. It is worth to note that only the actions for take-off and landing in static, flat and known surfaces are going to be described. Take-off and landing in moving or sloped surfaces, are not analyzed in this section and might not be equivalent.

Both take-off and landing action controllers have to operate in some of the stages of the maneuver in conditions where some of the onboard sensors are unable to give proper measurements because they are very close to the ground (e.g. downlooking cameras are blinded, distance to ground is smaller than the operating range of the distance sensors, etc.) or their measurements are not accurate enough. Additionally, some hard to model effects (typically ignored in the models) like the ground effect, appear during this actions, creating hard to control disturbances. Because of these facts, both action controllers have been designed to operate in two consecutive stages: a feedback controlled stage and an semi open-loop controlled stage.

The hover action is an infinite-time action that should never end unless requested. Unlike take-off and landing actions, this action controller only has a feedback stage.

All the three example actions require some perception components enabled and working properly, including the sensor setup, and a Situation Awareness System to have a basic feedback of some elements of the state of the robot (e.g. velocity of the robot with respect to world). The activation of these perception components is carried out using the concept of skills of the aerial robot (as exemplified in Section 4.4.2).

Take-off action

The first stage is the semi open-loop stage when the robot is touching the ground and while it is very near to it. A predefined thrust command profile is commanded while requesting to keep the attitude of the robot parallel to the ground.

The feedback stage starts as soon as the robot is far enough to the ground, so the sensors are giving reasonable and useful measurements. A constant vertical velocity (climbing velocity) is commanded, while the other velocity commands are set to zero (horizontal velocity and heading angular velocity). This stage ends when a certain altitude is reached or after a certain time, ending, therefore, the take-off maneuver finishes.

Landing action

The landing action is the opposite of the take-off action. During the landing, the first stage, when the aerial robot is approaching the floor, is the feedback stage. A constant vertical velocity (climbing velocity) is commanded, while the other velocity commands are set to zero (horizontal velocity and heading angular velocity).

As soon as a certain altitude is reached, where some of the sensors are not properly working anymore, the semi open-loop stage starts, finishing automatically after a period of time, finishing as well, the landing action.

During this stage, a predefined thrust command profile is commanded, while requesting to keep the attitude of the robot parallel to the ground.

Hover action

This simple action requires the velocity controllers enabled, setting their commanded velocity to zero (vertical velocity, horizontal velocity and heading angular velocity).

4.4.2. Example skills

This section briefly describes two skills for illustrative purposes, the visual markers detection skill, and the visual markers based obstacle detection skill.

Visual markers detection skill

This skill has the objective of enabling a component of the Feature Extraction System for the detection of visual markers on the images provided by a camera, as described in Section 7.1. As an extra implicit requirement, a camera has to be enabled. Depending on the number of cameras of the aerial robot, and the number of visual marker detectors available, this skill might include two parameters, the camera(s) identifier(s) C, and the visual marker detector(s) identifier(s) D.

Visual markers based obstacles detection skill

This skill is a more complex example. It enables a component of the Situation Awareness System for the detection of obstacles, using the information given by the mapped visual markers (see Section 9.3).

It has some extra implicit required components, that must be enabled and identified by using the appropriate parameters of the skill. First of all, it requires to enable a state estimator to calculate the robot state and the visual markers pose (for example, the one presented in Section 9.1). Second of all, it needs one or more visual marker detectors (for example the one described in Section 7.1). Third, it needs to enable one or more cameras whose measurements are used by the visual marker detectors. And finally, it might require other extra sensors like an IMU.

As stated before, identifying these required components, sequencing their start-up, ensuring their adequate activation, and before finishing the skill, ensuring their adequate deactivation, is the responsibility of the Executive System.

4.5. The Planning System

The Planning System, represented in Fig. 4.6, generates goals to accomplish a particular complex mission, task or deliberative action. These generated goals are represented as executive actions and skills that are forwarded to the Executive System. Additionally, it reacts to changes in the operation provided by the lower-level layers and to unexpected operation events given by the Supervision System, generating new goals that modify the previously produced ones. Unlike the Executive System, the Planning System works in future time, taking into account the current state of the robot, predicting the consequences of the planned actions in the future.

The Planning System has to be able to generate goals fast enough to make possible an efficient reaction to changes in the mission, in the environment, or in the state of the robot. This is specially critical in aerial robots since their unstable nature disqualifies them to passively wait for a slow response of the Planning System.

The Planning System can be divided in three components:

- *Mission planner*. The mission planner (Section 4.5.1) is a deliberative component that receives as input a mission, a task or a deliberative action to be performed and generates as output a sequence of executive actions to be executed together with their corresponding required skills. The mission planner generates such actions considering the dynamic changes in the environment or in the operation.
- *Action specialist*. The action specialist (Section 4.5.2) helps the mission planner during the deliberation to anticipate if a tentative actions is feasible, according to the current situation. For this purpose, the action specialist has a knowledge of all possible actions and their effects. For example, the action specialist can verify in advance that a certain spatial point is too far to be reached, considering the current charge of battery. The action specialist is also able to predict physical magnitudes of certain actions such as required time, distance to cover, amount of battery to consume, required free space, etc. It is important to know, that this estimation is approximate, i.e. it is done using inexact models and help to find more efficiently

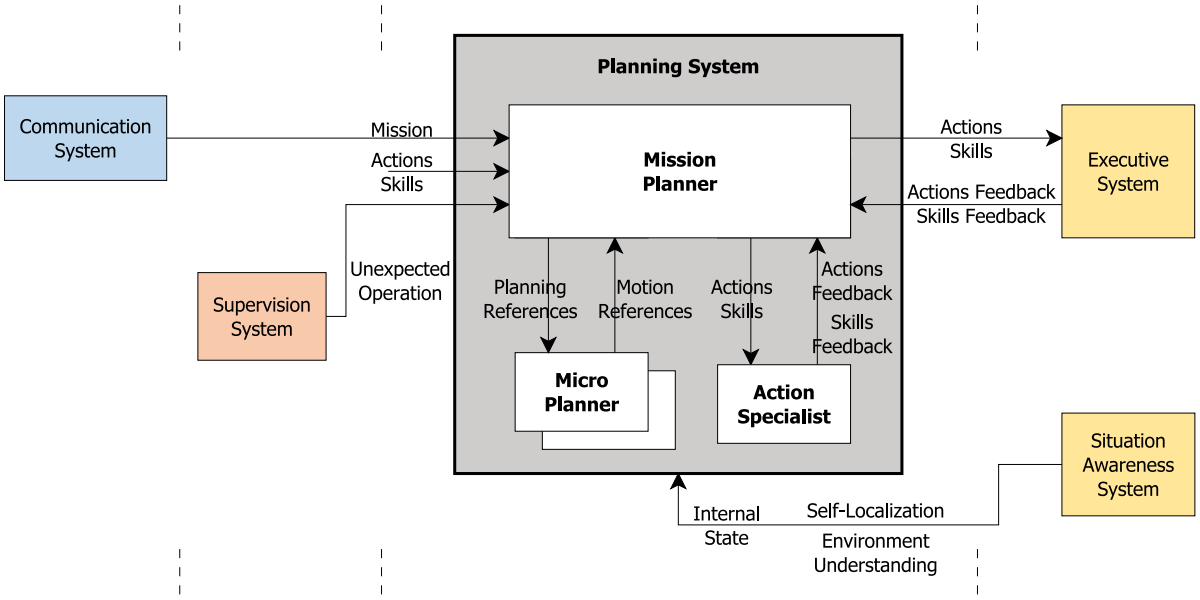


Figure 4.6: General description of the Planning System, and its relationship with the rest of the components of the proposed system architecture.

the solution, anticipating certain clear solutions. This means that, when the actions are executed, the robot can behave in a different way due to specific changes in the environment.

- *Micro planners* (optional, chosen by the mission planner). The micro planners generate motion references that can be followed by the robot, given the current (self-localization, and internal state) and desired (planning references) state of the robot, and the environment map (environment understanding). An example of a micro planner is the trajectory planner (Section 4.5.3), that generates trajectory references.

4.5.1. Mission planner

The mission planner receives as input a mission, a task, or a deliberative action to be performed and generates as output a sequence of executive actions to be performed together with their corresponding skills which the Executive System is able to manage. To do this, it is able to request motion references to several micro planners by sending them planning references. For example, the deliberative action "go to point (P) with active obstacle avoidance", being the point (P) the action complement that describes the coordinates of the the destination point, and it has to be transformed in the following executive action, "follow path (T) in path following mode", being the path (T) the action complement formulated as a sequence of waypoints. To do so, it is used a path planner (for example the one presented in Section 4.5.3) that generates a collision-free path (T) to reach the point (P).

The mission planner is able to react to changes in the operation provided by the lower-level layers. These changes in operation might include: changes in the environment (e.g. a new obstacle appeared) or in the knowledge of the environment (e.g. a new obstacle is mapped), changes in the internal state of the robot (e.g. low battery warning), changes in planned events (e.g. a planned action has finished, a planned trajectory is not collision-free anymore).

The mission planner is able to react to unexpected operations given by the Supervision System. These unexpected operations might be any kind of difficulties, including action problems (e.g. an action cannot be executed) or processes problems (e.g. a process has unexpectedly finished).

Finally, the mission planner has the responsibility to send to the Executive System, if feasible, the planned executive actions and skills.

Different task-based mission planners, that implements the features of the general proposed mission planner, by decomposing the complete mission in a task tree, are available in the current version of Aerostack and are presented in Section 11.2.

4.5.2. Action specialist

The main goal of the action specialist is to check the feasibility of a requested combination of actions and skills, generating a response indicating its feasibility (feasible or not feasible) together with its predicted

performance. It has the knowledge of all the possible actions, all the possible skills, and their effects.

The feasibility check of a combination of an action and skills includes: (1) the individual verification of the feasibility of the particular action, (2) the contextual verification to check the feasibility of the action in relation to other skills that occur at the same time, and (3) the temporal verification of the complete mission taking into account the temporal evolution of the previous set of actions and skills.

It is important to note that, unlike the Executive System, the action specialist is checking the feasibility of an action, not only in the present time, but also in the future time.

The action specialist might be implemented as the constraint-based action specialist presented in Section 11.3, and available in Aerostack.

4.5.3. Trajectory planner

The trajectory planner generates deliberative motion references for the aerial robot given planning references and the information of the situation awareness.

Both planning and motion references might be given in the workspace or in the configuration space and might include pose references, velocity references and acceleration references (among others) with or without time constraints. The generated motion references have to be compatible with the input of the used controllers available in the Motor System.

The trajectory planner has to include the knowledge of the dynamics of the aerial robot, together with the constraints of its actuation system.

For usual navigation tasks, like the path following, time is not included as a planning restriction (e.g. a particular point has to be reached in a particular instant time), and therefore, the trajectory planners do not need to take it into account.

In common under-actuated multirotor aerial robots, only four variables are controllable. Common under-actuated multirotor aerial robots can be considered as holonomic vehicles, so they do not present any planning constraint, if only their position and the heading (yaw angle) have planning restrictions (are given as planning references), being the other degrees of freedom, like their pitch and roll, velocity and acceleration, imposed by its dynamic model. Planning taking into account only pose references, without taking into account dynamic model related constraints is normally called path planning. This simplification is only valid for usual navigation tasks, like the path following, but is not valid for other more complex task, like performing aggressive maneuvers, where velocity and acceleration restrictions are needed to be taken into account.

Moreover, common under-actuated multirotor aerial robots are normally symmetric in their horizontal dimensions, and therefore, changing their heading angle is not needed to reach a particular position point, and consequently, the commanded heading and position can be decoupled, so the path planner might be divided in:

- *Position path planner*. It generates collision-free position references that can be followed by the aerial robot, given the current and desired position of the aerial robot, and the environment description. An implementation of a collision-free position path planner, available in Aerostack, is deeply described in Chapter 10.
- *Heading path planner*. It generates heading motion references that must be followed by the aerial robot. For example, the yaw value can be specified as a point to look or as a yaw to look. This can be used with different objectives, like maximizing the performance of the onboard sensors (for example a camera) or to achieve a particular goal (for example to watch over some target).

It is worth to highlight, for the sake of generality, that despite the simplification done in this section for common under-actuated multirotor aerial robots, the proposed system architecture allows using more complex trajectory planners, such as:

- Trajectory planners with time restrictions.
- Trajectory planners for aerial robots with more than four controllable variables (e.g. multirotors with tilting propellers or multirotors with manipulators).
- Trajectory planners with restrictions in other degrees of freedom of aerial robot different than the position or heading angle (i.e. for aggressive maneuvers such as crossing a small vertical hole).
- Trajectory planners for aerial robots without symmetry in the horizontal dimensions, where the position or heading cannot be decoupled anymore.

4.6. The Supervision System

The goal of the Supervision System (represented in Fig. 4.7) is to ensure the correct autonomous behavior of the whole aerial robot. It evaluates if the robot is actually making progress to its goals and to react in the presence of problems or unexpected situations (e.g. faults, lost communications, etc.) with recovery actions. The

Supervision System helps to provide therefore a fault-tolerance execution. In general, handling fault-tolerance typically consists of three steps: failure detection, notification, and recovery.

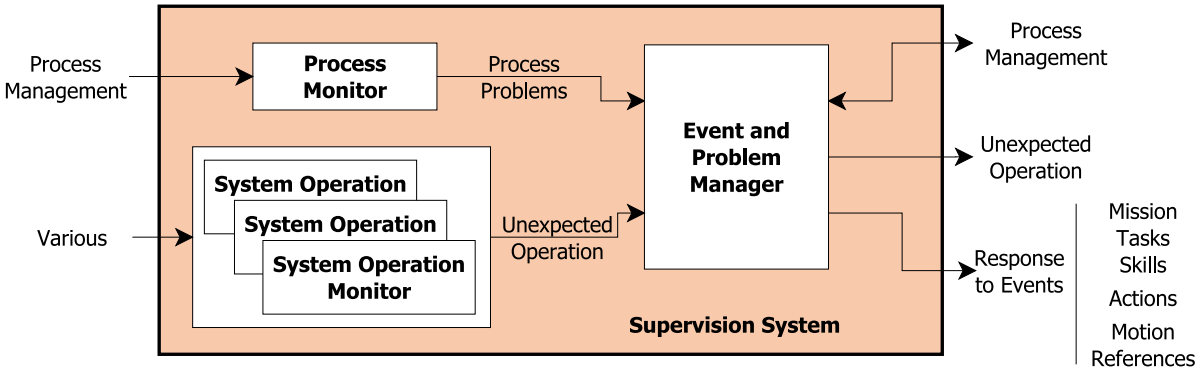


Figure 4.7: General description of the Supervision System. Since it is supervising all the other layers, it has a tight relationship with all the components of the proposed system architecture.

The general structure of the Supervision System includes (1) a set of specialists in kinds of events and problems, and (2) an event and problem manager.

Each specialist, called system operation monitor (analyzed in Section 4.6.1), is specialized in detecting a particular class of events or problems, receiving as input the necessary information about situation and actions, and generating as output a category of event or problem. A specific important system operation monitor is the process monitor (described in Section 4.6.2), that supervises the correct operation of the processes.

In addition to the system operation monitors, the Supervision System includes an event and problem manager (analyzed in Section 4.6.3), that is able to generate a response to events, malfunctions or unexpected situations, depending on the level of emergency and the nature of the event or problem received.

4.6.1. System operation monitor

The system operation monitors are specialized in detecting a particular class of events or problems, receiving as input the necessary information about situation and actions and generating as output a category of event or problem.

Examples of the system operation monitor (see Fig. 4.8) are the following:

- *Motor monitor*. It detects events and problems related to the Motor System and the Actuator Interfaces. An example would be a controller giving an incorrect actuator command, such as an *NaN* value or an infinite value.
- *Perception monitor*. It detects events and problems related to the Situation Awareness System, the Feature Extraction System, and the Sensor Interfaces. Examples of these events are incorrect measurements given by the sensors, like *NaN* values; or large covariances values in the estimated self-localization that require running re-localization algorithms.
- *Executive action monitor*. It monitors the executive actions and skills feedback from the Executive System. It supervises the execution of a requested action and informs when the action has been completed or when it has failed. For example, if the requested action is to move to a certain point, the action monitor verifies periodically the distance between the robot and the desired point and, when the distance is less than a threshold (established by a configuration parameter), the action monitor notifies that the requested action has been completed.
- *Deliberative action monitor*. It monitors the deliberative layer, in the same way than the executive action monitor.

4.6.2. Process monitor

The process monitor has the mission of supervising the correct operation of the processes. The process monitor is capable of monitoring all different processes hosted on different computers and is responsible for acquiring and informing about the different errors related to processes, including when a process stops its execution unexpectedly.

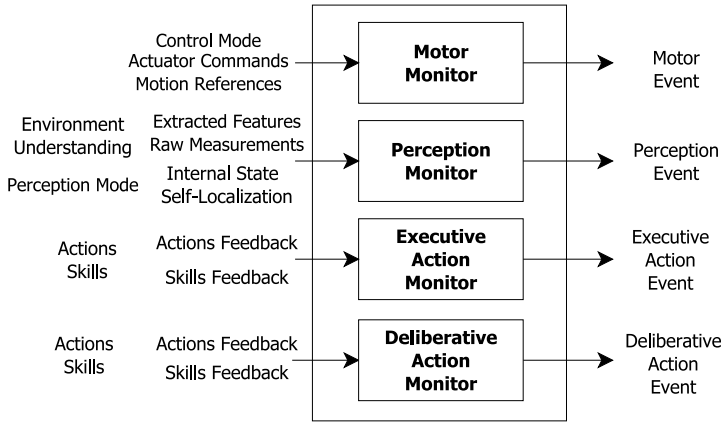


Figure 4.8: Example of system operation monitors.

To check if a process is alive, the process monitor uses a watchdog technique. This technique consists of periodically sending signals to a supervisor (in this case, the process monitor) to ensure that the supervised component is still alive. If the supervised component stops sending signals for a certain amount of time, the supervised component is considered to be offline. By using a watchdog, every process sends an alive signal to the process monitor, so that the monitor can track if a process has interrupted its correct execution.

In addition, the process monitor gets the execution state (e.g. paused, running, etc.) of the processes. Each process automatically notifies its execution state to the process monitor when the process sends the alive signal.

4.6.3. Event and problem manager

Recovery actions can be performed as a response to malfunctions or unexpected situations, detected by the problem monitors. In order to recover problems and depending on the level of emergency and the nature of the problem, the problem manager acts on a different level of the proposed system architecture. A priority scheme is needed since the presence of a failure can propagate several detected errors. For example, if the altitude sensor fails, it can generate a fault detection in the sensor interface but also in other processes that use the altitude measurement (for example the Situation Awareness System or the Motor System).

4.7. The Communication System

The goal of the Communication System, represented in Fig. 4.9, is to allow a robotic agent to exchange information with the human operators and with the rest of the robots of the system. This system is able, if needed, to establish a bidirectional communication with all the components of the agent, to permit this information transfer.

The Communication System can be separated into two main parts, (1) the robot side, that is included in every robot agent architecture, and (2) the human side, that has one instance per human operator.

Both sides include interfaces (in Fig. 4.9, called $X-Y$ Interface, being $X,Y = \{Robot, Human\}$) that convert data between different formats and networks allowing the intercommunication between all the agents (human or robotic) of the system. In general, the implementation of the communication interfaces is highly dependent on the kind of network used. For example, internally, every robotic agent might use ROS as the middleware for interprocess communication, but, between agents, MavLink middleware might be used for inter-agent communication.

The human side of the Communication System can also include multimodal user interfaces with graphics, speech, visual images, hand gestures, among others, that help to simplify the interaction between the human and the aerial robot.

The implementation of the multi-modal user interfaces, is done in the current version of Aerostack, by means of: (1) Command Line Interfaces (CLI), (2) Graphical User Interfaces (GUI) described in Section 11.4.1, and (3) Natural User Interfaces (NUI) presented in Section 11.4.2.

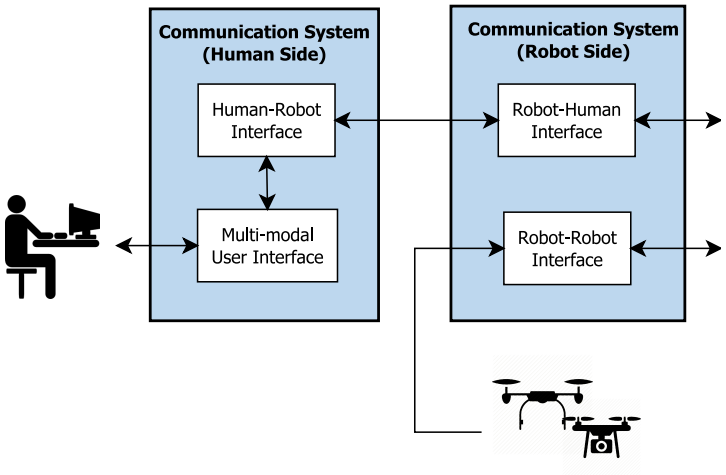


Figure 4.9: General description of the Communication System, and its relationship with the rest of the components of the proposed system architecture.

Chapter 5

Software Framework for Aerial Robotics

In order to test and evaluate the system architecture presented in Chapters 3 to 4, and to take advantage of it to solve the proposed problems tackled in competitions, self-proposed challenges, and public demonstrations, an implementation of this architecture is needed.

Implementation of the proposed system architecture

The implementation of the proposed system architecture has to be done in four dimensions: system hardware, components algorithms, components software, and system integration.

First, the hardware setup has to be selected and integrated into the used aerial platforms. The hardware setup is very dependent on the problem that needs to be solved. Different hardware setups for different applications are described in Chapter 12.

Second, the algorithms of the components of the system architecture have to be designed and developed. These algorithms are also highly dependent on the faced challenge, as well as on the hardware setup selected to solve this proposed problem. The design part includes the definition of the inputs and outputs of all the components, their concrete functionalities and their relationship with the rest of the components. It is important to remember that the proposed system architecture provides the generic design of every component, but the detailed design has to be done. The development of a component covers the detailed algorithm that is executed on the component, together with the tuning of its parameters if any. Several algorithms have been designed and developed, being the most relevant for this thesis described along Part III.

Third, the previously developed algorithms have to be properly implemented and translated into software components. This implementation includes the components software design, the components software coding, and the components software testing, together with their documentation and afterward maintenance.

Finally, the complete system, i.e. all the hardware and all the software components, has to be integrated and jointly tested for the final application in a similar operation environment. A fine tuning of the parameters of the components might be required.

After the system testing, an iterative process starts, reevaluating both the hardware setup, the developed algorithms, and the software implementation of the components, modifying them if required until the desired performance is achieved.

Aerostack

Aerostack, AErial RObotics software STACK, is a complete software framework that implements in a

software level the proposed system architecture for aerial robotics. This complete software framework has been designed, developed and implemented with the objective of ease the coding, testing, documentation, and maintenance tasks of the components to the developers, ensuring their compatibility and thus enhancing the usability of the software framework.

One of the main characteristics of the software framework is its modularity that allows creating independent software modules as processes with specific functionalities which can be exploited once connected to the rest of the architecture. The proposed software framework is taking advantage of ROS (Robot Operating System) (Quigley et al., 2009) as a middleware for inter-process communication. Other middlewares have been evaluated, but finally, ROS was selected since it is actively maintained and widely used by the scientific community.

The proposed software framework is organized in software packages with functional meaning, including common libraries, common processes, and optional processes.

It is worth to mention that thanks to the organization of the software packages of the framework, the migration to another middleware (in the case of a strong need) would be done in a limited number of packages that are specifically ROS-dependent, being able to reuse without any change a lot of packages.

Aerostack gathers all the packages with the software implementation of the multiple algorithms (e.g. controllers, state estimators, trajectory planners, perception components, etc.) that were used in the competitions, experiments, and demonstrations where Aerostack was employed. This helps to guarantee their inter-compatibility and maintenance.

This complete software framework has been released as open-source software under the GNU v3 license. The source code of the packages of the software framework is uploaded in a public well-known GIT server. The documentation, manuals, and use cases are uploaded in a public wiki. Finally, the support is done through an issue tracker, and directly by personal e-mail questions.

Aerostack features

Summing up, the main features of Aerostack are the following:

- Fully autonomous operation. A fully autonomous aerial system can be set up based on Aerostack.
- Multi-robot possibilities. Capability of realizing aerial multi-robot missions.
- Hardware agnostic. Compatibility with various multi-rotor platforms and sensors through the usage of a well-specified interface.
- Modularity. Its two-dimensional modularity arranges the components by their functionality as well as by their level of dependency.
- Scalability. It implements separately common processes and optional processes. This allows to modify or to develop new optional processes without the need of changing any other process.
- Versatility. Developers are able to build new optional processes easily by using the available common processes and libraries in addition to the well-defined internal messages and interfaces.
- Distributed processing. It allows the execution of the components both in one single computer or distributed over many computers.
- Flight proven and ready-to-use components with the capability of running simulations on big parts of the developed architectures.

Note that some of the previously enumerated features of the software framework are inherited by the fact of implementing the proposed system architecture presented in Chapters 3 to 4, such as fully autonomous multi-robot operation, hardware agnosticism, or the functional modularity.

Further details of the software framework related to its software engineering, including the software design diagrams (in Unified Modeling Language, UML) such as structure diagrams (e.g. class diagrams, package diagrams, etc.) and behavior diagrams (e.g. use case diagrams, activity diagrams, etc.), are out of the scope of this thesis and it is an active project in the research group. This thesis only presents the main characteristics of Aerostack from the robotics needs' point of view, avoiding to provide deep software related details, which are out of the scope of this thesis.

The remainder of the chapter is organized as follows: Section 5.1 goes deeper into the details related to modularity. The organization of the software framework can be found in Section 5.2. Section 5.3 gives more details about the development of new aerial robotics applications with some use cases as examples. The system requirements of the software framework are discussed in Section 5.4. Finally, Section 5.5 shows some features that support quality of the implementation.

5.1. Multi-process modular organization

One of the main characteristics of Aerostack is modularity. This allows creating independent modules with specific functionalities which can be exploited once connected to the rest of the architecture. This modularity allows the individual testing of modules, easing the project progress. Also, understanding the modules as input-output systems, permits to test in simulation the compatibility of their interfaces with the full system instance at hand.

A distinction is done between the intuitive meaning of a process (what it represents) and its computational support (how it is implemented). A process acts through time to change certain parameters of objects (e.g., certain physical quantities), consuming resources (e.g., time or space) to convert inputs into outputs. It is important to note that a process has a function, i.e., a purpose or practical use for which the process is designed or exists (the intention or the objective of the process).

Associating the idea of a component of Aerostack with a process helps to divide the whole problem of automated support for aerial robotics into partial functional roles. Each process is named as an agent according to its main function, for example, mission planner (main function: planning a mission) or obstacle recognizer (main function: recognize obstacles).

The computational support of a process is designed as an atomic executable unit (a data processor) that receives input data and, as a result of a certain information processing, generates output data. The idea of a process is also similar to the concept of an atomic functional block used in SysML with input/output ports. Processes are grouped in systems. A system is a complex module that includes a set of interconnected processes that provides a common functionality. The idea of the system is also similar to the concept of functional block (composite block) used in SysML, with input/output ports.

Aerostack uses asynchronous processing techniques (e.g., multitasking) that allow deliberative functions to execute independently reactive behaviors. This multitasking support is important, for example, to execute independently processes at lower level layers (executive and reactive, with frequencies between 10 Hz and 1000 Hz) and higher level layers (reflective and deliberative, with frequencies between 0.1 Hz and 10 Hz). Planning algorithms can be computationally more expensive, so they must be decoupled for real-time execution and avoid slow down the reaction time. In general, the inter-process communication (IPC) in multitasking operating systems, where different processes run concurrently, admits different communication mechanisms such as pipes, message queues, semaphores, sockets, shared memory, etc.

The multi-process implementation of Aerostack is supported by ROS (Robot Operating System) (Quigley et al., 2009). A process in Aerostack corresponds to the concept of a ROS node. Aerostack uses the inter-process communication methods provided by ROS, mainly: (1) a publish-subscribe mechanism using messages and topics, and (2) a request-reply scheme (services). Taking advantage of ROS features, Aerostack can perform a distributed processing, running its components both on one single computer or distributed in many computers and being only limited by the hardware.

Each process that operates in a particular aerial robotic system is programmed as a subclass of a common class called “DroneProcess” which provides default routines. When Aerostack is running for a particular aerial robotic system, each process is an executable instance of a program running on a computer (each process can also call subprocesses, child computational processes).

5.2. Division in groups of software packages

The presented software framework is organized in seven groups of software packages. This modular division, represented in Fig. 5.1, has been done following practical reasons according to different possible uses of the framework:

- The group of packages *Aerostack library* includes software modules that implement specific algorithms (e.g., computer vision algorithms, SLAM algorithms). This corresponds to ROS-independent programs that can be reused directly by developers to build new software for robotic platforms.
- The group of packages called *Aerostack ROS library* includes software modules that implement specific algorithms. This corresponds to ROS-dependent programs that can be reused directly.
- *Aerostack core* gathers the necessary common software to extend the functionalities of any process to a “DroneProcess”, as well as the definition of Aerostack custom messages. It also includes Aerostack-dependent helper packages. It is ROS-dependent.
- *Aerostack common processes* corresponds to the necessary common software to articulate a complete multi-layered architecture. This includes the Supervision System, the Executive System and the Communication System (with the human machine interface HMI).

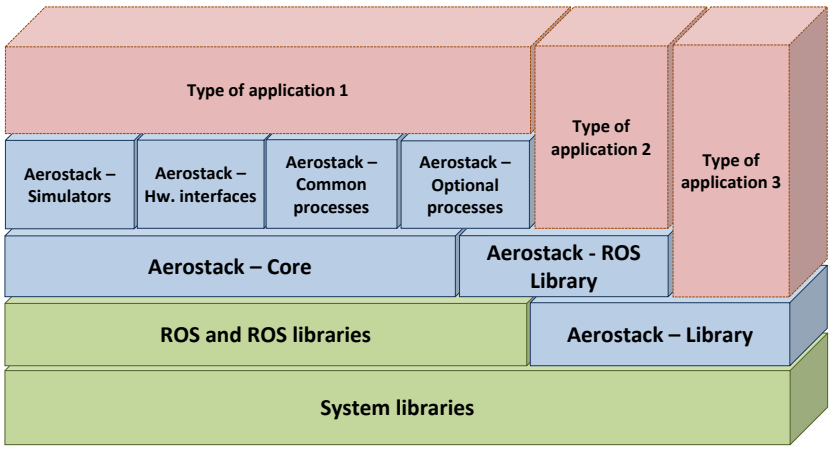


Figure 5.1: Software packages corresponding to Aerostack. The seven groups of software packages can be observed: library, ROS library, core, common processes, optional processes, simulators, and hardware interfaces.

- *Aerostack optional processes* includes alternative processes related to the following functional modules of the multi-layered architecture: the Planning System, the Situation Awareness System, the Feature Extraction System and the Motor System. For example, Aerostack provides optional modules for planning (e.g., trajectory planners), feature extraction (e.g., Aruco visual markers) and motion controllers (e.g., trajectory controllers or visual servoing controllers). This software implements the interfaces between the general algorithms of the library, ROS library groups of packages and Aerostack modules, as well as extending "DroneProcess" capabilities.
- *Aerostack hardware interfaces* includes the corresponding software interfaces for different aerial platforms, sensors, and actuators. For example, Aerostack includes interfaces for AscTec Autopilot, Parrot AR.Drone, Parrot Bebop, Mikrokopter Flight Controller as well as MavLink. This group of packages also includes interfaces for sensors such as UEye cameras and Optical Flow Sensors (Px4Flow), Hokuyo LIDARs and standard GPS, among others.
- *Aerostack simulators* includes simulation modules that are useful to test the correct behavior of the full system or specific modules before a field test. They include physics engines to ensure realism on the simulations.

In addition to these groups, another extra division based on the current status of the software packages is done in:

- *stack*: Fully functional software.
- *stack_devel*: Software under development that has not been released yet. It is stored in a private repository to avoid difficulties to the non-developer users.
- *stack_deprecated*: Deprecated software that will be removed in the future and therefore should not be used, but that is maintained to provide backward compatibility until all the existing packages are adapted.
- *stack_obsolete*: Obsolete software that has been removed and it is not useful anymore. It is only stored in a private repository for historical analysis.

5.3. Library of aerial robotic components: use cases

Aerostack provides an open library of reusable software components for aerial robotics implemented as processes.

The library of components provides modularity, so new processes can be included easily in the future, without changing the core of the system and the rest of processes. To create a new process it is necessary to program the corresponding algorithms and data structures in a class (e.g., in C++ language) and apply certain Aerostack conventions to be part of the library.

Processes can also be used by developers outside the proposed system architecture to reuse certain algorithms (e.g., computer vision algorithms) for a new aerial robotic system. In this case, they can be used as single classes without the dependence on the proposed multi-layered architecture.

To develop a complete software application for a particular aerial robotic system, the developer follows a compositional modeling approach using the processes from the library as model fragments or building blocks. The developer can select the appropriate processes from the Aerostack library and compose the global application

by assembling and adapting the selected components configuring their inputs, outputs and local parameters (if they have parameters). The developer selects the components according to the required features for the aerial robotic system. For example, if the developer wants to build an aerial robotic system with the feature self-localization by visual markers, she or he must select four processes: visual markers localizer, obstacle detector visual marks, obstacle distance calculator and self-localizer. The relation between features and processes is documented in the Aerostack library to facilitate this selection to the developer.

Bellow, different examples of use cases that show the de high level of versatility of Aerostack, as well as its scalability, are described.

Users can take advantage of Aerostack to operate aerial robotic system platforms in the following way:

- Perform teleoperated flights controlled by user commands using the Human Machine Interface. Examples of user commands are: take-off, move forward, move backward, turn, move up, move down, land, etc. The user operates with the keyboard, the joystick, and the mouse of the ground station computer to control the flight of the aerial robotic system.
- Following a selected object by means of visual feedback. The user can select the object to follow by selecting camera images using the Human Machine Interface.
- Carrying out a specific mission plan, defined by the user as a set of tasks or a set of waypoints.
- Flying within a specific spatial area, with certain limits defined by the user.
- Use a specific physical configuration that modifies significantly the size or weight of the aerial robot. The flight controllers can be adapted by the user to the new configuration (in an XML file).

Developers who are familiar with software programming can use Aerostack to operate new aerial robot with the following characteristics:

- An aerial robot with new specific types of sensors. The developer can reuse Aerostack but the developer needs to program and integrate new software modules to process the information from the new sensors.
- An aerial robot with a different physical platform. The developer can reuse Aerostack for different aerial robotic platforms, but the developer needs to program the interfaces between Aerostack and the actuators of the new physical platform.
- An aerial robot with one or several software components that substitute an existing software component (for example, a new localization and mapping) but have the same inputs/outputs as the previous one. The developer can reuse Aerostack, but the developer might need to substitute the core algorithm used by the previous component within the package of optional processes.
- An aerial robot with one or several software components that substitute an existing software component (for example, a new localization and mapping) that have a different inputs/outputs map than the previous one. The developer can reuse Aerostack, but the developer might need to program and integrate a new software component within the package of optional processes. Additionally, the developer might need to extend the functionalities of the connected components to be consistent with the new inputs/outputs map.
- An aerial robot with new functionalities that needs additional software components. The developer can reuse Aerostack but the developer needs to program and integrate the new software modules. Additionally, the developer might need to extend the functionalities of the connected components to extend the functionalities of Aerostack.

It is important to emphasize that any of the available components of Aerostack can be replaced by a similar counterpart at the simple cost of developing an interface with Aerostack (e.g. the trajectory controllers given by the “PX4 Flight Stack” could be used within Aerostack).

5.4. System Requirements

Aerostack requires a computer with Ubuntu Linux 14.04 or above with ROS Indigo or above installed. The preferred programming languages for the components are C++ (some packages make use of C++11 standard) and Python.

Aerostack has been run in a wide variety of computers, including the following:

- ARMv7-A architecture based microcomputers: ODROID-XU3, and ODROID-XU4 (a 2.0 GHz ARM quad-core processor, 2 GB of memory).
- Intel architecture based microcomputers: AscTec Mastermind (Intel Core i7 processor, 4 GB of memory); Intel NUC nuc5i7ryh (Intel Core i7 processor, 4 GB of memory).
- Intel architecture based laptops: average laptops (different Intel Core i5 and i7 processors, 4 and 8 GB of memory).

But it is important to note that the required computing power depends on the selected algorithms of the components for a specific application.

5.5. Quality features of the implementation

All the software packages provided in Aerostack are flight-proven and ready-to-use, and therefore, the users and developers of Aerostack are able to count on with these packages in their applications, speeding up their developments and allowing them to test their algorithms even in early stages of the project.

A complete and uniform documentation, both in source code and text documents, has been produced to help users and developers, creating basic rules that must be followed to ensure its compatibility, enhancing the usability of the software framework.

Aerostack counts also with manuals and tutorials that present the main aspects of Aerostack with cases of uses and examples ranging from basic users to developers.

To ease the download and installation tasks for non-expert users, an installer has been created, what allows its complete installation (with all additional dependencies except ROS) with one single Linux command.

Moreover, the complete research group (Computer Vision and Aerial Robotics, CVAR), to which the thesis author belongs, is supporting, maintaining and regularly updating Aerostack as one of the main group's research lines.

Part III

PROPOSED ALGORITHMS

Chapter 6

Helipad Detection and Reconstruction for Shipboard Landing

This chapter presents a solution capable to detect an helipad mark on a monocular color image and to recover its pose with respect to the camera sensor, using the knowledge of the intrinsic parameters of the camera (camera calibration) and the shape and dimensions of the helipad mark.

The helipad[1] consists on a white H surrounded by a white circle (Fig. 6.1) painted on the flat heliport surface (normally gray). These marks are the most extended marks to indicate the presence of a heliport surface.

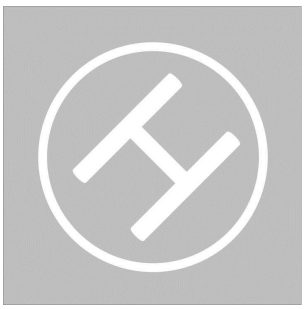


Figure 6.1: Typical helipad marks used in the proposed solution.

The helipad is not assumed to have any salient keypoints features (like SURF or ORB features). Lines might be used as image descriptors since they are present in the H mark of the helipad. Nevertheless, the most salient and high-level image descriptors of the helipad are both white marks (an H surrounded by a circle) painted on a normally gray surface. The proposed solution is based on these high-level descriptors.

In order to be able to recover the pose of the helipad with respect to the camera sensor, p_H^C, the intrinsic parameters of the camera (camera calibration) and the shape and dimensions of the helipad are needed to be known. Assuming to have available the knowledge of the shape and the size of the helipad is acceptable since such information might be transferred to the aerial robot before the landing stage on a specific heliport.

[1] The word "heliport" is used to refer to the whole surface to land, and the word "helipad" to refer to the drawings painted on the heliport.

Reference frames

The reference frames involved on the proposed helipad detection and reconstruction are summed up in Fig. 7.2.

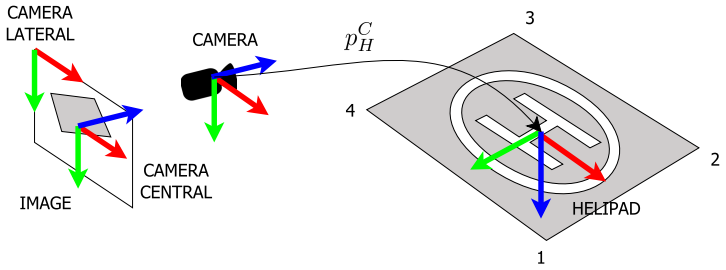


Figure 6.2: Reference frames involved on the presented helipad detection and reconstruction.

The camera reference frame (also called camera central reference frame) is rigidly attached to the camera of the robot, being the z-axis perpendicular to the image plane and in the direction of the environment. The y-axis is in the direction of the increasing vertical pixels of the image, and the x-axis in the direction of the increasing horizontal pixels of the image.

The camera lateral reference frame is a 2D reference frame rigidly attached to the top left corner of the acquired image of the camera, being its y-axis in the direction of the increasing vertical pixels of the image, and the x-axis in the direction of the increasing horizontal pixels of the image.

The helipad reference frame is rigidly attached to the helipad, being the plane formed by its x-axis and y-axis is parallel to the helipad surface. Its position and orientation are arbitrarily defined by the user.

The remainder of the chapter is organized as follows: in Section 6.1, the proposed helipad detection, and reconstruction algorithm are presented. In Section 6.2 the performance of the proposed solution is evaluated both with simulations and with real experiments. Section 6.3 discusses the main weaknesses of the proposed solution, analyzing the way to overcome them and pointing out some lines of future work. Finally, Section 6.4 sums up the contributions of the chapter.

6.1. Helipad detection and reconstruction algorithm description

This section deeply describes the main details of the proposed algorithm, which can be divided into two sequential stages:

1. Helipad detection (Section 6.1.1): the helipad is detected in the image, obtaining the four corners that describe it.
2. Helipad reconstruction (Section 6.1.2): the pose of the helipad in camera coordinates is calculated.

6.1.1. Helipad detection

The detection of the helipad in the image is carried out by means of a computer vision algorithm that has the following sequential stages described below:

1. Image preprocessing.
2. Heliport zone extraction.
3. Helipad marks segmentation.
4. Helipad marks classification.
5. Helipad detection.

The computer vision algorithm has been developed maximizing its accuracy, performance, and robustness. False positives are avoided using a large decision tree. False negatives (no measure when it has to be measured) are preferred over false positives since they are more easily manageable by a following state estimation stage.

Image preprocessing

After the acquisition of the RGB image (Fig. 6.3), it is converted to an intensity image and to the HSV (Hue-Saturation-Value) color space. Both converted images are preprocessed with a mean filter and then, with an opening morphological transformation (Gonzalez and Woods, 2002).

Figure 6.3: Example of an acquired image.

This stage prepares the image for the following stages, allowing them to work faster, more accurately, and with higher performance and robustness.

Heliport zone extraction

This stage uses the processed HSV color image from the previous stage to find a candidate heliport zone. A color thresholding is done to get a binary image with the gray pixels of the heliport. After the thresholding, a median filter followed by an opening morphological transformation are applied. Then, the blobs are extracted and the smaller ones are deleted. These steps allow clearing all the noise present in the image and all the small regions segmented. After that, the remaining blobs are filled in, and an OR logical transformation is applied to get a whole binary image that represents the candidate pixels to belong to the heliport.

Finally, a per-element multiplication of the intensity image obtained in the first stage and the binary image of the heliport zone candidate pixels is done, getting Fig. 6.4.

Figure 6.4: Example of an image after heliport zone extraction.

The reader must note that this heliport zone extraction stage gives not only the gray colored pixels of the heliport but also the white ones that belong to the H and circle marks because of the blob's filling step done.

It is worth to mention that this stage gives also other gray colored regions that can be visible in the image, giving some false positives that will require being filtered in the next stages.

Helipad marks segmentation

This stage extracts the helipad marks (the H and the circle) candidates present on the previously calculated candidate heliport zone.

The intensity image coming from the heliport zone extraction stage is thresholded looking for the white pixels of the helipad marks (the H and the circle). In order to remove noise and prepare this binary image, it is again processed with a median filter and an opening morphological transformation. After this, the resulting blobs are calculated, filtering those with a small area, getting Fig. 6.5.

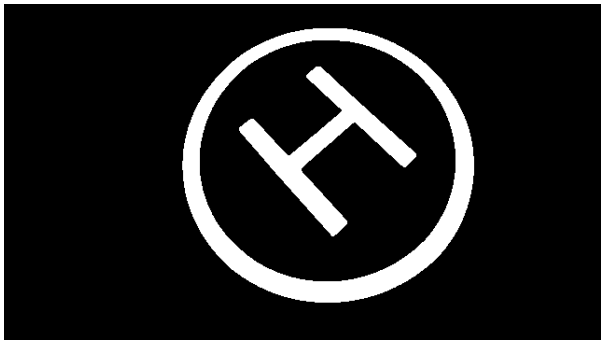


Figure 6.5: Example of an image where the helipad marks has been segmented, but not classified yet.

Helipad marks classification

This step filters out the candidate helipad marks from the previous stage to get a single and valid helipad. To do so, it previously requires classifying the candidate helipad marks.

The classification of the helipad marks is done with a decision tree, where, each level gives some false positives, but however, at the end of the tree, one single solution is provided. With this methodology, the speed and specially the accuracy of the proposed algorithm is improved. In the first stages, an individual classification for Hs and circles is done. Then, the obtained individually classified candidates are used (if found) to classify them jointly and verify them.

The first level of the decision tree is a fast classification using the Euler number (Euler number = connected components − number of holes) of each blob. H blobs have a Euler number equal to zero, and circle blobs' Euler number is one. Blobs with different Euler numbers are discarded. The Euler number is a scale, translation, rotation and homography invariant feature.

The second level is a classification with a multi-layer perceptron (MLP) artificial neural network (ANN) with ten neurons in the hidden layer, tree outputs (H candidate, O candidate and other) and five inputs (see Fig. 6.6). The inputs of the neural network are obtained after a principal component analyze (PCA) applied to the first seven invariant Hu moments of each blob. Hu moments are invariant to scaling, translation, and rotation and are frequently used in Optical Character Recognition (OCR), (Chen and Petriu, 2003; Sanjeev Kunte and Sudhaker Samuel, 2008; Kmiec, 2011). It worth to note that the homography transformation modifies a little the Hu moments, but they still can be used in the decision tree if enough training samples have been used to train the ANN.

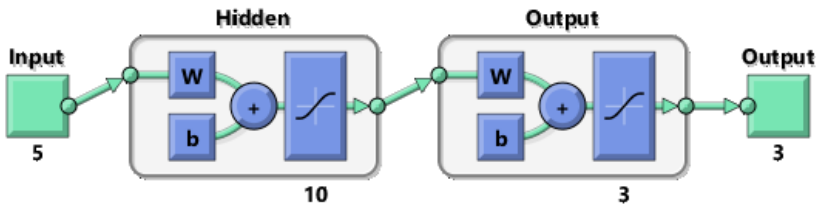


Figure 6.6: Multi-layer perceptron (MLP) artificial neural network (ANN) with ten neurons in the hidden layer, tree outputs (H candidate, O candidate and other) and five inputs, used in the second level of the classification tree.

A third classification level uses the signature of the H candidates. The signature is invariant to rotation and translation; it preserves its shape to scaling; and some features of the shape (relative maximum and minimum) are preserved to homography transformations. The H signature has four relative maximum and four relative minima. The four maximum are the external corners of the H; and the four minima are the bisectrix of the horizontal segment (above and below the centroid), and the bisectrix of the vertical segments (the external points). As the H has more information than the circle (because it is less symmetrical that the circle), this level might only be applied to classify H candidates.

Because of the symmetry of the H, the four maximum points of the signature can be connected, forming a quadrilateral polygon whose center should be near to the centroid of the H blob. The same phenomenon appears with the minimum points. The fourth level in the classification tree checks the fact that all the previously

calculated three centers (center of maximum points of the signature, center of minimum points of the signature and centroid of the blob) have to be near in the image plane (distances in pixels have to be small).

The fifth level checks the distance between the vertical straight lines of the maximum points (vertical segment of the H) and the points of the minimum that should be in the vertical segment of the H. This distance has to be small (ideally zero).

Hitherto, the proposed classification tree uses only individual features of each mark to achieve its task. The following stages select one of the candidate H blob and one of the candidate circle blob among all the resulting candidates, analyzing the joint compatibility. This is done for all the candidate blobs. The sixth level is based on the knowledge that the H has to be inside the circle. The seventh and last level calculates the coefficient between the area of the H and the area of the circle, which should be, more or less, a constant value. In these two last levels, all H and circle blob candidates are tested, discarding those that do not satisfy the conditions checked in these last levels.

The output of this stage are the helipad marks (a single H and a single circle) extracted from the image (Fig. 6.7a and Fig. 6.7b).

(a) Example of circle selected blob.

(b) Example of an H selected blob.

Figure 6.7: Examples of the helipad marks extracted of the image.

Helipad extraction

In this last detection stage, the corners of the heliport are computed. With the position of the corners of the H blob of the helipad in the image (obtained thanks to its signature), the homography transformation matrix between these points of the image and the same points in a predefined target image can be calculated. Then, using the homography matrix, the corners of the helipad are computed knowing where the corners of the helipad are located in the target image (Fig. 6.8).

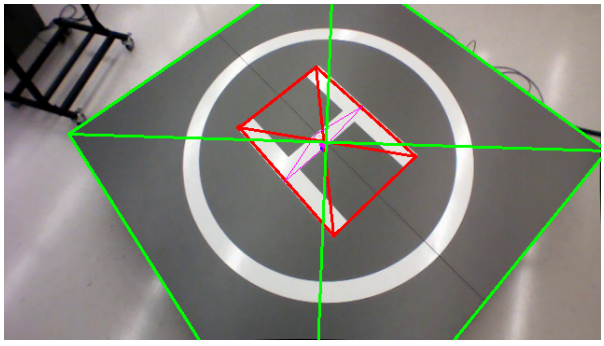


Figure 6.8: Example of output image after the computer vision algorithm. In green, the segmented helipad. In red, the H corners (maximum of signature). In purple, the minimum of the H signature.

6.1.2. Helipad reconstruction

Once the helipad has been detected, i.e. the corners of the helipad in the image have been obtained, the 3D reconstruction of the helipad has to be performed, that is, the 3D pose of the helipad with respect to the camera

frame has to be reconstructed (see Fig. 6.9).

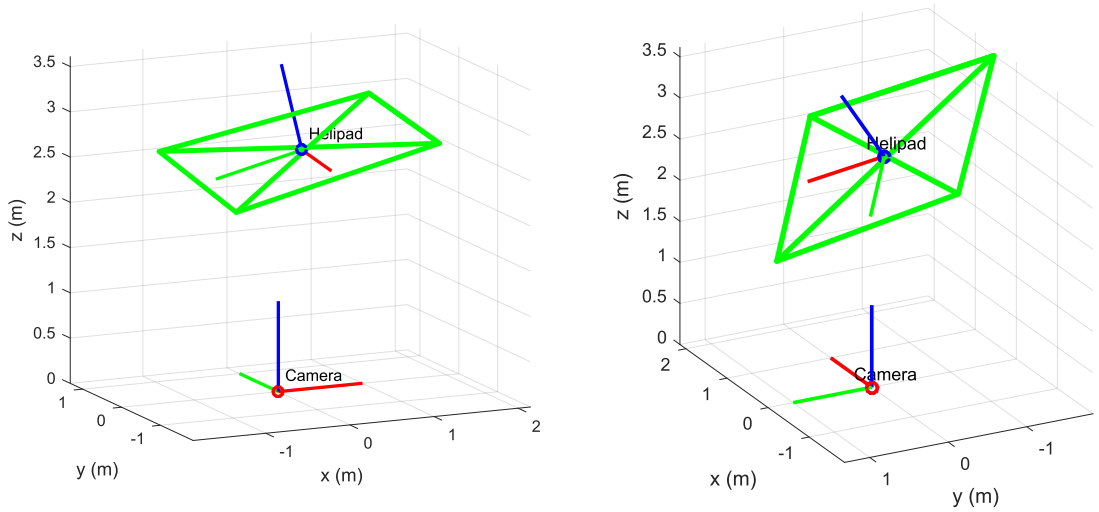


Figure 6.9: Example of the helipad reconstruction after the computer vision algorithm. The camera is fixed in the point $(0,\ 0,\ 0)$, looking upwards.

The intrinsic camera parameters are assumed to be known (the camera has to be previously calibrated, obtaining scaled focal distances f_x and f_y and k_y and principal point c_x and c_y). Additionally, the shape and dimensions of the helipad are required.

An state of the art P*n*P algorithm could be used to directly calculate the pose of the helipad with respect to the camera frame, (Wu and Hu, 2006), (Leng and Sun, 2009), (Li et al., 2012), (Hesch and Roumeliotis, 2011), (Lepetit et al., 2008). Nevertheless, a custom algorithm is used. This algorithm, first of all, reconstruct the 3D position of the helipad corners, and after that, the pose of the helipad is recovered.

Helipad corners reconstruction

To reconstruct the 3D position of the helipad corners in camera coordinates, a minimization by means of a Levenberg-Marquardt algorithm is carried out, using the following equations:

Pin-hole camera model: No distortion is assumed, so the images are required to be previously rectified.

$$x_i \cdot f_x - (x_{fi} - c_x) \cdot z_i = 0, \qquad \text{for all} \quad i = 1..4 \tag{6.1}$$

$$y_i \cdot f_y - (y_{fi} - c_y) \cdot z_i = 0, \qquad \text{for all} \quad i = 1..4 \tag{6.2}$$

Known geometry of the helipad:

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\| = L_{ij}, \qquad \text{for all} \quad i,j = 1..4, \quad i \neq j \tag{6.3}$$

$$(\boldsymbol{x}_i - \boldsymbol{x}_j) \circ (\boldsymbol{x}_j - \boldsymbol{x}_k) = L_{ij} \cdot L_{jk} \cdot \cos \alpha_{ji}^{jk}, \qquad \text{for all} \quad i,j,k = 1..4, \quad i \neq j \neq k \tag{6.4}$$

Where $\boldsymbol{x}_i = [x_i,\ y_i,\ z_i]^T$ are the 3D coordinates of the point i in the camera central coordinate system, $\boldsymbol{x}_{fi} = [x_{fi},\ y_{fi}]^T$ are the 2D coordinates of the point i in the camera lateral coordinate system, L_{ij}, and α_{ji}^{jk}.

The reader must note that the system of equations has two possible solutions (since the helipad is flat). The solution that provides a z-coordinate positive of the corners of the helipad is selected ($z_i > 0$).

Pose of the helipad reconstruction

Once the corners of the helipad have been reconstructed, the pose of the helipad reference frame in camera coordinates is calculated easily.

In the first place, the origin of the helipad reference is calculated according to its arbitrary definition. With this point, the translation part of the pose of the helipad is obtained.

Secondly, the axis of the helipad reference frame are calculated according to its arbitrary definition. With this axis, the orientation part of the pose of the helipad is obtained.

6.2. Evaluation and results

6.2.1. Methodology

The performance of the proposed helipad detection and reconstruction has been evaluated by means of experiments with real images in real environments with challenging conditions.

The presented algorithm has been tested in different environments with diverse light conditions. The helipad has been acquired by the camera from different points of view to test a wide range of all the possible conditions that the presented algorithm has to tackle. Additionally, the helipad has been contaminated with multiple objects of different shape and color, generating very challenging partial occlusions.

The results of some of the experiments are presented bellow. Unfortunately, ground truth data is not available, so the performance of the presented solution can only be evaluated qualitatively.

The proposed solution has never been tested with images acquired by cameras mounted on flying aerial robots, nevertheless, as the detection and reconstruction are based on single images (not a sequence of images), the performance when used by flying robots is expected to remain equivalent.

6.2.2. System setup

Two different cameras have been used in the presented experiments: the first one is a Point Gray Inc, Chameleon USB color camera (model CMLN-13S2C-CS) with a resolution of 640×480 pixels. The other one is the front camera of an AR.Drone Parrot 2.0 with a resolution of 640×360 pixels.

An emulated helipad consisting on a gray surface with a white H surrounded by a white circle is used. Two different size helipads are used, a small one of approximate dimensions of 0.19×0.19 m^2 for preliminary tests, and a real size one of approximate dimensions of 2.5×2.5 m^2 for real tests.

6.2.3. Results

Fig. 6.10 shows an example of how the helipad detection and the 3D reconstruction performs with an emulated helipad.

Fig. 6.11 shows an example of how the helipad detection and the 3D reconstruction performs with an emulated helipad when it is very tilted (around 45°) with respect to the camera frame, but relatively close to it.

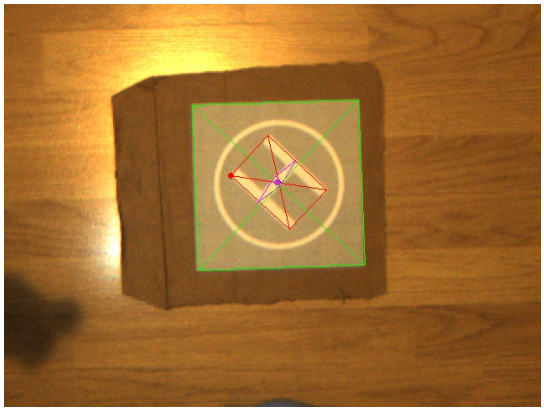
Fig. 6.12 shows another example of how the helipad detection and the 3D reconstruction performs with an emulated helipad when it is very tilted (around 45°) with respect to the camera frame, but relatively close to it.

Table 6.1 shows an analysis of the helipad reconstruction results of the three previous experiments.

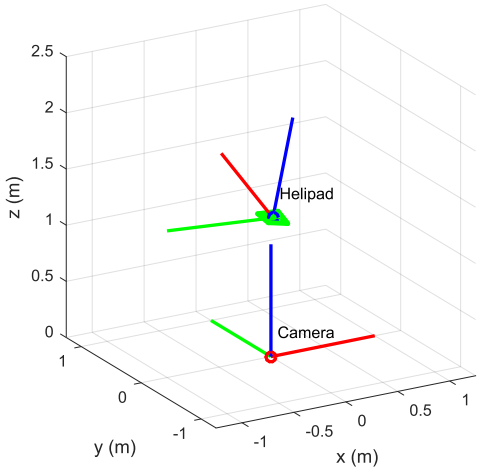
Experiment	x (m)	y (m)	z (m)	Yaw (°)	Pitch (°)	Roll (°)
Fig. 6.10	0.0053	-0.0206	1.2297	86.1399	-19.1853	1.5412
	$(6 \cdot 10^{-7})$	$(1.14 \cdot 10^{-6})$	$(7.6 \cdot 10^{-5})$	(1.2741)	(7.2436)	(17.1859)
Fig. 6.11	-0.0363	0.0002	0.8185	92.0723	5.5006	-44.4490
	$(2.5 \cdot 10^{-7})$	$(1.09 \cdot 10^{-6})$	$(1.655 \cdot 10^{-5})$	(0.5546)	(0.7299)	(0.9623)
Fig. 6.12	-0.0397	-0.0257	0.8139	132.6900	43.1263	2.0786
	$(4.64 \cdot 10^{-6})$	$(3.8 \cdot 10^{-7})$	$(1.624 \cdot 10^{-5})$	(1.8216)	(8.1056)	(12.304)

Table 6.1: Analysis of the performance in the reconstruction of the heliport. The numbers of the table represent the following: mean (variance).

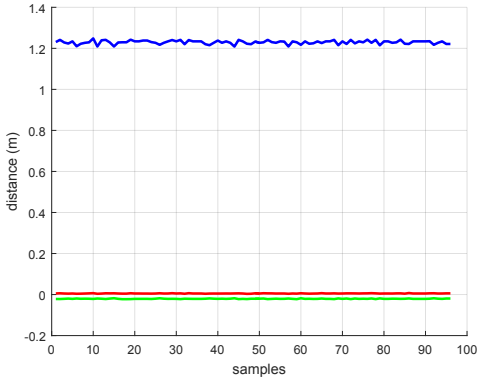
Fig. 6.13 demonstrates that the detection of an emulated helipad has a good performance despite being highly contaminated with multiple elements on its surface that are causing partial occlusions of the helipad.



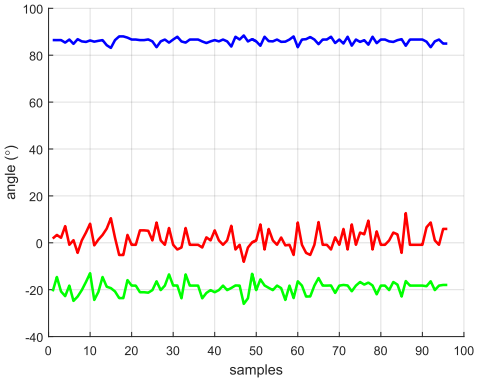
(a) Detection of the helipad.



(b) Reconstruction of detected the helipad.

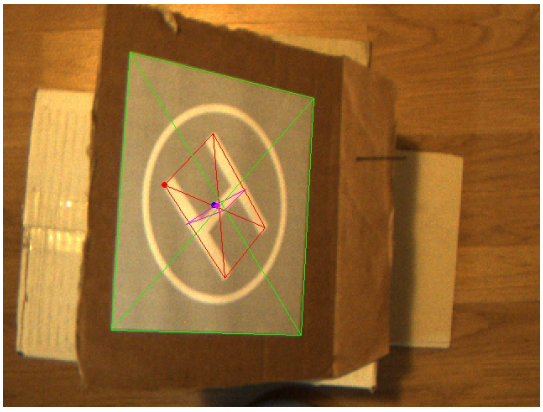


(c) Reconstructed position during a sequence of time. x is red, y is green, z is blue.

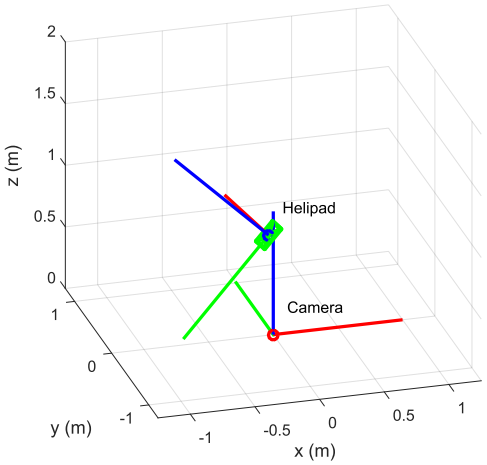


(d) Reconstructed attitude during a sequence of time. Yaw is blue, Pitch is green, Roll is red.

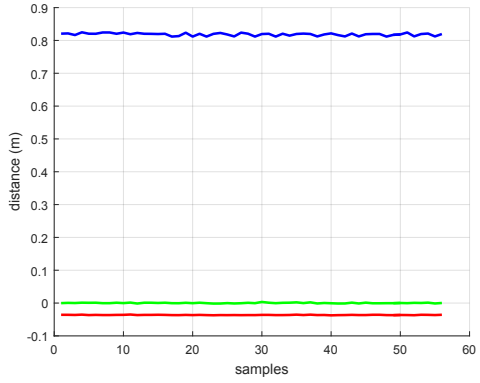
Figure 6.10: Example of the detection of the helipad and the 3D reconstruction of the heliport.



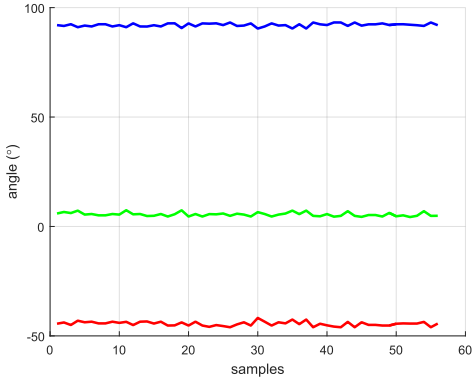
(a) Detection of the helipad.



(b) Reconstruction of detected the helipad.

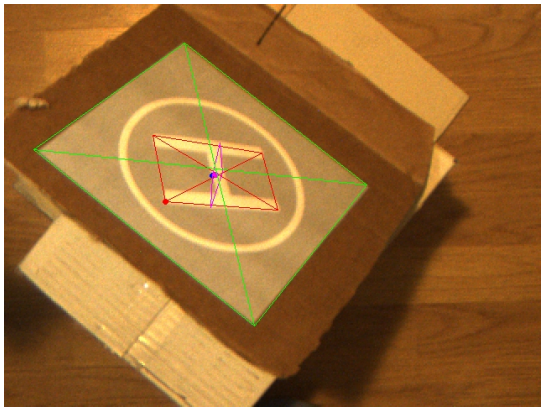


(c) Reconstructed position during a sequence of time. x is red, y is green, z is blue.

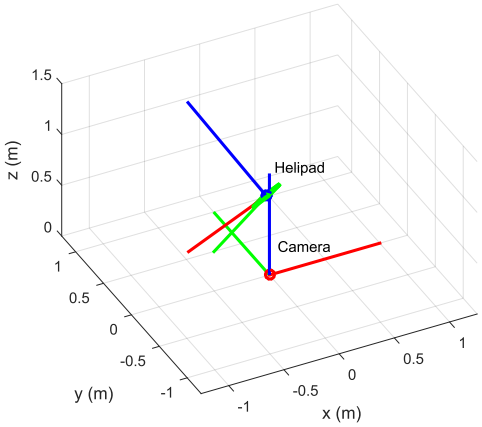


(d) Reconstructed attitude during a sequence of time. Yaw is blue, Pitch is green, Roll is red.

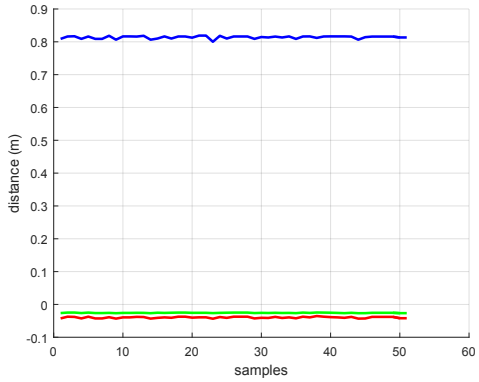
Figure 6.11: Example of the detection of the helipad and the 3D reconstruction of the heliport.



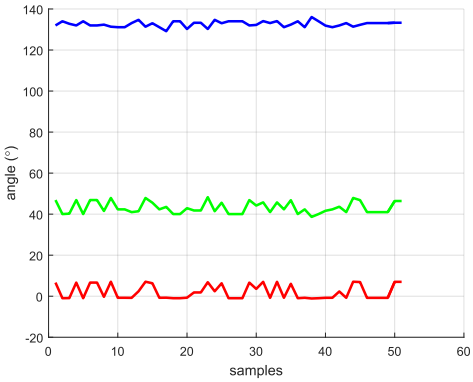
(a) Detection of the helipad.



(b) Reconstruction of detected the helipad.

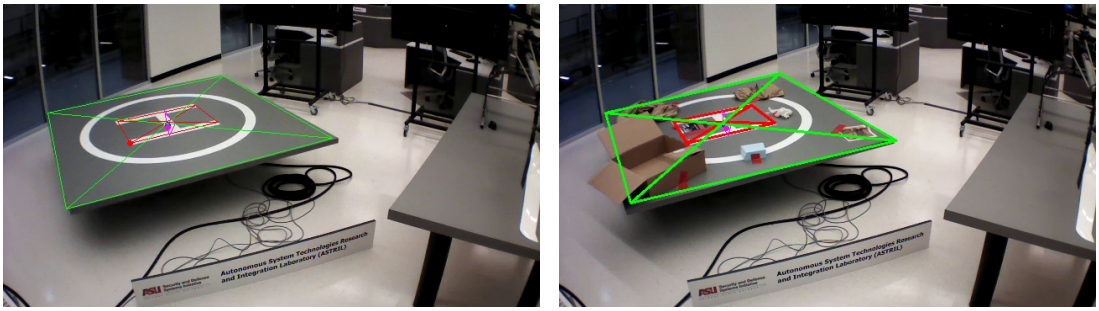


(c) Reconstructed position during a sequence of time. x is red, y is green, z is blue.



(d) Reconstructed attitude during a sequence of time. Yaw is blue, Pitch is green, Roll is red.

Figure 6.12: Example of the detection of the helipad and the 3D reconstruction of the heliport.



(a) Clear helipad. (b) Partially occluded helipad.

Figure 6.13: Example of the detection of the helipad.

6.3. Considerations and further improvements

This section explores some of the limitations of the presented helipad detection and reconstruction and points out how to overcome them.

Helipad detection

The detection of the helipad in the image is done by means of a computer vision algorithm formed by five steps, as described in Section 6.1.1.

This algorithm has some parameters, for example, the values of the thresholding steps, that require being manually adjusted by the user and that depends on the light conditions and the exact colors of the helipad (gray for the background and white for the helipad marks). Having an automatic configuration of these parameters depending on the light conditions, for example by means of adaptive thresholding, is an immediate future work line.

In the last years, new machine learning techniques, known as deep learning techniques, have revolutionized the computer vision field, overcoming the performance of the classic computer vision techniques. Despite the usage of a machine learning algorithm based on an artificial neural network in the classification stage of the proposed algorithm, the usage of novel machine learning techniques for the complete detection problem should be explored.

Helipad reconstruction

As indicated in Section 6.1.2, the pose of the helipad is reconstructed using a custom minimization taking into account the pin-hole camera model and the geometry of the helipad using a Levenberg-Marquardt algorithm. The evaluation of the performance of the proposed minimization compared with the state of the art PnP algorithms is an immediate future work need.

Experiments

The performance of the proposed helipad detection and reconstruction has been evaluated by means of experiments with real images in real environments with challenging conditions, as analyzed in Section 6.2. Nevertheless, the proposed solution has never been tested with images acquired by cameras mounted on flying aerial robots, being a future work that has to be done to confirm the adequate expected performance.

6.4. Summary

In this chapter, a solution capable of detecting an helipad mark on a monocular color image and to recover its pose with respect to the camera sensor, using the knowledge of the intrinsic parameters of the camera (camera calibration) and the shape and dimensions of the helipad mark, has been presented.

The detection of the helipad is done by means of a robust computer vision algorithm with a large decision tree to avoid false positives. The heliport is extracted based on the knowledge that is colored as gray. The helipad marks (the H and the circle) are segmented knowing that they are located on the heliport surface and they are white. After the segmentation of the helipad marks, they are classified using seven different steps that include individual and joint tests. Some of the tests are based on simple geometric conditions, whereas others are more complex such as a classifier based on an artificial neural network. Finally, the corners of the helipad are recovered using the previously calculated helipad marks, and a previously known model of the helipad.

The reconstruction of the helipad consists of two steps. First of all, the position of the four corners of the helipad in camera coordinates are calculated by a minimization taking into account the pin-hole camera model and the geometry of the helipad. Secondly, the pose of the helipad in camera coordinates is obtained.

The performance of the proposed helipad detection and reconstruction has been tested by means of experiments with real images in real environments with challenging conditions, such as light changes and partial occlusions.

An analysis of the main limitations of the proposed perception solution, together with the proposed future work to overcome them has been carried out.

The proposed helipad detection and reconstruction algorithm, presented in this chapter, can be found on (Sanchez-Lopez et al., 2013b; Sanchez-Lopez et al., 2014b).

Chapter 7

Perception based on Odometry and Visual Markers with Environment Reconstruction

This chapter presents a complete perception solution that allows the estimation of the state of the aerial robot using visual markers together with the measurements provided by several sensors. In addition, this solution provides an environment reconstruction formed by geometric primitives.

The proposed solution, represented in Fig. 7.1, estimates the state of the aerial robot thanks to the information provided by several sensors, obtaining in the first place an odometric drifting state estimation, followed afterward by a simultaneous localization and mapping algorithm that relies on visual markers for the drift-free state estimation.

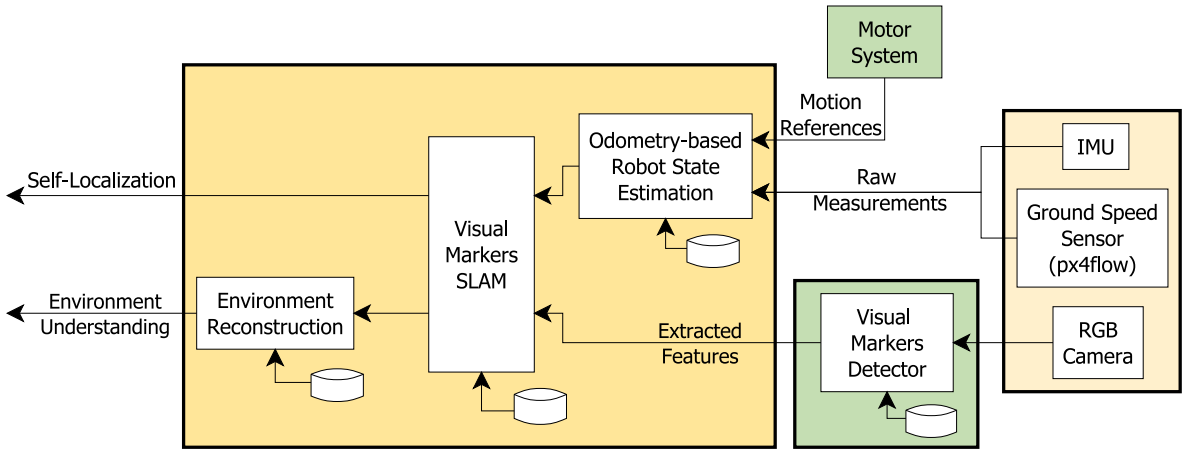


Figure 7.1: Perception solution based on odometry and visual markers with environment reconstruction.

The environment is assumed to be formed by (elliptical) poles and (rectangular) walls. A set of visual markers

are augmenting the environment, labeling its elements with a previously known relationship. Therefore, to be able to reconstruct the environment, the information given by the visual markers is used.

Sensors

The proposed solution imposes a particular sensor setup, where the aerial robot is required to have rigidly attached one of each of the following sensors:

- A device with an IMU (accelerometer and gyro) and a magnetometer that has an internal state estimator that provides a measurement of the attitude, z_ψ; the angular velocity, z_ω; and the acceleration, z_a of the sensor attached to the robot in sensor coordinates. Having available this device is common in aerial robotics, as it is typically included in the autopilot hardware component.
- A distance sensor (e.g. ultrasound sensor) for measuring the ground distance, providing the measurement of the distance to the ground in sensor coordinates, z_d.
- A horizontal linear velocity sensor, as for example the px4flow sensor, (Honegger et al., 2013), providing measurements of the horizontal velocity of the sensor attached to the robot in sensor coordinates, z_{v_x} and z_{v_y}.
- A RGB camera.

The images given by the RGB camera are used by the visual marker detection component (as explained in Section 7.1) included in the Feature Extraction System, providing the measured pose of the visual markers in camera coordinates, z_{VM_i}. In order to extract this pose of the visual markers, the camera calibration parameters, as well as the size of the visual markers are needed to be previously known.

As required in Section 7.2, all the sensors except the camera are assumed to be located in the center of the robot reference frame, that is, $p^R_{S\star} = 0$.

Motion references

To have an accurate prediction of the robot process model, improving the accuracy on the state estimation, the following motion references commanded to the Motor System, are used as input commands on the state estimation algorithm:

- Horizontal attitude commands: $\left(\psi_{P^W_R}\right)_{ref}$, and $\left(\psi_{R^W_R}\right)_{ref}$.
- Vertical angular velocity command: $\left(\omega_{z^R_{R|W}}\right)_{ref}$.
- Vertical linear velocity command: $\left(v_{z^W_{R|W}}\right)_{ref}$.

Outputs

The outputs of this solution are both a drift-free estimation of the state of the aerial robot, as well as the estimated state of the environment in a usable format formed by higher level geometric features in 2D (ellipses and rectangles).

Components

The proposed perception solution is formed by four components[1]: (1) a visual markers detection (Section 7.1), (2) an odometry based state estimation (Section 7.2), (3) a visual markers based SLAM (Section 7.3), and (4) an environment reconstruction (Section 7.4).

The visual markers detection component extracts uniquely labeled fiducial visual markers from an RGB image, calculating their pose in camera coordinates.

The division between the odometry based state estimation and the visual markers based SLAM permits on having two independent specialized components for state estimation. The odometry based robot state estimation predicts the state of the aerial robot thanks to a model of the multirotor aerial robot, updating it with the measurements given by the odometric sensors. The visual markers simultaneous localization and mapping uses the previous odometric estimation of the pose of the aerial robot and the visual markers to estimate a drift-free state of the aerial robot, together with the pose of the perceived landmarks in the world.

The environment reconstruction converts the map of landmarks estimated by the visual markers simultaneous localization and mapping component, into a usable format, generating higher level geometric features in 2D (ellipses and rectangles). This configuration with several independent components allows decoupling the two main functionalities of the perception, the robot state estimation, and the environment reconstruction.

[1] The components presented here have been developed jointly with other researchers.

Reference frames

All the reference frames involved on the presented perception solution might be arbitrarily defined, although is is recommended to follow the standards in robotics (for example in ROS[2]). The preferred definition of the involved reference frames is summed up in Fig. 7.2.

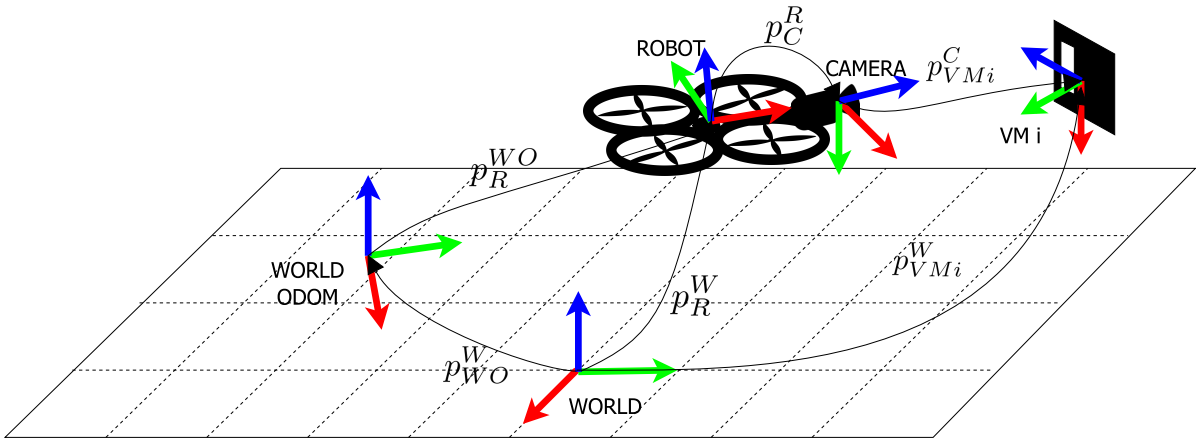


Figure 7.2: Reference frames involved on the presented perception solution.

The world reference frame is arbitrarily defined being the z-axis vertical and its center located on the ground.

The robot reference frame is rigidly attached to the robot, defined as a Front-Left-Up reference frame.

The camera reference frame is rigidly attached to the camera of the robot, being the z-axis perpendicular to the image plane and in the direction of the environment. The y-axis is in the direction of the increasing vertical pixels of the image, and the x-axis in the direction of the increasing horizontal pixels of the image. The transformation p^R_C is known by calibration.

The visual markers reference frame is defined, being the z-axis normal to the visual markers plane and the x-axis downwards.

The world odometry reference frame is initially coincident to the world reference frame, $p^W_{WO} = 0$. Nevertheless, as explained in Section 7.2, the drift of the odometric state estimator is accumulated in the transformation p^W_{WO}, and therefore it moves with respect to the world reference frame.

The remainder of the chapter is organized as follows: Sections 7.1 to 7.4, describe the components of the proposed perception solution. In Section 7.5, the performance of the proposed solution is evaluated with real experiments. Section 7.6 points out some lines of future work based on the limitations of the proposed solution. Section 7.7 sums up the contributions of the chapter.

7.1. Fiducial visual marker detection

This component is able to extract uniquely labeled fiducial visual markers from an RGB image. Additionally, if the intrinsic camera parameters (camera calibration parameters) are known, together with the dimensions of the visual markers, their 3D pose with respect to the camera frame, p^W_{VMi}, might be calculated.

There exist multiple fiducial visual markers, but the preferred ones are the ArUco visual markers, (Garrido-Jurado et al., 2014; Garrido-Jurado et al., 2016). These visual markers are widely used, and they have an open source software library[3] that is currently maintained and updated. Fig. 7.3 shows an example of these visual markers, how they are detected and how their 3D pose is reconstructed.

[2]Online: `http://www.ros.org/reps/rep-0105.html`
[3]Online: `http://www.uco.es/investiga/grupos/ava/node/26`

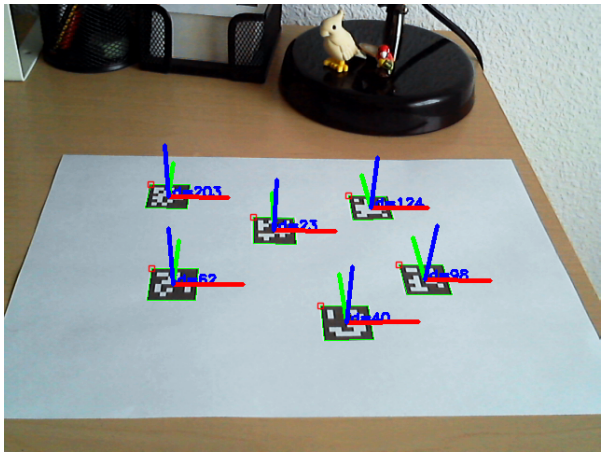


Figure 7.3: Detection and reconstruction of some sample ArUco visual markers. Image from `www.opencv.org`.

7.2. Odometry based robot state estimation

This component estimates the state of the aerial robot thanks to the information provided by the following sensors: (1) an IMU device, (2) a distance sensor, and (3) a horizontal linear velocity sensor.

Additionally, to have an accurate prediction of the state of the aerial robot, it takes as motion inputs the following motion references: (1) the horizontal attitude commands, (2) the vertical angular velocity command, and (3) the vertical linear velocity command.

7.2.1. Extended Kalman filter algorithm

This component is based on an extended Kalman filter algorithm, described in Appendix D.2, whose state, input commands, measurements, and models are described bellow.

It has two specific requirements that need to be satisfied: (1) provide a synchronous estimate of the robot state, and (2) accept asynchronous measurements. To achieve these requirements, two independent loops are used. In the first loop, the estimated state is predicted synchronously at a user-specified rate, using the process model. In the other loop, whenever a measurement arrives at the state estimator, the estimated state is predicted up to the current time and afterward updated with the available measurements.

Synchronous estimate of the robot state

This requirement is imposed by the components that use the estimate of the robot state, such as the Motor System, that, to adequately generate the control commands, it needs a synchronous estimate of the robot state to use it as the feedback in the feedback control loop.

To integrate the continuous-time process model between two instants of time, k and $k+1$, a Runge-Kutta method is used. The input commands employed in this integration are the last received ones and they are used with a zero-order hold, assuming that this command is constant until a new input command is received.

More particularly, the integration period, Δt between two instants of time, k and $k+1$, is divided into several intermediate steps to ensure that the dynamics of the system is appropriately represented. In every intermediate step, the integration is carried out by means of the simple Euler method.

Asynchronous measurements

This requirement is due to the sensors are not assumed to be synchronized, and therefore their measurements might arrive at the state estimator asynchronously and with a different frequency.

This requirement can be solved, if the measurements are not correlated, with a variable measurement model that takes into account only the part of the model that represents the incoming measurements, as well as its associated matrix of variances and covariances. The hypothesis of having uncorrelated measurements is acceptable since the sensors are totally independent.

An adaptation of the extended Kalman filter update stage equations is done to be able to use the complete measurement model together with a vector of active measurement flags, $\boldsymbol{e}$. This modification can be done by

means of two different approaches: the first one, explained in (Sanchez-Lopez, 2012), preserves internally the dimension of the measurement model, but drastically increases the computational cost. The second one reduces the internal dimension of the measurement model.

The vector of active measurement flags, $\boldsymbol{e}$ is given by:

$$\boldsymbol{e} = \begin{bmatrix} \boldsymbol{e}_1 \\ \cdots \\ \boldsymbol{e}_i \\ \cdots \\ \boldsymbol{e}_{n_m} \end{bmatrix}_{n_{m_T} \times 1} \tag{7.1}$$

being $\boldsymbol{e}_i$ the vector of active measurement flags of the i-th measurement with dimensions $n_{m_i} \times 1$, n_{m_i} the dimension of the i-th measurement, $n_{m_T} = \sum_{\forall i \in n_m} n_{m_i}$, and n_m the number of measurements. Additionally, $\boldsymbol{e}_i = \mathbf{1}$ if the measurement has been received and needs to be updated (it is enabled or active for the update stage), and $\boldsymbol{e}_i = \boldsymbol{o}$ otherwise. The number of the active or enabled measurements is n_{m_a}, being the total dimension of the active measurements given by $n_{m_{a_T}} = \sum_{\forall i \in n_{m_a}} n_{m_i}$.

The active measurement matrix $\boldsymbol{E}$ is built using the vector $\boldsymbol{e}$ in two different ways, depending the approach used: In the first approach, where the dimension of the measurement model requires to be internally preserved, the matrix $\boldsymbol{E}$ is formed as $\boldsymbol{E} = diag(\boldsymbol{e})$, being a square matrix of dimension $n_{m_T} \times n_{m_T}$ that has all elements equal to zeros except its diagonal that is equal to $\boldsymbol{e}$. In the second approach, where the dimension of the measurement model can be internally reduced, the matrix $\boldsymbol{E}$ is a block-matrix of dimension $n_{m_a} \times n_m$, where the block $\boldsymbol{E}(i,j) = diag(\boldsymbol{e}_i)$ is the i-th active measurement is the j-th total measurement and $\boldsymbol{E}(i,j) = \mathbf{0}$ otherwise.

The measurement model needs to be updated as follows:

$$\boldsymbol{z'}(k) = \boldsymbol{E} \cdot \boldsymbol{z}(k) = \boldsymbol{E} \cdot \boldsymbol{h}(\boldsymbol{x}(k), \boldsymbol{\mu}, \boldsymbol{n}) \tag{7.2}$$

so therefore, the Jacobian matrices of the measurement model are:

$$\boldsymbol{H}'_x = \boldsymbol{E} \cdot \boldsymbol{H}_x = \boldsymbol{E} \cdot \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}} \tag{7.3}$$

$$\boldsymbol{H}'_\mu = \boldsymbol{E} \cdot \boldsymbol{H}_\mu = \boldsymbol{E} \cdot \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{\mu}} \tag{7.4}$$

$$\boldsymbol{H}'_n = \boldsymbol{E} \cdot \boldsymbol{H}_n = \boldsymbol{E} \cdot \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{n}} \tag{7.5}$$

The vector of measurements is updated as:

$$\tilde{\boldsymbol{z}}'(k) = \boldsymbol{E} \cdot \tilde{\boldsymbol{z}}(k) \tag{7.6}$$

In the first approach, that preserves internally the dimension of the measurement model, the equation to calculate the Kalman filter gain $\boldsymbol{K}$, needs to be modified because the covariance matrix of the innovation $\boldsymbol{S}(k)$ is not invertible anymore, being:

$$\boldsymbol{K} = \boldsymbol{P}(k+1|k) \cdot (\boldsymbol{H}'_x)^T \cdot \boldsymbol{S}(k)^+ \tag{7.7}$$

where $\boldsymbol{S}(k)^+$ is the pseudo-inverse of the matrix $\boldsymbol{S}(k)$.

All the other extended Kalman filter update equations remain equivalent.

It is important to note that the first approach that internally preserves the dimension of the measurement model is more computationally expensive than the other method, not only because it works with bigger size matrices, but also because it requires calculating a pseudo-inverse of a matrix instead of its inverse. It is therefore preferred the second approach that internally reduces the dimension of the measurement model.

7.2.2. State

The full estimated state, $\boldsymbol{x}$, is formed by the following elements:

- full pose of the robot in world coordinates, p_R^W, using angles to describe the attitude: $\boldsymbol{t}_R^W = [t_{x_R}^W, t_{y_R}^W, t_{z_R}^W]^T$ and $\boldsymbol{\psi}_R^W = [\psi_{Y_R}^W, \psi_{P_R}^W, \psi_{R_R}^W]^T$.
- linear velocity of the robot in world coordinates with respect to the world: $\boldsymbol{v}_{R|W}^W = [v_{x_{R|W}}^W, v_{y_{R|W}}^W, v_{z_{R|W}}^W]^T$.
- vertical angular velocity of the robot with respect to the world in world coordinates: $w_{z_{R|W}}^W$.
- some auxiliar variables to adequately represent the dynamics of the horizontal attitude control and the vertical velocity controller.

7.2.3. Input commands

The input commands used in this state estimator are the motion references described before:

- Horizontal attitude commands: $\left(\psi_{P}^{W}{}_{R}\right)_{ref}$, and $\left(\psi_{R}^{W}{}_{R}\right)_{ref}$.
- Vertical angular velocity command: $\left(\omega_{z}^{R}{}_{R|W}\right)_{ref}$.
- Vertical linear velocity command: $\left(v_{z}^{W}{}_{R|W}\right)_{ref}$.

7.2.4. Process model

The process model, that represents the dynamics of the state of the multirotor aerial robot, uses the input commands listed above and it is graphically described in Fig. 7.4.

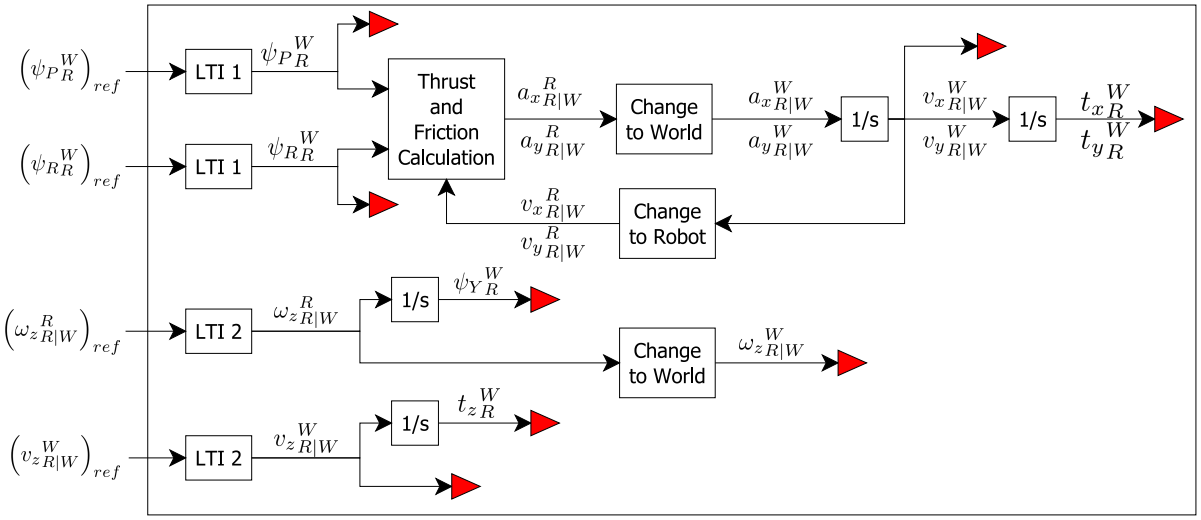


Figure 7.4: Graphical representation of the process model used to describe the dynamics of the aerial robot.

Some elements of the dynamics of the aerial robot are simplified by means of first and second order linear time invariant (LTI) systems ($LTI1 = \frac{A}{s+B}$; $LTI2 = \frac{A}{s^2+B \cdot s+C}$).

The multirotor aerial robot is assumed to move slowly, which is translated in the following assumptions:

$$\psi_{P}^{W}{}_{R} \approx 0 \tag{7.8}$$

$$\psi_{R}^{W}{}_{R} \approx 0 \tag{7.9}$$

$$f_{z}^{R}{}_{R} \approx \frac{g}{m_{robot}} \tag{7.10}$$

These hypothesis imply:

$$\omega_{x}^{R}{}_{R|W} \approx \omega_{x}^{W}{}_{R|W} \approx 0 \tag{7.11}$$

$$\omega_{y}^{R}{}_{R|W} \approx \omega_{y}^{W}{}_{R|W} \approx 0 \tag{7.12}$$

$$\omega_{z}^{R}{}_{R|W} \approx \omega_{z}^{W}{}_{R|W} \tag{7.13}$$

Thanks to these hypotheses, the horizontal acceleration of the robot can be easily calculated using the knowledge of some aerodynamics coefficients. Once the horizontal acceleration is calculated, the linear velocity, as well as the 2D position of the robot can be obtained by integration.

Additionally, these hypotheses allow to directly integrate the vertical angular velocity of the robot, $\omega_{z}^{W}{}_{R|W}$, to obtain the yaw angle, $\psi_{Y}^{W}{}_{R}$.

A further description of the robot process model can be found in (Pestana, 2012), together with the offline procedure for the estimation of the LTI systems coefficients and the aerodynamics coefficients. Providing additional details of this state estimator is out of the scope of this thesis.

7.2.5. Measurements

The measurements used by this state estimator are:
- Attitude, $\boldsymbol{z}_{\psi}$, given by the internal state estimator of the IMU device.
- Vertical angular velocity, z_{ω_z}, given by the internal state estimator of the IMU device.
- Ground distance, z_d, given by the distance sensor.
- Horizontal linear velocity, z_{v_x} and z_{v_y}, given by the horizontal linear velocity sensor.

7.2.6. Measurement model

The measurement model, that associates the measurements with the state of the robot, is simplified thanks to the assumption that all the sensors are located in the center of the robot and aligned with the robot reference frame; and taking into account the previously defined slow movement assumption. Additionally, the ground is assumed to be flat and static. The complete measurement model is formed by the all the individual measurement model corresponding to every measurement.

Measurement model corresponding to the attitude measurement given by IMU device internal state estimator:

$$\boldsymbol{z}_{\psi} = \boldsymbol{\psi}_R^W + \boldsymbol{n}_{z_{\psi}} \tag{7.14}$$

Measurement model corresponding to the vertical angular velocity measurement given by IMU device internal state estimator:

$$\begin{aligned} z_{\omega_z} =& \boldsymbol{\omega}_{z_{R|W}}^R + n_{z_{\omega_z}} \\ \approx& \boldsymbol{\omega}_{z_{R|W}}^W + n_{z_{\omega_z}} \end{aligned} \tag{7.15}$$

Measurement model corresponding to the ground distance measurement given by the ground distance sensor:

$$z_d \approx -t_{z_R}^W + n_{z_d} \tag{7.16}$$

Measurement model corresponding to the horizontal linear velocity measurement given by the horizontal linear velocity sensor:

$$\begin{aligned} z_{v_x} =& v_{x_{R|W}}^R + n_{z_{v_x}} \\ \approx& \cos(\psi_{Y_R}^W) \cdot v_{x_{R|W}}^W + \sin(\psi_{Y_R}^W) \cdot v_{y_{R|W}}^W + n_{z_{v_x}} \end{aligned} \tag{7.17}$$

$$\begin{aligned} z_{v_y} =& v_{y_{R|W}}^R + n_{z_{v_y}} \\ \approx& -\sin(\psi_{Y_R}^W) \cdot v_{x_{R|W}}^W + \cos(\psi_{Y_R}^W) \cdot v_{y_{R|W}}^W + n_{z_{v_y}} \end{aligned} \tag{7.18}$$

The Jacobian matrices of the measurement model are very straight-forward and are omitted for clarity.

7.2.7. Considerations

The usual way to deal with a drifting pose estimation in robotics (for example in ROS[4]) is to accumulate the drift in the transformation p_{WO}^W.

Therefore, the presented odometry based state estimator calculates the state of the robot in odometry world coordinates.

To estimate the pose of the robot in world coordinates, p_R^W, the following transformation needs to be done:

$$p_R^W = p_{WO}^W \oplus p_R^{WO} \tag{7.19}$$

and therefore:

$$\boldsymbol{t}_R^W = \boldsymbol{R}_{WO}^W \cdot \boldsymbol{t}_R^{WO} + \boldsymbol{t}_{WO}^W \tag{7.20}$$

$$\boldsymbol{R}_R^W = \boldsymbol{R}_{WO}^W \cdot \boldsymbol{R}_R^{WO} \tag{7.21}$$

where $\boldsymbol{R}_{\circ}^{\bullet}$ is the rotation matrix associated to the Euler angles $\psi_{\circ}^{\bullet}$.

The presented odometry based state estimator provides an absolute drift-free estimate of the attitude of the robot, $\psi_R^{WO} = \psi_R^W$, and the angular velocity of the robot, $\boldsymbol{\omega}_{R|WO}^{WO} = \boldsymbol{\omega}_{R|W}^W$, thanks to the measurements given by the internal estimator of the IMU device that fuses accelerometer, gyro and magnetometer measurements.

[4]Online: `http://www.ros.org/reps/rep-0105.html`

Additionally, for a flat floor, this state estimator provides a reliable absolute drift-free estimate of the vertical coordinate of the aerial robot, $t_{z_R}^{WO} = t_{z_R}^{W}$, since it relies on the measurements provided by the distance sensor. The vertical velocity of the robot can also be noisily estimated without drifts, $v_{z_{R|WO}}^{WO} = v_{z_{R|W}}^{W}$.

In the same way, the presented state estimator, by means of the measurements given by the horizontal linear velocity sensor, is capable of estimating a drift-free estimation of the horizontal linear velocity, $v_{x_{R|WO}}^{WO} = v_{x_{R|W}}^{W}$ and $v_{y_{R|WO}}^{WO} = v_{y_{R|W}}^{W}$.

Nevertheless, as there is no measurement that directly updates the estimation of the horizontal position of the robot, it will drift over the time with no bounds, $t_{x_R}^{WO} \neq t_{x_R}^{W}$ and $t_{y_R}^{WO} \neq t_{y_R}^{W}$.

7.3. Visual markers based localization and mapping

To estimate a drift-free pose of the robot in world coordinates, visual markers are used, overcoming the limitations of the odometry based robot state estimator presented in Section 7.2. This state estimator uses as input command, the drifting pose of the robot calculated by the odometry based robot state estimator.

7.3.1. Extended Kalman filter algorithm

An extended Kalman filter algorithm for simultaneous localization and mapping, described in Appendix D.2, is used in this component, whose state, input commands, measurements, and models are described bellow.

Similarly than the odometry based state estimator presented in Section 7.2, it is able to synchronously estimate the state of the aerial robot, updating it asynchronously whenever the measurements of the visual marker detection arrive.

Some visual markers have a previously known pose in world coordinates. These visual markers can be directly used as landmarks without any extra effort.

Nevertheless, the usual case is that the visual markers have a previously unknown pose in world coordinates. To be able to take advantage of them in the robot state estimation, as well as to estimate their pose in world coordinates, a mapping stage needs to be done.

Unlike it is presented in Appendix D.2, the mapping stage is carried out following a maximum incremental probability approach (also called maximum likelihood) which allows mapping the new landmarks in a computationally efficient way, without increasing the dimension of the covariance of the state. Therefore, after the mapping, the state of the landmarks is never modified again and thus only the covariance of the robot state is needed to be taken into account in the prediction, update, and mapping stages.

In the following sections, the Jacobian matrices of the process model, measurement model required by this component are omitted for clarity.

7.3.2. State

The estimated state is formed by the robot state and the landmarks state:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_R \\ \boldsymbol{x}_{VM1} \\ \dots \\ \boldsymbol{x}_{VMi} \\ \dots \end{bmatrix} \tag{7.22}$$

The robot state, $\boldsymbol{x}_R$, includes the full pose of the robot in world coordinates, p_R^W, using angles to describe the attitude: $\boldsymbol{t}_R^W = \left[t_{x_R}^W, t_{y_R}^W, t_{z_R}^W\right]^T$ and $\boldsymbol{\psi}_R^W = \left[\psi_{Y_R}^W, \psi_{P_R}^W, \psi_{R_R}^W\right]^T$.

Every visual marker state, $\boldsymbol{x}_{VMi}$, includes the full pose of the visual marker in world coordinates, p_{VMi}^W, using angles to describe the attitude: $\boldsymbol{t}_{VMi}^W = \left[t_{x_{VMi}}^W, t_{y_{VMi}}^W, t_{z_{VMi}}^W\right]^T$ and $\boldsymbol{\psi}_{VMi}^W = \left[\psi_{Y_{VMi}}^W, \psi_{P_{VMi}}^W, \psi_{R_{VMi}}^W\right]^T$.

7.3.3. Input command

This component only uses as input command, the estimated pose of the robot in odometry world coordinates, p_R^{WO}, given by the odometry based state estimator presented in Section 7.2.

As this estimated pose has a drift in the horizontal position, to have a prediction of the movement of the robot, it is used the increment of this estimated pose. Therefore, the input command taken by this component is given

by:

$$\Delta \boldsymbol{u}(k) = p_{R(k-1)}^{R(k)} = \ominus p_R^{WO}(k-1) \oplus p_{WO(k)}^{WO(k-1)} \oplus p_R^{WO}(k) \tag{7.23}$$

And assuming a small drift between two instants of time, $p_{WO(k)}^{WO(k-1)} \approx 0$, then:

$$\Delta \boldsymbol{u}(k) \approx \ominus p_R^{WO}(k-1) \oplus p_R^{WO}(k) \tag{7.24}$$

It is worth to highlight that an important simplification is done in terms of noise of the input command used in the process model, being the noisy input command given by:

$$\Delta \tilde{\boldsymbol{u}}(k) \approx \Delta \boldsymbol{u}(k) + \boldsymbol{n}_u \tag{7.25}$$

7.3.4. Process model

The process model has two different parts, one related to the robot, and the other related to the landmarks.

Robot process model

The robot state is modified following:

$$p_R^W(k) = p_R^W(k-1) \oplus p_{R(k-1)}^{R(k)} \tag{7.26}$$

And therefore, the robot process model is given by:

$$\boldsymbol{x}_R(k) = \boldsymbol{x}_R(k-1) \oplus \Delta \tilde{\boldsymbol{u}}(k-1) \tag{7.27}$$

Landmarks process model

The landmarks are modeled as static, and their pose is not changing over the time:

$$p_{VMi}^W(k) = p_{VMi}^W(k-1) \tag{7.28}$$

And therefore, the landmarks process model is given by:

$$\boldsymbol{x}_{VMi}(k) = \boldsymbol{x}_{VMi}(k-1) \tag{7.29}$$

Nevertheless, as stated before, once mapped, the state of the landmarks is never modified and therefore this process model is never used.

7.3.5. Measurements

It uses as measurements, the perceived poses of the visual markers in camera coordinates, p_{VMi}^C, being the noisy measurement given by:

$$\boldsymbol{z}_{VMi} = p_{VMi}^C + \boldsymbol{n}_{z_{VM}} \tag{7.30}$$

7.3.6. Measurement model

The measurement model that relates the state with the measurements is given by:

$$\begin{aligned} \boldsymbol{z}_{VMi} &= p_{VMi}^C + \boldsymbol{n}_{z_{VM}} \\ &= \ominus p_C^R \ominus p_R^W \oplus p_{VMi}^W + \boldsymbol{n}_{z_{VM}} \\ &= \ominus p_C^R \ominus \boldsymbol{x}_R \oplus \boldsymbol{x}_{VMi} + \boldsymbol{n}_{z_{VM}} \end{aligned} \tag{7.31}$$

being p_C^R, a calibration parameter.

It is worth to note that, since the visual markers have unique ids, data association is not needed.

7.3.7. Mapping model

The mapping model, that allows to calculate the state of a new visual marker using the unmatched measurements and the previously known state of the robot is given by:

$$p_{VMi}^W = p_R^W \oplus p_C^R \oplus p_{VMi}^R \tag{7.32}$$

being, p_C^R, a calibration parameter.

7.4. Environment reconstruction

This last component of the presented perception solution has the objective to convert the map of landmarks estimated by the visual markers simultaneous localization and mapping component described in Section 7.3, into a usable format.

This component generates higher level geometric features in 2D (ellipses and rectangles) associating the estimated pose of the landmarks to the estimated pose and dimensions of these 2D objects.

A previous knowledge of the association between the landmark ids with the kind of object that are labeling and its dimension is required.

7.4.1. Ellipses

An ellipse in 2D is defined by the 2D position of its center in world coordinates $\boldsymbol{t}_C^W = [x_c,\ y_c]^T$, its two dimensional radius $\boldsymbol{r} = [r_x,\ r_y]^T$, and its rotation angle ψ_C^W in world coordinates. Its implicit equation allows to calculate the position of every point of its border in coordinates of the frame attached to its center, $\boldsymbol{t}_P^C = [x_e,\ y_e]^T$:

$$\left(\frac{x_e}{r_x}\right)^2 + \left(\frac{y_e}{r_y}\right)^2 = 1 \tag{7.33}$$

A 2D transformation has to be done to obtain its implicit equation in world coordinates, $\boldsymbol{t}_P^W = [x,\ y]^T$ instead of in its center reference frame coordinates:

$$\begin{aligned} \boldsymbol{t}_P^W &= \boldsymbol{R}_C^W(\psi)\cdot\boldsymbol{t}_P^C + \boldsymbol{t}_C^W \\ \Rightarrow \boldsymbol{t}_P^C &= \left(\boldsymbol{R}_C^W(\psi)\right)^T\cdot(\boldsymbol{t}_P^W - \boldsymbol{t}_C^W) \end{aligned} \tag{7.34}$$

being $\boldsymbol{R}_C^W$ the 2D rotation matrix associated to ψ_C^W:

$$\boldsymbol{R}_C^W(\psi_C^W = \psi) = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix} \tag{7.35}$$

Therefore, as the visual markers are located on the surface of the objects, the pose of the center of the object in world coordinates, $\boldsymbol{t}_C^W$, and ψ_C^W, can be calculated using the previously described implicit equation.

In the case that more than one visual marker has been mapped, the pose of the center of the object in world coordinates is calculated by averaging the estimated pose of the center of the object in world coordinates by every visual marker.

7.4.2. Rectangles

Similarly to the ellipses, a rectangle in 2D is defined by the 2D pose of its center in world coordinates $\boldsymbol{t}_C^W = [x_c,\ y_c]^T$, its two dimensional size $\boldsymbol{s} = [2\cdot s_x,\ 2\cdot s_y]^T$, and its rotation angle ψ_C^W in world coordinates. Its implicit equation on Lamé format allows to calculate the position of every point of its border in coordinates of the frame attached to its center, $\boldsymbol{t}_P^C = [x_r,\ y_r]^T$:

$$\left|\frac{x_r}{s_x} + \frac{y_r}{s_y}\right| + \left|\frac{x_r}{s_x} - \frac{y_r}{s_y}\right| = 2 \tag{7.36}$$

The same 2D transformation that has been done for the ellipses, has to be done for the rectangles to obtain its implicit equation in world coordinates, $\boldsymbol{t}_P^W = [x,y]^T$.

The pose of the center of the object in world coordinates, $\boldsymbol{t}_C^W$, and ψ_C^W, is estimated following a similar procedure than the indicated for the ellipses.

7.5. Evaluation and results

7.5.1. Methodology

The proposed perception solution, incorporated as several open-source components in Aerostack, has been intensively used in multiple research projects, including the IMAV 2013 international competition (see Section 12.2.2); in several experiments (see Section 12.3.1 and Section 12.3.4), and multiple public demonstrations (see Section 12.4).

Bellow, an experiment with a real flight is presented. In this experiment, the performance of the presented perception solution is evaluated within Aerostack performing a mission similar than the indoors challenge of the IMAV 2013 competition, presented in Section 12.2.2.

In this experiment, several aerial robots are performing the same mission sequentially. All the aerial robots take-off from the team base. After the take-off, they cross through a window, entering a zone with several poles as static obstacles. Four poles, with previously known positions, create a square, around which the aerial robots have to perform loops. After several loops, the aerial robots have to cross through the obstacles zone which has four more poles located in unknown positions. Finally, they are commanded to land on a static heliport in a previously known position.

Unfortunately, ground truth data is not available, so the performance of the presented solution can only be evaluated qualitatively.

7.5.2. System setup

The same system setup than in the experiments presented in Section 12.2.2 is used.

Three Parrot AR.Drone 2.0 are used as the aerial platform, individually connected to three average laptops using a Wi-Fi interface. The three laptops are connected to a LAN by means of a switch. Every laptop is running all the components of Aerostack required for every robotic agent to perform the fully autonomous mission, including the presented path planner.

7.5.3. Results

Fig. 7.5 shows, for *drone1* only, plots (ordered by time) of the drift-free (green) and the odometry-based (blue) estimates of the same experiment represented in Fig. 12.8.

The known environment elements are shown in black and the progressively mapped unknown elements are shown in red. This figure shows how the visual markers based SLAM component is able to calculate a drift-free pose from the odometry based estimate. Unfortunately, ground truth is not available.

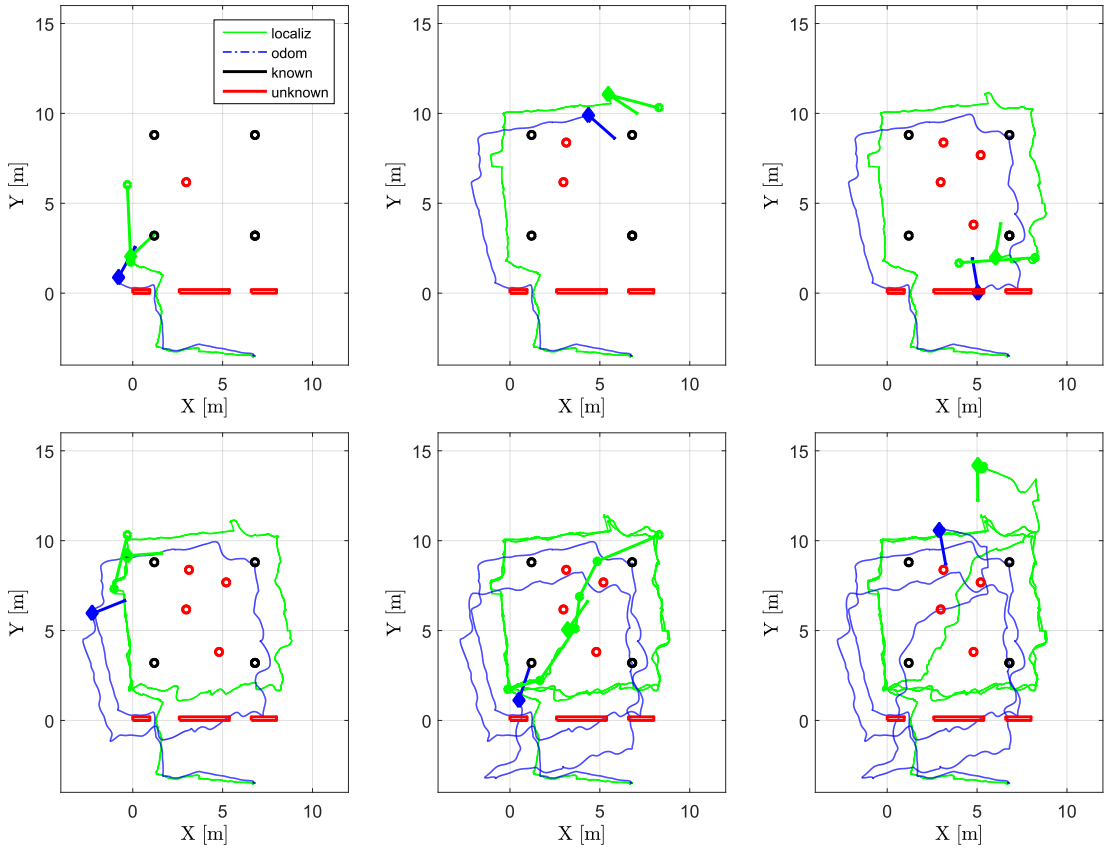


Figure 7.5: Different instant times of the same experimental flight as shown in Fig. 12.8, focusing on the state estimators performance of *drone1*.

7.6. Considerations and further improvements

This section explores some of the limitations of the presented perception solution and points out how to overcome them. Most of these limitations are overcome by the perception solution presented in Chapter 9.

Limited kinds of sensors

The presented solution allows combining the information coming from four different kinds of sensors, an IMU device, a distance sensor, a horizontal linear velocity sensor, and an RGB camera used for the detection of visual markers. Despite being enough to have a drift free estimation of the state of the aerial robot, it has a tight requirement that is the fact of augmenting the environment with visual markers. Without these visual markers, the proposed solution is unable to calculate a drift-free estimate of the state of the aerial robot.

To overcome this limitation, the information provided by other sensors or state estimation algorithms, such as a Motion Capture System (MoCap), a 3D LIDAR SLAM, a Stereo Visual SLAM, a Global Navigation Satellite System (GNSS) or a monocular visual SLAM, should be used in the state estimation process.

Simplified robot process model

In Section 7.2, a simplified robot process model, under the assumption that the multirotor aerial robot moves slowly, has been presented. This model allows having an accurate prediction of the state of the aerial robot in absence of measurements. Nevertheless, this process model does not represent properly the dynamics of the aerial robot whenever it is moving relatively fast, resulting in an inaccurate prediction. Additionally, the LTI parts of this model require being estimated offline for every multirotor aerial robot, lacking versatility.

A more versatile and accurate robot process model for the prediction stage should be defined.

Simplified sensors models

Another important limitation of the proposed perception solution is the highly simplified measurement models presented Section 7.2. The sensors, except the camera, are assumed to be located and aligned with the robot reference frame, what, depending on the particular sensor setup might be unrealistic. Additionally, the models of the sensors do not include any internal state such as biases, what is a very common practice to increase the accuracy of the state estimation.

To allow more realistic sensor setups, and to improve the accuracy of the state estimation, more complex measurement models should be defined.

Mapping algorithm of landmarks

As stated in Section 7.3, the mapping of a new landmark is carried out following a maximum incremental probability approach. On the one hand, this allows mapping the new landmarks in a computationally efficient way, without increasing the dimension of the covariance of the state. Nevertheless, on the other hand, if the estimation of the robot state is inaccurate (i.e. high covariance) when the landmark is mapped, the state of the landmark is never updated even if the covariance in the estimation robot state decreases.

Other mapping techniques should be analyzed to improve the accuracy of the state estimation, both of the robot and landmarks.

7.7. Summary

In this chapter, a perception solution that allows the estimation of the state of the aerial robot using visual markers together with the measurements provided by several sensors has been presented.

The proposed perception solution is formed by four different components: (1) a visual markers detection, (2) an odometry based state estimation, (3) a visual markers based SLAM, and (4) an environment reconstruction.

The visual markers detection extracts uniquely labeled fiducial visual markers from an RGB image. Additionally, their pose in camera coordinates is calculated if the intrinsic camera parameters are known together with the dimensions of the visual markers.

The odometry based state estimation provides a drifting estimate of the state of the aerial robot thanks to the information coming from the following sensors: (1) an IMU device, (2) a distance sensor, and (3) a horizontal linear velocity sensor. It uses a simplified model of the multirotor aerial robot with input commands, under the assumption that the robot is moving slowly, for the prediction of the robot state. Additionally, the models used to describe the measurements provided by the sensor are as well highly simplified, assuming to be located and aligned with the robot reference frame.

The visual markers based SLAM estimates a drift-free pose of the robot in world coordinates using visual markers. This state estimator uses as input command, the drifting pose of the robot calculated by the odometry based robot state estimator. This component is able to map previously unknown detected visual markers following a maximum incremental probability approach which allows mapping the new landmarks in a computationally efficient way, without increasing the dimension of the covariance of the state.

The environment reconstruction converts the map of landmarks estimated by the visual markers simultaneous localization and mapping component, into a usable format, generating higher level geometric features in 2D (ellipses and rectangles).

The proposed solution is able to synchronously estimate the state of the aerial robot, updating it asynchronously whenever a measurement arrives.

The performance of the proposed perception solution has been widely demonstrated, despite all the simplifications, thanks to its usage in multiple research projects with real flights, including an international aerial robotics competition, several different experiments, and various public demonstrations. Additionally, in this chapter, its performance has been evaluated with a real flight experiment where three fully autonomous aerial robots were executing a mission similar than the indoors challenge of the IMAV 2013 competition, examining the behavior of one of the agents.

An analysis of the main limitations of the proposed perception solution, together with the proposed future work to overcome them has been carried out.

The proposed perception solution is completely available to the scientific community as an open-source software integrated into Aerostack.

The proposed perception solution that allows the estimation of the state of the aerial robot using visual markers together with the measurements provided by several sensors, presented in this chapter, can be found on (Sanchez-Lopez et al., 2013a; Pestana et al., 2014e; Sanchez-Lopez et al., 2014a; Sanchez-Lopez et al., 2016b; Pestana et al., 2016).

Chapter 8

Perception based on Odometry and Computer Vision for Gridded Maps

This chapter presents a complete perception solution that allows the estimation of the state of the aerial robot for an environment with a grid, as in the mission 7 of the IARC Competition (see Section 12.2.3 for more information about the competition).

The proposed solution, represented in Fig. 8.1, has been designed to solve this complex problem based on the measurements given by computer vision algorithms, together with the measurements provided by several sensors.

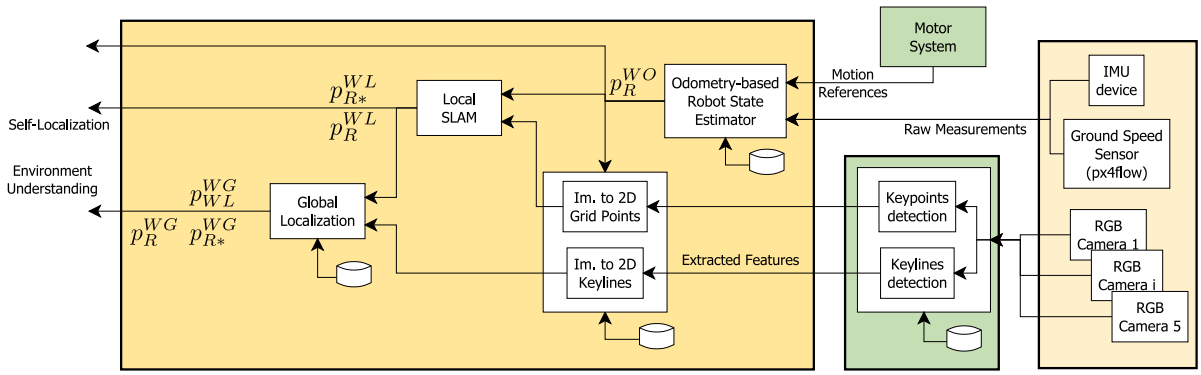


Figure 8.1: Perception solution based on odometry and computer vision for gridded maps.

Arena

As indicated, the presented solution is intended to solve the perception for the mission 7 of the IARC Competition (see Section 12.2.3).

The arena of this competition (represented in Fig. 12.10) is a large (20×20 m^2) flat indoors area that contains some visual clues: a white wide line marks two sides of the arena, while a red and a green wide line mark the

other sides. Additionally, a 1×1 m^2 white grid gives the only guaranteed visual features inside the arena.

Sensors

The proposed solution imposes a particular sensor setup, where the aerial robot is required to have rigidly attached the following sensors:

- A device with an IMU (accelerometer and gyro) and a magnetometer that has an internal state estimator that provides a measurement of the attitude, $\boldsymbol{z}_{\psi}$; the angular velocity, $\boldsymbol{z}_{\omega}$; and the acceleration, $\boldsymbol{z}_{a}$ of the sensor attached to the robot in sensor coordinates. Having available this device is common in aerial robotics, as it is typically included in the autopilot hardware component.
- A distance sensor (e.g. ultrasound sensor) for measuring the ground distance, providing the measurement of the distance to the ground in sensor coordinates, z_d.
- A horizontal linear velocity sensor, as for example the px4flow sensor, (Honegger et al., 2013), providing measurements of the horizontal velocity of the sensor attached to the robot in sensor coordinates, z_{v_x} and z_{v_y}.
- Several (five) RGB cameras.

The images given by the RGB cameras are used by the visual marker detection component (as explained in Section 8.1) included in the Feature Extraction System, providing the points where the vertical and horizontal lines of the grid intersect and the lines that label the borders of the arena.

As required in Section 8.2, all the sensors except the cameras are assumed to be located in the center of the robot reference frame, that is, $p^R_{S\star} = 0$.

Motion references

Similarly as proposed in Chapter 7, to have an accurate prediction of the robot process model, improving the accuracy on the state estimation, the following motion references commanded to the Motor System, are used as input commands on the state estimation algorithm:

- Horizontal attitude commands: $\left(\psi_{P^W_R}\right)_{ref}$, and $\left(\psi_{R^W_R}\right)_{ref}$.
- Vertical angular velocity command: $\left(\omega_{z^R_{R|W}}\right)_{ref}$.
- Vertical linear velocity command: $\left(v_{z^W_{R|W}}\right)_{ref}$.

Outputs

The outputs of this solution are both a drift-free estimation of the state of the aerial robot in two different frames of coordinates (described bellow), a local reference frame, which center is at the take-off point, and a global reference frame, which center is at a predetermined point with respect to the arena.

Components

The proposed perception solution is formed by five components: (1) a grid intersection points and arena border lines detection[1] (Section 8.1), (2) an odometry based state estimation (Section 8.2), (3) an image to 2D feature conversion (Section 8.3), (4) a 2D grid based local SLAM (Section 8.4), and (5) a 2D arena global localization (Section 8.5).

The grid intersection points and arena border lines detection use the images provided by the RGB cameras to extract the points where the vertical and horizontal lines of the grid intersect, the lines that delimit the borders of the arena.

The odometry based state estimation predicts the state of the aerial robot thanks to a model of the multirotor aerial robot, updating it with the measurements given by the odometric sensors.

The image to 2D feature conversion transforms the extracted features provided by the grid intersection points and arena border lines detection from image-based features to spatial features in robot coordinates.

The local SLAM component performs a SLAM, based on the grid intersection points of the arena, providing a drift-free estimation of the state of the aerial robot in coordinates of a reference frame located at the take-off point.

To calculate the state of the aerial robot in coordinates of a reference frame fixed to the arena, the global localization component is used. This component estimates the transformation between the local world reference frame and the global world reference frame.

The advantage of having both a local and a global world reference frames, is that the first one allows to have a drift-free estimate of the state of the aerial robot from the beginning of the mission, even if the location of the

[1] This component has been developed jointly with other researchers.

local world reference frame in the arena (global world reference frame) is unknown. To calculate the relationship between the local and the global world reference frame, an exploration stage is required to be performed in the mission. The state estimated in local world coordinates might be used to control the aerial robot, while the state estimation in global world coordinates might be used for planning.

Reference frames

All the reference frames involved on the presented perception solution might be arbitrarily defined, although is is recommended to follow the standards in robotics (for example in ROS[2]). The preferred definition of the involved reference frames is summed up in Fig. 8.2.

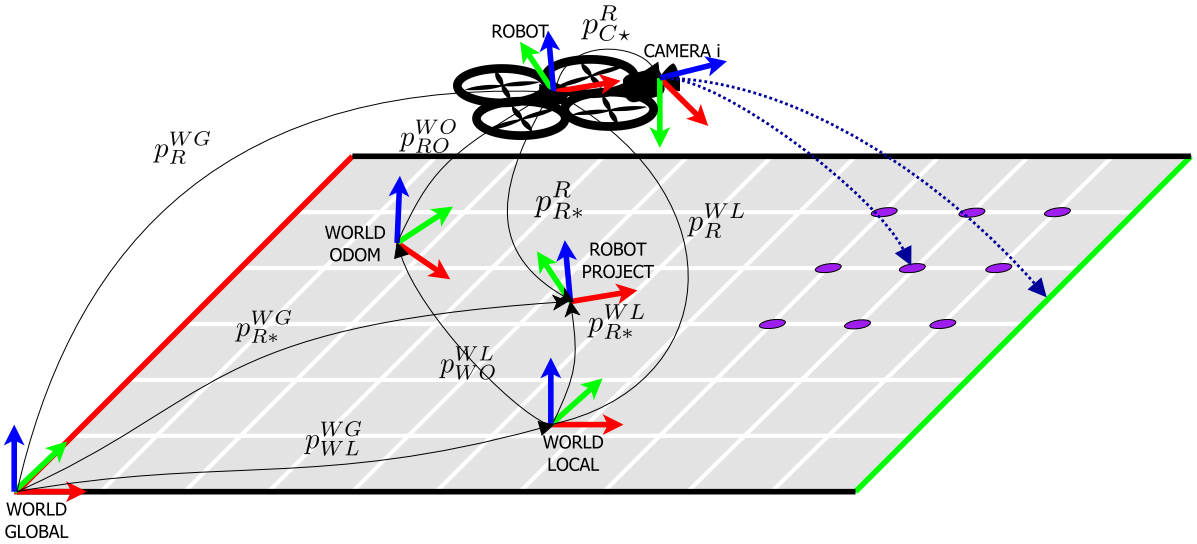


Figure 8.2: Reference frames involved on the presented perception solution.

A global world reference frame is arbitrarily defined in a specific known point of the ground of the arena, being the z-axis vertical and pointing upwards.

A local world reference frame is located on the ground of the arena, but unlike the global world reference frame, located in an arbitrary horizontal position that is not previously known with respect to the arena, for example, the take-off point.

A robot reference frame rigidly attached to the aerial robot defined as a Front-Left-Up reference frame.

Another reference frame is defined to represent the aerial robot. The robot projection reference frame (designed as $R*$) has its center in the projection of the robot reference frame on the ground, and it always remains horizontal, that is, its x-y plane is parallel to the ground. The transformation p^{R*}_R combines a rotation given by the drift-free attitude of the robot in world coordinates, $\psi_{R^W_R}$ and $\psi_{P^W_R}$, and a translation given by the altitude of the robot in world coordinates, $t^W_{z_R}$.

The i-th camera reference frame is rigidly attached to the i-th camera of the robot, being the z-axis perpendicular to the image plane and in the direction of the environment. The y-axis is in the direction of the increasing vertical pixels of the image, and the x-axis in the direction of the increasing horizontal pixels of the image. The transformation $p^R_{C\star}$ is known by calibration.

The world odometry reference frame is initially coincident to the world reference frame, $p^{WL}_{WO} = 0$. Nevertheless, as explained in Section 8.2, the drift of the odometric state estimator is accumulated in the transformation p^{WL}_{WO}, and therefore it moves with respect to the world local reference frame.

The remainder of the chapter is organized as follows: Sections 8.1 to 8.5, describe the components of the proposed perception solution. In Section 8.6, the performance of the proposed solution is evaluated with real experiments. Section 8.7 points out some lines of future work based on the limitations of the proposed solution. Section 8.8 sums up the contributions of the chapter.

[2]Online: `http://www.ros.org/reps/rep-0105.html`

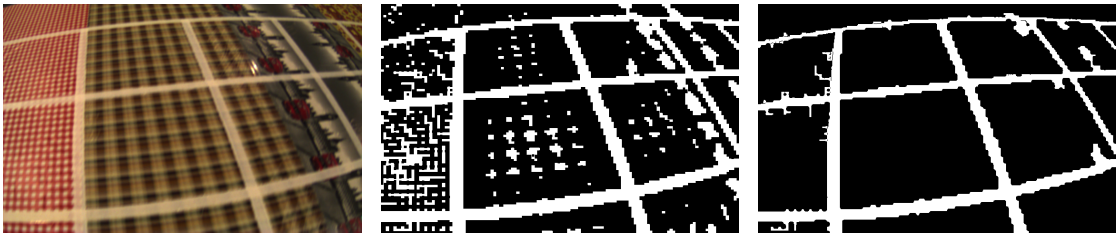
8.1. Grid intersection points and arena border lines detection

This component uses the images provided by the RGB cameras to extract the points where the vertical and horizontal lines of the grid intersect, hereinafter called keypoints. Additionally, it has the responsibility of extracting the lines that delimit the borders of the arena, called keylines.

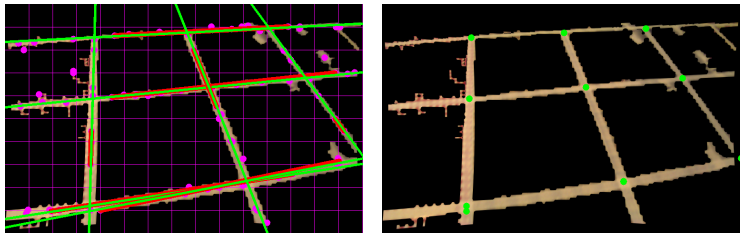
In order to extract these features, the following algorithm, illustrated in Fig. 8.3, is proposed:

1. Segmentation of the acquired image (Fig. 8.3a) using an adaptive threshold (Fig. 8.3b). As the luminance is the most salient characteristic of the lines, the Y channel of the YCrCb colorspace is used.
2. Refinement of the segmentation by flood-filling the grid (Fig. 8.3c). This enables the removal of unwanted high-luminance blobs.
3. Two-pass RANSAC algorithm to compute the grid lines (Fig. 8.3d):
   - Computation of a distance map of the obtained mask. The ridges represent the lines of the grid, and the peaks their intersections. Because of perspective, the farthest lines appear smaller than the closest ones. Hence fixing a threshold cannot extract the peaks; instead, we propose sampling local maxima of subregions of the distance map and fitting lines to these maxima.
   - Division of the distance map into subregions and sample one local maxima by subregion.
   - Usage of a RANSAC (Fischler and Bolles, 1981) algorithm to identify possible lines in the image. Each local maxima can vote for multiple lines.
   - Sorting of the possible line models obtained by vote count in order to identify the most important ones.
   - Second usage of a RANSAC algorithm, using the line models by descending vote order. This time, each local maxima can only vote for a single line, ensuring uniqueness of the lines.

   In Fig. 8.3d, magenta lines and discs respectively show the division of the image in subregions and the local maxima extracted from each subregion. Green lines are the result of the two-pass RANSAC, and red lines link the two generating points of each green line, which are used for identifying keylines.
4. Computation of the grid intersection points knowing the equations of the retained lines (Fig. 8.3e).
5. Computation of the mean color of the neighborhood of each couple of local maxima having generated one of the retained lines in order to identify the keylines.



(a) Acquired image. (b) Threshold mask. (c) Floodfilled grid.



(d) 2-pass RANSAC. (e) Grid intersection points.

Figure 8.3: Illustration of the grid intersection points extraction.

It is important to mention that the acquired image requires being rectified (removing the distortion) before using it in the present component, ensuring that the straight lines of the grid, remain straight in the image.

Further details of this component are out of the scope of this thesis, and the reader is referred to (Pestana et al., 2014d; Sanchez-Lopez et al., 2015).

8.2. Odometry based robot state estimation

This component estimates the state of the aerial robot thanks to the information provided by the following sensors: (1) an IMU device, (2) a distance sensor, and (3) a horizontal linear velocity sensor.

Additionally, to have an accurate prediction of the state of the aerial robot, it takes as motion inputs the following motion references: (1) the horizontal attitude commands, (2) the vertical angular velocity command, and (3) the vertical linear velocity command.

The reader is referred to Section 7.2, where this component has been deeply described. As stated, this odometry based state estimation provides a drift-free estimate of the state of the aerial robot except in the estimation of the horizontal position of the robot, that will drift over the time with no bounds. The drift is accumulated in the transformation p_{WO}^{WL}.

8.3. Image to 2D feature conversion

This component uses the information given by: (1) the grid intersection points and arena borders detection (Section 8.1); (2) the odometric pose of the robot given by the odometry based robot state estimation (Section 8.2); and (3) the previously-known information relative to the calibration of the camera (including its intrinsic parameters and its pose in the aerial robot, $p_{C\star}^{R}$).

With this information, and assuming that the detected features belong to the ground, converts them from image-based features to 3D spatial features in robot coordinates (not camera coordinates). Additionally, it calculates the 2D features of the detected features in robot projection coordinates.

The algorithm of this component is the following:

1. Using the calibrated extrinsic camera parameters, $p_{C\star}^{R}$, and the intrinsic camera parameters (the focal distances f_x and f_y; the center of the image point c_x and c_y; and the image size s_x and s_y), the normalized 3D coordinates of the extracted feature in robot coordinates are calculated. Basically, what is calculated in this step is, for every extracted feature, a line in robot coordinates that represent the possible 3D coordinates of the center of the frame of the extracted feature in aerial robot coordinates.
2. Using the extrinsic camera parameters, $p_{C\star}^{R}$, and the odometric pose of the robot, p_{R}^{WO}, the parameters of the ground plane (normal vector, n_G, and a point, x_G) in robot coordinates are calculated.
3. Assuming that the detected features are in the ground plane, the 3D coordinates of every extracted feature in aerial robot coordinates, $p_{F\star}^{R}$, are calculated by computing the intersection between the ground plane in robot coordinates, and the line of the normalized 3D coordinates of the extracted feature in robot coordinates.
4. A change of the reference frame from the robot reference frame to the robot projection reference frame is done, obtaining the 3D (and 2D) coordinates of extracted feature in 2D aerial robot coordinates, $p_{F\star}^{R*}$.

Note that the fact that the used odometric pose of the robot given by the odometry based robot state estimation, p_{R}^{WO}, drifts in the horizontal coordinates, ${t_x}_{R}^{WO}$ and ${t_y}_{R}^{WO}$, is not an inconvenience in this algorithm, since the only needed components of this estimated pose are the vertical coordinate, ${t_z}_{R}^{WO}$, and the attitude, ψ_{R}^{WO}, which are drift-free estimates.

Grid intersection points

The grid intersection points (keypoints) do not have orientation and are described only by its position, $\boldsymbol{t}_{L\star}^{R*}$.

Arena borders

The arena borders (keylines) are described by a vector, $\boldsymbol{u}_{KL\star}^{R*}$, and a point, ${\boldsymbol{x}_0}_{KL\star}^{R*}$.

8.4. Local SLAM: 2D grid SLAM

This component estimates the drift-free horizontal 2D pose of the robot projection in world local coordinates, $\boldsymbol{t}_{x_{R*}}^{WL}$ and $\boldsymbol{t}_{y_{R*}}^{WL}$.

To estimate this drift-free 2D pose of the robot, the intersection points of the grid lines painted on the ground of the arena (keypoints) are used.

8.4.1. Extended Kalman filter algorithm

An extended Kalman filter algorithm for simultaneous localization and mapping algorithm, described in Appendix D.2, is used in this component, whose state, input commands, measurements, and models are described

bellow.

Similarly than the odometry based state estimator, it is able to synchronously estimate the state of the aerial robot, updating it asynchronously whenever the measurements of the visual marker detection arrive.

8.4.2. State

The state is divided into two components, aerial robot, and landmarks. The state of the robot is always part of the state while the state of the landmarks is added while mapped:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_R \\ \boldsymbol{x}_{L1} \\ \dots \\ \boldsymbol{x}_{L\star} \\ \dots \end{bmatrix} \tag{8.1}$$

The state of the robot encodes the pose of the aerial robot in world local coordinates, p_{R*}^{WL}, given by:

$$\boldsymbol{x}_R = \begin{bmatrix} \boldsymbol{t}_{R*}^{WL} \\ \theta_{R*}^{WL} \end{bmatrix} \tag{8.2}$$

The state of the landmarks encodes the 2D position of the grid intersection points in world local coordinates, $\boldsymbol{t}_{L\star}^{WL}$:

$$\boldsymbol{x}_{L\star} = \boldsymbol{t}_{L\star}^{WL} \tag{8.3}$$

It is important to highlight that the landmarks are 2D points without orientation.

8.4.3. Input command

Similarly than in Section 7.3, this component only uses as input command, the estimated pose of the robot in odometry world coordinates, p_R^{WO}, given by the odometry based state estimator presented in Section 8.2.

As this estimated pose has a drift in the horizontal position, to have a prediction of the movement of the robot, it is used the increment of this estimated pose. Therefore, the input command taken by this component is given by:

$$\Delta \boldsymbol{u}_T(k) = p_{R(k-1)}^{R(k)} = \ominus p_R^{WO}(k-1) \oplus p_{WO(k)}^{WO(k-1)} \oplus p_R^{WO}(k) \tag{8.4}$$

And assuming a small drift between two instants of time, $p_{WO(k)}^{WO(k-1)} \approx 0$, then:

$$\Delta \boldsymbol{u}_T(k) \approx \ominus p_R^{WO}(k-1) \oplus p_R^{WO}(k) \tag{8.5}$$

And, as explained before, only the horizontal 2D values are used as input command, $\Delta \boldsymbol{u}(k)$:

$$\Delta \boldsymbol{u} = \begin{bmatrix} \boldsymbol{t}_{\Delta \boldsymbol{u}} \\ \theta_{\Delta \boldsymbol{u}} \end{bmatrix} \tag{8.6}$$

being:

$$\boldsymbol{t}_{\Delta \boldsymbol{u}} = \begin{bmatrix} \boldsymbol{t}_{x_{R*(k)}}^{R*(k-1)} \\ \boldsymbol{t}_{y_{R*(k)}}^{R*(k-1)} \end{bmatrix} \tag{8.7}$$

$$\theta_{\Delta \boldsymbol{u}} = \psi_{Y_{R*(k)}}^{R*(k-1)} \tag{8.8}$$

It is worth to highlight that an important simplification is done in terms of noise of the input command used in the process model, being the noisy input command given by:

$$\Delta \tilde{\boldsymbol{u}}(k) \approx \Delta \boldsymbol{u}(k) + \boldsymbol{n}_u \tag{8.9}$$

8.4.4. Process model

The process model has two different parts, one related to the aerial robot, and the other related to the landmarks:

Robot process model

The robot state is modified following:

$$p_{R*}^{WL}(k) = p_{R*}^{WL}(k-1) \oplus p_{R*(k-1)}^{R*(k)} \tag{8.10}$$

And therefore, the process model of the aerial robot is given by:

$$p_{R*}^{WL}(k) = p_{R*}^{WL}(k-1) \oplus \Delta\tilde{\boldsymbol{u}}(k-1) \tag{8.11}$$

and therefore:

$$\boldsymbol{t}_{R*}^{WL}(k) = \boldsymbol{R}_{R*}^{WL}(k-1) \cdot \boldsymbol{t}_{\Delta\boldsymbol{u}}(k-1) + \boldsymbol{t}_{R*}^{WL}(k-1) \tag{8.12}$$

$$\boldsymbol{R}_{R}^{WL}(k) = \boldsymbol{R}_{R*}^{WL}(k-1) \cdot \boldsymbol{R}_{\Delta\boldsymbol{u}}(k-1) \tag{8.13}$$

$$\theta_{R*}^{WL}(k) = \theta_{R*}^{WL}(k-1) + \theta_{\Delta\boldsymbol{u}}(k-1) \tag{8.14}$$

being the 2D rotation matrix $\boldsymbol{R}_{\bullet}^{\circ}$ associated to the angle $\theta_{\bullet}^{\circ}$, given by:

$$\boldsymbol{R}_{\bullet}^{\circ} = \begin{bmatrix} \cos\theta_{\bullet}^{\circ} & -\sin\theta_{\bullet}^{\circ} \\ \sin\theta_{\bullet}^{\circ} & \cos\theta_{\bullet}^{\circ} \end{bmatrix} \tag{8.15}$$

Note that equation 8.14 requires the angles to be carefully added.

Landmarks process model

The landmarks are modeled as static, and their pose is not changing over the time:

$$p_{L\star}^{WL}(k) = p_{L\star}^{WL}(k-1) \tag{8.16}$$

And therefore, the landmarks process model is given by:

$$\boldsymbol{t}_{L\star}^{WL}(k) = \boldsymbol{t}_{L\star}^{WL}(k-1) \tag{8.17}$$

Jacobian matrices of the process model

The existing Jacobian matrices of the discrete-time process model are indicated in Table 8.1.

k \ $k-1$	$\partial \boldsymbol{t}_{R*}^{WL}$	$\partial \theta_{R*}^{WL}$
$\partial \boldsymbol{t}_{R*}^{WL}$	$\boldsymbol{I}_{2\times2}$	$\left[\frac{\partial \boldsymbol{R}_{R*}^{WL}(k-1)}{\partial \theta_{R*}^{WL}(k-1)}\right]_{2\times2} \cdot \boldsymbol{t}_{\Delta\boldsymbol{u}}$
$\partial \theta_{R*}^{WL}$	$\boldsymbol{0}_{1\times2}$	1

(a) Robot state in k vs. robot state in $k-1$

k \ $k-1$	$\partial \boldsymbol{t}_{L\star}^{WL}$
$\partial \boldsymbol{t}_{L\star}^{WL}$	$\boldsymbol{I}_{2\times2}$

(b) Landmark $\star$ state in k vs. Landmark $\star$ state in $k-1$

k \ $k-1$	$\partial \boldsymbol{t}_{\Delta\boldsymbol{u}}$	$\partial \theta_{\Delta\boldsymbol{u}}$
$\partial \boldsymbol{t}_{R*}^{WL}$	$\boldsymbol{R}_{R*}^{WL}$	$\boldsymbol{0}_{2\times1}$
$\partial \theta_{R*}^{WL}$	$\boldsymbol{0}_{1\times2}$	1

(c) Robot state in k vs. input command in $k-1$

Table 8.1: Jacobian matrices of the discrete-time process model.

Being:

$$\left[\frac{\partial \boldsymbol{R}_{R*}^{WL}(k-1)}{\partial \theta_{R*}^{WL}(k-1)}\right]_{2\times2} = \begin{bmatrix} -\sin\theta_{R*}^{WL} & -\cos\theta_{R*}^{WL} \\ \cos\theta_{R*}^{WL} & -\sin\theta_{R*}^{WL} \end{bmatrix} \tag{8.18}$$

8.4.5. Measurements

The measurements used in this state estimator are the 2D position of the observed grid intersection points in robot projected coordinates, $\boldsymbol{z}_{t\star}$. The observed grid intersection points are given by the image to 2D feature conversion component, described in Section 8.3.

It is important to note that an important simplification has been done when modeling the noise of the measurement, being the noisy measurement given by:

$$\boldsymbol{z}_{t\star} = \boldsymbol{t}_{L\star}^{R*} + \boldsymbol{n}_{z_t} \tag{8.19}$$

The grid intersection points cannot be not uniquely tagged, and therefore, the data association requires to be solved.

8.4.6. Measurement model

The measurement model that relates the state with the measurements is given by:

$$\begin{aligned} \boldsymbol{z}_{t\star} =& \boldsymbol{t}_{L\star}^{R*} + \boldsymbol{n}_{z_t} \\ =& \left(\boldsymbol{R}_{R*}^{WL}\right)^T \cdot \left(\boldsymbol{t}_{L\star}^{WL} - \boldsymbol{t}_{R*}^{WL}\right) + \boldsymbol{n}_{z_t} \end{aligned} \tag{8.20}$$

Jacobian matrices of the measurement model

The existing Jacobian matrices of the measurement model are indicated in Table 8.2

	$\partial \boldsymbol{t}_{R*}^{WL}$	$\partial \theta_{R*}^{WL}$
$\partial \boldsymbol{z}_{t\star}$	$-\left(\boldsymbol{R}_{R*}^{WL}\right)^T$	$\left[\frac{\partial \left(\boldsymbol{R}_{R*}^{WL}\right)^T}{\partial \theta_{R*}^{WL}}\right]_{2\times 2} \cdot \left(\boldsymbol{t}_{L\star}^{WL} - \boldsymbol{t}_{R*}^{WL}\right)$

(a) Value vs. robot state.

	$\partial \boldsymbol{t}_{L\star}^{WL}$
$\partial \boldsymbol{z}_{t\star}$	$\left(\boldsymbol{R}_{R*}^{WL}\right)^T$

(b) Value vs. landmark $\star$ state.

	$\partial \boldsymbol{n}_{z_t}$
$\partial \boldsymbol{z}_{t\star}$	$\boldsymbol{I}_{2\times 2}$

(c) Value vs. noise.

Table 8.2: Jacobian matrices of the measurement model.

Being:

$$\left[\frac{\partial \left(\boldsymbol{R}_{R*}^{WL}\right)^T}{\partial \theta_{R*}^{WL}}\right]_{2\times 2} = \begin{bmatrix} -\sin\theta_{R*}^{WL} & \cos\theta_{R*}^{WL} \\ -\cos\theta_{R*}^{WL} & -\sin\theta_{R*}^{WL} \end{bmatrix} \tag{8.21}$$

8.4.7. Mapping model

The mapping model that allows calculating the state of a new visual marker using the unmatched measurements and the previously known state of the robot is given by:

$$\boldsymbol{t}_{L\star}^{WL} = \boldsymbol{R}_{R*}^{WL} \cdot \boldsymbol{t}_{L\star}^{R*} + \boldsymbol{t}_{R*}^{WL} \tag{8.22}$$

Jacobian matrices of the mapping model

The existing Jacobian matrices of the mapping model are indicated in Table 8.3.

	$\partial \boldsymbol{t}_{R*}^{WL}$	$\partial \theta_{R*}^{WL}$
$\partial \boldsymbol{t}_{L\star}^{WL}$	$\boldsymbol{I}_{2\times 2}$	$\left[\frac{\partial\left(\boldsymbol{R}_{R*}^{WL}\right)}{\partial \theta_{R*}^{WL}}\right]_{2\times 2} \cdot \boldsymbol{t}_{L\star}^{R*}$

(a) Value vs. robot state.

	$\partial \boldsymbol{t}_{L\star}^{R*}$
$\partial \boldsymbol{t}_{L\star}^{WL}$	$\boldsymbol{R}_{R*}^{WL}$

(b) Value vs. measurement.

Table 8.3: Jacobian matrices of the mapping model.

8.5. Global localization: 2D arena localization

This component has the objective of estimating the transformation between the global world reference frame and the local world reference frame, p_{WL}^{WG}. Once this transformation is properly estimated, the full state of the aerial robot and the map elements can be calculated in global world coordinates to be used for planning purposes.

To estimate this transformation, the four arena border lines painted on the ground (keylines) are used. This knowledge is exploited to localize the robot in the arena.

The keylines are described on the ground plane by a 2D vector $\boldsymbol{u}_{KL\star}^{WG} = \left[\boldsymbol{u}_{x_{KL\star}}^{WG},\ \boldsymbol{u}_{y_{KL\star}}^{WG}\right]^T$ and a 2D point $\boldsymbol{x}_{0_{KL\star}}^{WG} = \left[x_{0_{KL\star}}^{WG},\ y_{0_{KL\star}}^{WG}\right]^T$, being its general (or implicit) equation given by:

$$r_{KL\star}^{WG} : a_{KL\star}^{WG} \cdot x_{KL\star}^{WG} + b_{KL\star}^{WG} \cdot y_{KL\star}^{WG} + c_{KL\star}^{WG} = 0 \tag{8.23}$$

where:

$$a_{KL\star}^{WG} = \boldsymbol{u}_{y_{KL\star}}^{WG} \tag{8.24}$$

$$b_{KL\star}^{WG} = -\boldsymbol{u}_{x_{KL\star}}^{WG} \tag{8.25}$$

$$c_{KL\star}^{WG} = \boldsymbol{u}_{x_{KL\star}}^{WG} \cdot y_{0_{KL\star}}^{WG} - \boldsymbol{u}_{y_{KL\star}}^{WG} \cdot x_{0_{KL\star}}^{WG} \tag{8.26}$$

A particle filter (also called sequential Monte Carlo) is used, being its main stages explained in Section 8.5.5.

8.5.1. State

The state encodes the transformation between the world local and the world global, p_{WL}^{WG} reference frames, being:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{t}_{WL}^{WG} \\ \theta_{WL}^{WG} \end{bmatrix} \tag{8.27}$$

8.5.2. Process model

The transformation between the world local and the world global reference frames remains constant.

$$p_{WL}^{WG}(k) = p_{WL}^{WG}(k-1) \tag{8.28}$$

So therefore, the process model is given by:

$$\boldsymbol{t}_{WL}^{WG}(k) = \boldsymbol{t}_{WL}^{WG}(k-1) \tag{8.29}$$

$$\boldsymbol{\theta}_{WL}^{WG}(k) = \boldsymbol{\theta}_{WL}^{WG}(k-1) \tag{8.30}$$

8.5.3. Measurements

The keylines observed by the image to 2D feature conversion component (described in Section 8.3) are represented in 2D robot coordinates by a vector $\boldsymbol{u}_{KL\star}^{R*}$ and a point $\boldsymbol{x}_{0_{KL\star}}^{R*}$. Nevertheless, the point used to define the keylines cannot be guaranteed to be always the same. This fact leads to the employment as keylines measurements of a unique combination of the keylines descriptors. The magnitudes used as keylines measurements are the distance from the 2D robot reference frame to the keyline, $d_{R*}^{KL\star}$, and the dot product (cosine of the angle) between the main axis (x-axis) of the 2D robot reference frame and the normalized vector of the keyline, $\cos\alpha_{R*}^{KL\star}$.

Finally, it is assumed that the grid points and keylines detection is able to uniquely tag the keylines (using its color information), so the data association problem does not apply here.

The noisy measurement $\boldsymbol{z} = [z_d,\ z_\alpha]^T$, is given by:

$$z_d = d_{R*}^{KL\star} + n_{z_d} \tag{8.31}$$

$$z_\alpha = \cos\alpha_{R*}^{KL\star} + n_{z_\alpha} \tag{8.32}$$

8.5.4. Measurement model

As stated before, the measurement is given by:

$$z_d = d_{R*}^{KL\star} + n_{z_d}$$
$$z_\alpha = \cos\alpha_{R*}^{KL\star} + n_{z_\alpha}$$

To obtain the measurement model, the following equations are used:

$$d_{R*}^{KL\star} = \frac{\left|a_{KL\star}^{WG}\cdot x_{R*}^{WG} + b_{KL\star}^{WG}\cdot y_{R*}^{WG} + c_{KL\star}^{WG}\right|}{\sqrt{\left(a_{KL\star}^{WG}\right)^2 + \left(b_{KL\star}^{WG}\right)^2}} \tag{8.33}$$

$$\cos\alpha_{R*}^{KL\star} = \boldsymbol{i}_{R*}^{WG}\circ\boldsymbol{u}_{KL\star}^{WG} = \left(\boldsymbol{i}_{R*}^{WG}\right)^T\cdot\boldsymbol{u}_{KL\star}^{WG} \tag{8.34}$$

where:

$$p_{R*}^{WG} = p_{WL}^{WG}\oplus p_{R*}^{WL} \tag{8.35}$$

being:

$$\boldsymbol{t}_{R*}^{WG} = \boldsymbol{R}_{WL}^{WG}\cdot\boldsymbol{t}_{R*}^{WL} + \boldsymbol{t}_{WL}^{WG} \tag{8.36}$$

$$\boldsymbol{R}_{R*}^{WG} = \boldsymbol{R}_{WL}^{WG}\cdot\boldsymbol{R}_{R*}^{WL} \tag{8.37}$$

8.5.5. Particle filter algorithm

A standard particle filter algorithm is used to estimate the state. As every Bayes filter based state estimator, the particle filter algorithm has the following stages:

- Initialization: executed only at the beginning of the algorithm.
- Prediction: executed periodically.
- Update: executed only when keyline measurements are available.

Initialization

The state of the particles, $\boldsymbol{x}_{[i]}$, is initialized for all the particles set, being n_P the number of particles in the set. The particles are uniformly distributed all over the arena. If a prior knowledge of where the aerial robot takes off if available, it can be incorporated to the initialization of the state of the particles, improving the convergence of the particle filter.

Prediction

This stage is executed periodically, updating the state of the particles according to the process model. In this proposed state estimator, the process model does not change anything, so this stage could be omitted.

Update

This stage is executed every time a measurement, $\tilde{\boldsymbol{z}}^j$, of a keyline j is available. This stage is divided into the following steps: Measurement prediction, importance factor, average value calculation, and resampling.

Update: Measurement prediction

The predicted measurements of the keylines are calculated for every particle in the set, $\hat{\boldsymbol{z}}_{[i]}^j$, using the measurement model.

Update: Importance factor

For every particle i in the set, an importance factor (also called weight, $w_{[i]}$) is calculated using the measurement innovation:

$$\boldsymbol{\nu}_{[i]}^{j} = \hat{\boldsymbol{z}}_{[i]}^{j} - \tilde{\boldsymbol{z}}^{j} \tag{8.38}$$

Where $\bullet^{j} = \left[\bullet^{j,z_d}, \bullet^{j,z_\alpha}\right]^T$.

The importance factor equation is given by:

$$w_{[i]} = \prod_{\forall j} w_{[i]}^{j} = \prod_{\forall j} \left(\prod_{\forall k} w_{[i]}^{j,k} \right) \tag{8.39}$$

being $w_{[i]}^{j}$ the weight associated to the particle i given by the keyline measurement j; $w_{[i]}^{j,k}$ is the weight associated to the particle i given by the keyline measurement j and the k element of the measurement; and $w_{[i]}^{j,k}$ is the Dirac delta of the element k of the measurement j:

$$w_{[i]}^{j,k} = e^{-\frac{1}{2}\cdot\left(\frac{v_{[i]}^{j,k}}{\sigma_{z_k}}\right)^2} \tag{8.40}$$

Update: Average value calculation

The output of this state estimator is the weighted average value of the particles state, calculated by:

$$\hat{\boldsymbol{t}}_{WL}^{WG} = \sum_{\forall i} \left(w_{[i]} \cdot \boldsymbol{t}_{[i]}{}_{WL}^{WG} \right) \tag{8.41}$$

$$\hat{\theta}_{WL}^{WG} = atan2\left(\sum_{\forall i} \sin\left(w_{[i]} \cdot \theta_{[i]}{}_{WL}^{WG} \right), \sum_{\forall i} \cos\left(w_{[i]} \cdot \theta_{[i]}{}_{WL}^{WG} \right) \right) \tag{8.42}$$

for all particles i in the set.

Note that the average of an angle has to be done as a mean of circular quantities.

Update: Resampling

The total weight ($w_T = \sum w_{[i]}$), and the average weight per particle ($w_A = \frac{w_T}{n_P}$) are calculated, being n_P the total number of particles.

If the total weight is greater than a threshold ($w_T > w_{thres}$), the particles weight is normalized ($w_{[i]} = \frac{w_{[i]}}{w_T}$) and a resampling algorithm is executed.

This resampling algorithm generates a new set of particles from the previous set of particles, based on their normalized weight.

After the resampling, or in the case that the total weight is lower or equal than the threshold ($w_T \leq w_{thres}$), the particles weight is recomputed to $w_{[i]} = \frac{1}{n_P}$ for the next iteration.

To avoid loss of diversity, or in the case that the particles do not represent properly the state (the case that the total weight is lower or equal than a threshold), a random noise is added to the particles state. The amplitude of this noise depends on the average weight per particle, w_A, being the noise amplitude bigger if the average weight per particle is small, and vice-verse.

8.6. Evaluation and results

8.6.1. Methodology and considerations

The proposed perception solution, incorporated as several open-source components in Aerostack, has been used in 2014 for the participation in the mission 7 of the IARC Competition (see Section 12.2.3).

Due to different issues, such as hardware limitations, or delays in the delivery of a fully functional grid intersection points and arena border lines detection, the complete perception solution could not be tested with real flights before the competition day.

To evaluate the performance of the proposed solution, partial experiments and simulations were carried out. The performance of the grid intersection points and arena border lines detection (Section 8.1) is analyzed in Section 8.6.2, using real images of an emulated arena, acquired from a camera mounted onboard the aerial robot in a real flight. The performance of the odometry based robot state estimation (Section 8.2) was previously

analyzed in Chapter 7 when introducing this component for the first time. Finally, the performance of the two remaining state estimators, the local SLAM component (Section 8.4) and the global localization component (Section 8.5) is evaluated in Section 8.6.3 by means of simulations.

8.6.2. Grid intersection points and arena border lines detection performance

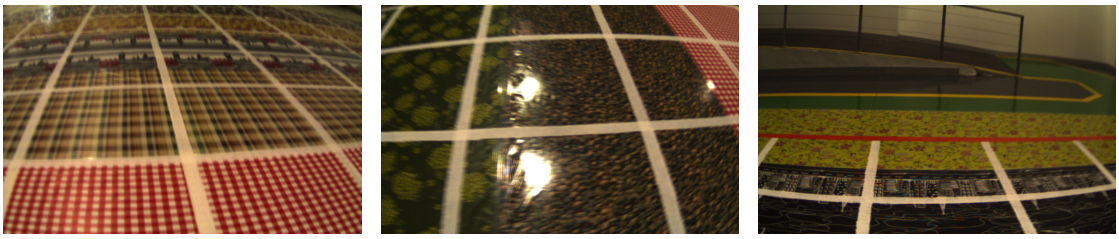
The arena of the competition has been emulated with a very challenging background, with different colors and textures (see Fig. 8.4). Only a reduced arena with a 15×8 m^2 area has been built due to space limitations. The emulated backgrounds were built combining 1.5×8 m^2 strips. Some of the strips were chosen to have a similar color than the grid lines or the arena border lines.

(a) View 1 of the emulated arena.

(b) View 2 of the emulated arena.

Figure 8.4: Emulated arena built for the evaluation of the performance of the grid points and keylines detection.

Fig. 8.5 shows three images acquired from a camera mounted onboard the aerial robot in a real flight. As it can be seen, the grid points and keylines detection has to be able to deal with blurry images, and with light reflections. Additionally, for the proposed camera setup, the grid can be hardly seen in the images whenever it is more than 3 m away from the camera for a flight altitude of 1.5 m, since it is only represented by very few pixels of the images.



(a) Detail of the background.

(b) Light reflections.

(c) Detail of the red keyline.

Figure 8.5: Acquired images of the emulated arena built for the evaluation of the performance of the grid points and keylines detection.

Examples of grid intersection points detections are shown in Fig. 8.6 as purple circled dots. Additional information related to the ground robot detection (not analyzed in this thesis) is also shown in the images.

The analysis of the performance of the grid points and keylines detection reveals that the grid points can only be detected if they are close to the camera (less than 3 m away from the camera for a flight altitude of 1.5 m). In addition, their position in the image is detected with a large noise equivalent to the width of the grid lines. Finally, multiple false grid points are detected, as the bottom middle keypoint detected in Section 8.6.2.

8.6.3. Local SLAM and global localization performance

A simulation using Matlab was carried out to evaluate the performance of the Local SLAM and the global localization. The simulated environment consists on a 20×20 m^2 arena with a 1×1 m^2 grid and the four labeled arena border lines.

The aerial robot was simulated with the following configuration:

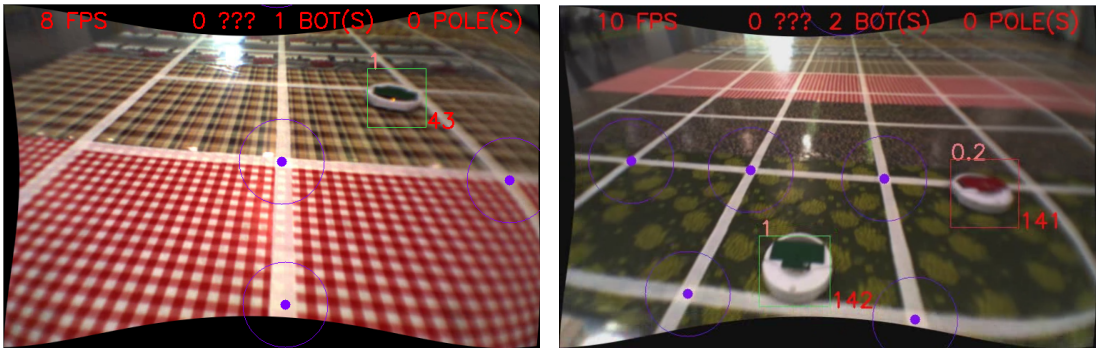


Figure 8.6: Grid intersection points detection results.

- The aerial robot is initially placed in an arbitrary previously unknown point of the environment, more concretely in the point $P_0 = [10,\ 0]$ in global world coordinates, $t_{x_{R*}}^{WG}$ and $t_{y_{R*}}^{WG}$, with an orientation of $\psi_{Y_{R*}}^{WG} = 0$ rad.
- The aerial robot is performs a circular trajectory of a radius of $r = \frac{v}{\omega} = \frac{0.2}{0.0225} = 8.89$ m inside the arena. More than three loops are carried out by the aerial robot, as represented in Fig. 8.7.

Additionally, the outputs of the grid intersection points and arena border lines detection, the odometry based robot state estimation, and the image to 2D feature conversion were emulated with the following parameters, based on their real performance:

- The standard deviation in the estimated state given by the odometry based robot state estimation is fixed to $1/3 \cdot 0.05$ in the three estimated values that are used in the local SLAM component (position and orientation).
- The range in the perception of the keypoints is limited to 3 m, and its standard deviation is set to $1/3 \cdot 0.05$ in the two elements of the measurement, z_t.
- The range in the perception of the keylines is limited to 4 m, and its standard deviation is set to 0.5 in the measurement of the distance, z_d; and 0.05 in the measurement of the cosine of the angle, z_α. The measurements are assumed to be uniquely labeled.

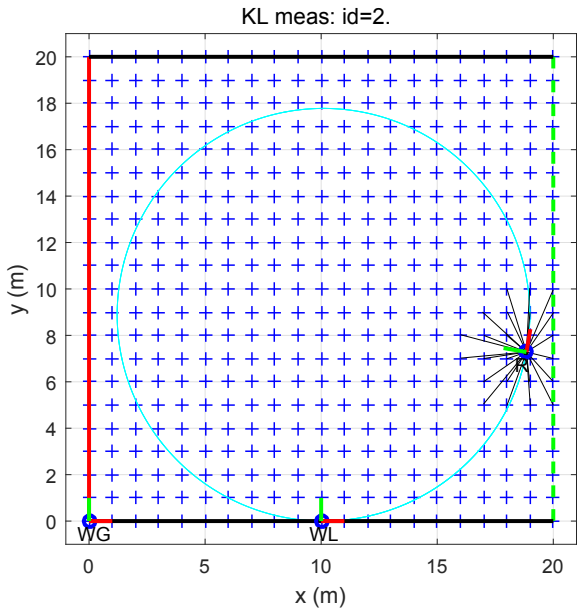


Figure 8.7: Simulation environment for the evaluation of the local SLAM and global localization.

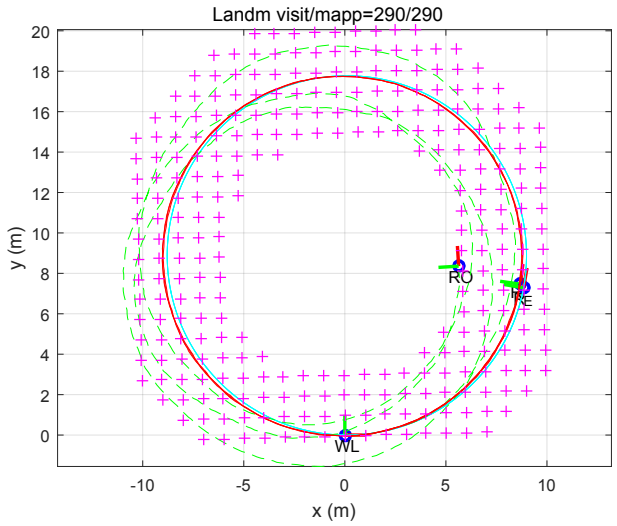
Fig. 8.7 shows the simulation environment for the evaluation of the local SLAM and global localization. The three ground truth reference frames, robot projected, world global and world local are depicted. Blue pluses represent the keypoints, and the black lines symbolize the perceived keypoints. The keylines are represented as

continuous thick lines with their appropriate color. Whenever a keyline is measured, it is represented as a dashed line. The cyan line depicts the ground truth 2D trajectory followed by the aerial robot.

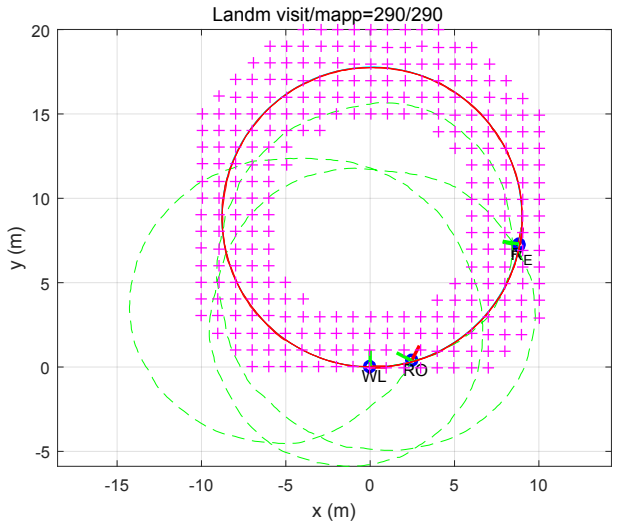
Local SLAM

Fig. 8.8 shows the performance of the global localization in simulation. Two different simulation runs are shown. The dashed green line represents the odometric trajectory (trajectory provided by the odometry based state estimation, assuming $p^{WL}_{WO} = 0$); the red line, the estimated trajectory by the local SLAM; and the cyan line, the ground truth trajectory. The magenta pluses represent the mapped keypoints by the local SLAM.

As can be seen, despite the large drift of the odometry based robot state estimation, the global SLAM component provides an accurate estimation of the position of the robot in world local coordinates, p^{WL}_{R*}, together with an estimated map of the keypoints. In both simulations, 290 different keypoints were measured, mapping all them properly. The first simulation, represented in Fig. 8.8a, has a larger estimation error than the second simulation, in Fig. 8.8b, as the error between the estimated trajectory and the ground truth trajectory is larger. Additionally, the first simulation, the estimated map of keypoints is not perfectly aligned.



(a) Simulation 1.



(b) Simulation 2.

Figure 8.8: Performance of the local SLAM component in simulation.

Fig. 8.9 shows the performance of the local SLAM component in simulation with some extra keypoints randomly placed in the arena. These random keypoints emulate false positives in the detection of the keypoints provided by the grid intersection points and arena border lines detection. In this simulation, 361 different

keypoints were measured, being 392 the number of keypoints mapped. This difference is due to mapping errors caused by wrong data associations. This happened whenever the extra random keypoints were added very close to the grid intersection points, being the noise of the measurement higher than the distance between points. Despite these random keypoints and the mapping errors, the state estimation remains accurate.

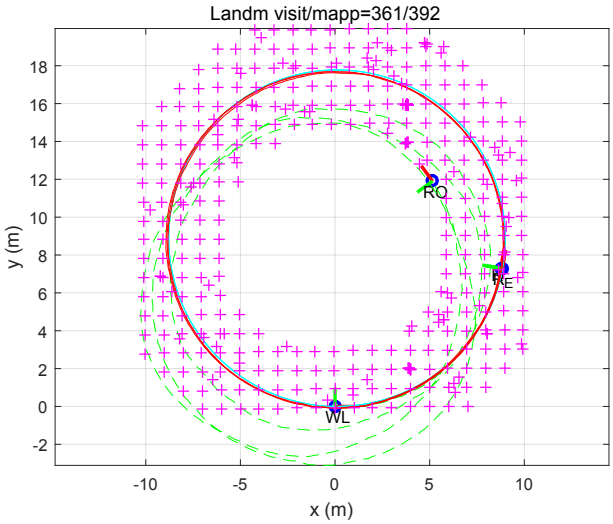
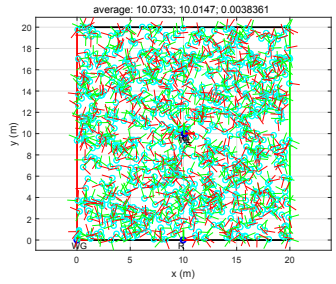


Figure 8.9: Performance of the local SLAM component in simulation with some extra keypoints randomly placed in the arena.

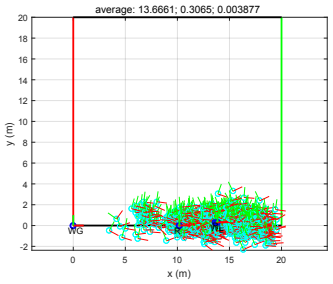
Global localization

Fig. 8.10 shows the evolution of the particles of the global localization in simulation. Initially, at $t = 0$ s, the particles are uniformly distributed in the arena. Once the simulation starts, from $t = 1$ s to $t = 33$ s only the black bottom keyline is measured. Hence, the particles can update their angle and their y position estimation, but they cannot update their x position as it is not observable. From $t = 34$ s to $t = 41$ s, the black bottom and the green keylines are perceived. From this moment, the estimation of the x position of the particles can be updated. As can be seen, from $t = 34$, the estimated pose of the world local in world global coordinates, p_{WL}^{WG}, converged. From $t = 42$ s to $t = 106$ s, only the green keyline is measured. The estimation keeps on updating and refining. From $t = 107$ s to $t = 111$ s, no keyline is measured and therefore the estimation is not updated. After this, from $t = 112$ s, the black top keyline is measured, and the estimation is updated again. The process continues, and the estimation is refined.

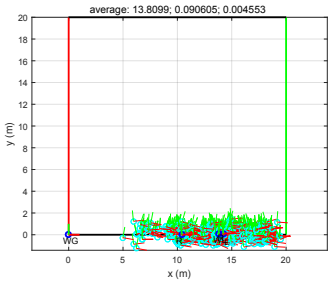
Fig. 8.11 shows the performance of the global localization component in simulation. Two different simulation runs are shown. These two figures correspond to the same simulation runs carried out in Fig. 8.8. As can be seen, at the end of the simulation, the estimation of the pose of the world local in world global coordinates, p_{WL}^{WG}, is very accurate. The error in the estimation of the pose of the robot in world local coordinates, p_{R}^{WL}, estimated by the local SLAM component is accumulated in the estimation provided by the global localization component. This is the reason for which the first simulation is less accurate than the second simulation.



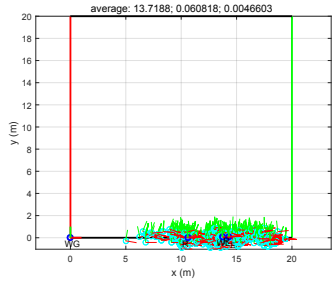
(a) $t = 0$ s. The particles are uniformly distributed in the arena.



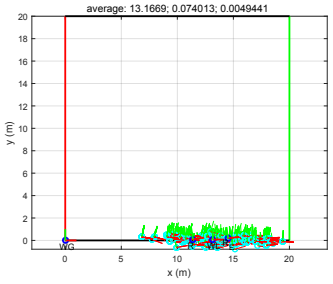
(b) $t = 1$ s. The black bottom keyline is measured.



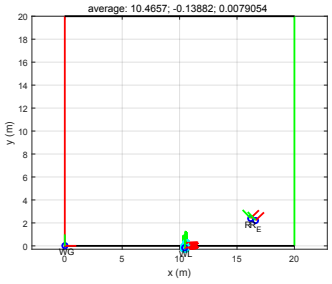
(c) $t = 2$ s. The black bottom keyline is measured.



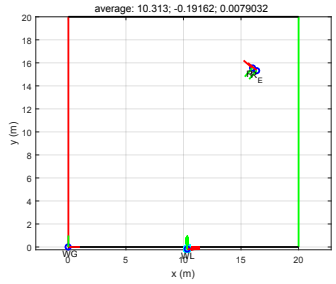
(d) $t = 3$ s. The black bottom keyline is measured.



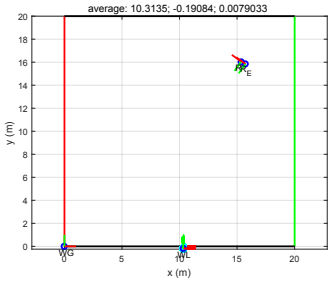
(e) $t = 7$ s. The black bottom keyline is measured.



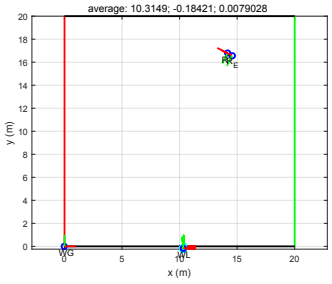
(f) $t = 34$ s. The black bottom and the green keylines are measured.



(g) $t = 107$ s. No keyline is measured.

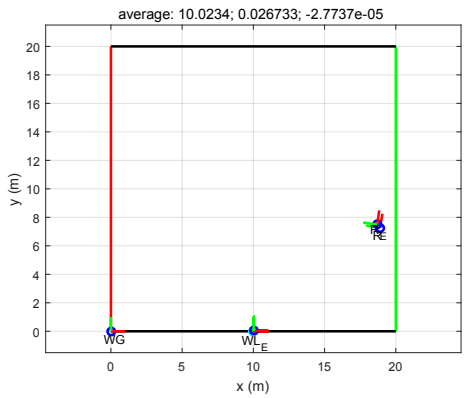


(h) $t = 112$ s. The black top keyline is measured.

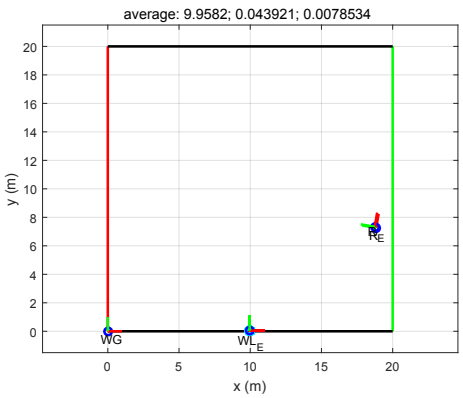


(i) $t = 120$ s. The black top keyline is measured.

Figure 8.10: Evolution of the particles of the global localization component in a simulation.



(a) Simulation 1.



(b) Simulation 2.

Figure 8.11: Performance of the global localization component in simulation.

8.7. Considerations and further improvements

This section explores some of the limitations of the presented perception solution and points out how to overcome them.

Grid points and keylines detection

The grid points and keylines detection, described in Section 8.1, is the first stage of which the local SLAM and the global localization depends. Therefore, the performance of the complete solution is highly correlated to the individual performance of this component. Thus, every improvement in this component will be directly translated into an improvement of the complete solution.

As analyzed in Section 8.6, this component is capable of detecting the keypoints that are in the neighborhood of the robot with a relatively high noise and some false positives. It is desirable that the false positives are eliminated and the noise is reduced. Additionally, it would be beneficial a larger keypoints detection range.

Odometry based robot state estimation

The main weaknesses of the odometry based robot state estimation, presented in Section 8.2, such as the limited kinds of sensors, the simplified robot process model, and the simplified sensors model have been already analyzed in Section 7.6.

Local SLAM

The information provided by the existence of a grid is simplified into the grid intersection points, which is used for the local SLAM, as described in Section 8.4.

Using the grid lines as the measurements of the local SLAM instead of using the grid intersection points would increase the accuracy of the state estimation, reducing the mapping errors due to data association, as the grid lines represent a richer information.

Global localization

As presented in Section 8.5, a particle filter algorithm is used for the global localization. The diversity of the particles is ensured with a complex resampling step of the update stage. This step proposes to resample the particles adding a random noise which amplitude depends on the diversity of the particles, solving the loss of diversity problem, but reducing the performance on the state estimation

To improve the performance of the state estimation, the loss of diversity problem should be avoided, instead of solving it whenever it occurs.

Experiments

As stated in Section 8.6, due to different issues, only partial experiments and simulations were carried out, being the complete perception solution still untested in real experiments. An immediate future work is the complete evaluation of the complete perception solution by means of real experiments.

8.8. Summary

In this chapter, a perception solution that allows the estimation of the state of the aerial robot for an environment with a grid, as in the mission 7 of the IARC Competition (see Section 12.2.3 for more information about the competition) has been presented.

The proposed perception solution is formed by five components: (1) a grid intersection points and arena border lines detection, (2) an odometry based state estimation, (3) an image to 2D feature conversion, (4) a 2D grid based local SLAM, and (5) a 2D arena global localization.

This grid intersection points and arena border lines detection uses the images provided by the RGB cameras to extract the points where the vertical and horizontal lines of the grid intersect (keypoints). Additionally, it has the responsibility of extracting the lines that delimit the borders of the arena (keylines).

The odometry based state estimation provides a drifting estimate of the state of the aerial robot thanks to the information coming from the following sensors: (1) an IMU device, (2) a distance sensor, and (3) a horizontal linear velocity sensor.

The image to 2D feature conversion converts the keypoints and keylines detected from image-based features to 3D spatial features in robot coordinates (not camera coordinates). In addition, it calculates the 2D features of the detected features in robot projection coordinates.

The local SLAM estimates a drift-free 2D pose of the robot, using the intersection points of the grid lines painted on the ground of the arena (keypoints).

The global localization estimates the transformation between the global world reference frame and the local world reference frame using the four arena border lines painted on the ground (keylines).

Due to different issues, only partial experiments and simulations to demonstrate the performance of the proposed perception solution were carried out. These partial results confirm the potential of the proposed solution.

An analysis of the main limitations of the proposed perception solution, together with the proposed future work to overcome them has been carried out.

The proposed perception solution is completely available to the scientific community as an open-source software integrated into Aerostack.

The proposed perception solution that allows the estimation of the state of the aerial robot for an environment with a grid, as in the mission 7 of the IARC Competition, presented in this chapter, can be found on (Pestana et al., 2014d; Sanchez-Lopez et al., 2015).

Chapter 9

Perception based on Multi-sensor Fusion with Environment Reconstruction

This chapter presents a complete perception solution that allows a versatile and robust estimation of the state of the aerial robot, by combining the information coming from different sources such as sensors, feature extraction components, and state estimation algorithms (such as other positioning or SLAM algorithms). In addition, this solution provides an environment reconstruction formed by geometric primitives.

This solution is intended to cover the same functionalities than the one presented in Chapter 7, but with a higher performance and versatility.

The sensor fusion process can be classified in two different approaches depending on how the measurements are used: (1) tightly-coupled, if the raw measurements are fused directly in a single component; and (2) loosely-coupled, if every raw measurement is used only in a small state estimator, and in cascade, a global state estimator fuses their outputs. The complexity of the tightly-coupled approaches is, in general, higher than the loosely-coupled counterpart. Nevertheless, the accuracy of the estimation is, in general, higher in tightly-coupled systems.

The proposed perception solution, represented in Fig. 9.1, is an intermediate approach, that uses some raw measurements directly if they are simple enough (like accelerometers, gyros, etc.), but it uses intermediate state estimators (e.g. positioning or SLAM algorithms) for complex raw measurements (like images, ranging sensors, etc.). This approach permits to have high accuracies with a limited complexity.

The environment is assumed to be formed by cylinders, spheres, and cubes. A set of visual markers are augmenting the environment, labeling its elements with a previously known relationship. Therefore, to be able to reconstruct the environment, the information given by the visual markers is used.

Multi sensor fusion

The higher versatility and robustness of this presented solution is achieved thanks to the capability to fuse the information coming from different sources, including sensors, feature extraction components, and state estimation algorithms, and therefore it is not limited to a particular sensor setup or environment type.

The combination of this information is responsibility of the multi sensor fusion (MSF) state estimation component (deeply described in Section 9.1 and Section 9.2).

In general, this component has a variable number and type of inputs. In the one hand, the state estimator

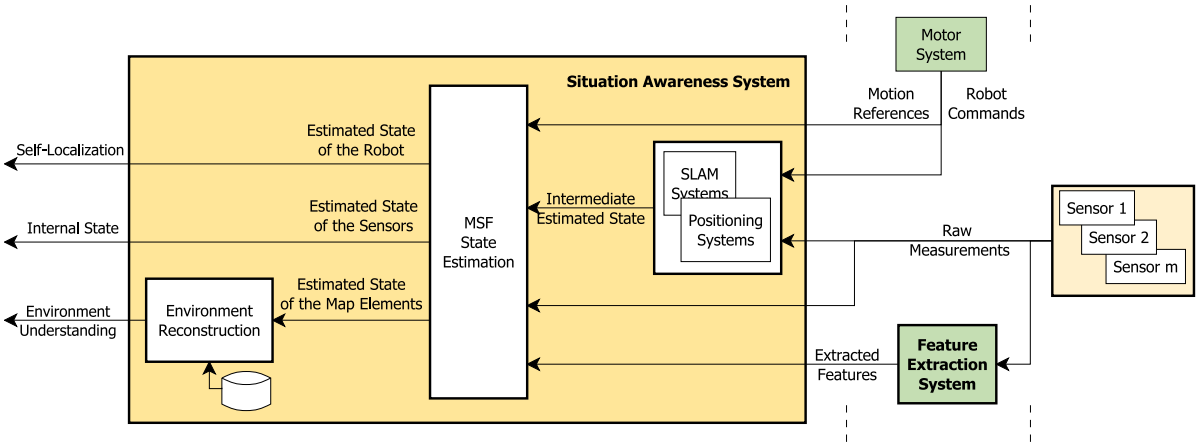


Figure 9.1: Perception solution based on multi sensor fusion with environment reconstruction.

admits *commands* such as motion references and robot commands, that allows having different robot models for the prediction stage. In the other hand, different kinds of inputs can be used as *measurements*, including: (1) the raw measurements given by the sensors, (2) the extracted features given by the Feature Extraction System, and (3) the intermediate estimated state given by other positioning and SLAM systems.

The internal state, parameters, and models of the input sources influence the state estimation given by the MSF state estimation component. Thus, an adequate modeling of these components is essential for an accurate and robust state estimation. Although the MSF state estimation component does not have the capability of modifying the input sources, it is capable of monitoring, estimate and understand some of their internal state and parameters.

An important requirement of this component is that all its inputs (including commands and measurements) have to be precisely timestamped for an accurate performance. As analyzed in Section 2.5, some state estimators require having their inputs synchronized, what, in many cases, can not be easily satisfied due to hardware limitations of the input sources. Some other estimators simply take into account the measurements whenever they arrive, what results on a poor performance when fusing measurements coming from different sources of information such as cameras and accelerometers or gyros, as they different nature might cause delays on the acquisition and feature extraction.

In the case that the inputs are not accurately timestamped due to hardware limitations of the input sources or to software limitations of their software drivers, an approximated timestamp can be set with a software interface, assuming that the performance of the proposed state estimator will slightly decrease. Some works, like (Olson, 2010), explore the problem of accurately timestamp inputs by estimating the delay between the acquisition of the input and its reception.

The MSF state estimation component only takes into account a limited number of map elements like the landmarks of high-level visual features (i.e. visual markers), the world reference frames of the positioning systems, or the ground plane, among others. The proposed system is able to map these elements and use them for the estimation of the robot state, that is, it is performing internally a simultaneous localization and mapping (SLAM). It is required to highlight that the proposed system is not performing a full map estimation and fusion from the used SLAM system, being the full map estimation and fusion from the used mapping components, a remaining future work.

Finally, the outputs of this component are classified into three groups: (1) the complete estimated state of the robot; (2) the estimated internal state of the sensors, feature extraction components, and positioning systems; and (3) the estimated state of the map elements.

Environment reconstruction

It is important to highlight that the proposed solution does not generate a global dense map of the environment. Depending on the used sensor setup and the SLAM systems used, one or several internal semi-dense or dense maps of the environment might be generated. Nevertheless, these maps are not fused together.

The component named environment reconstruction (described in Section 9.3) provides the estimated information related to some map elements in a usable format like the geometric primitives of the obstacles or the ground plane.

The remainder of the chapter is organized as follows: Section 9.1 presents the main algorithm of the MSF state estimation component, whereas Section 9.2 describes its available modules. In Section 9.3, the component for the environment reconstruction is analyzed. In Section 9.4 the performance of the proposed perception solution is evaluated both with real experiments. Section 9.5 points out some lines of future work based on the limitations of the proposed solution. Finally, Section 9.6 sums up the contributions of the chapter.

9.1. Multi-sensor fusion state estimation: algorithm description

As stated before, this component performs the combination of the information coming from a variable number and type of input sources, providing an estimate of the robot state, an estimate of the state of the sensors, and an estimate of the state of the map elements.

This component is designed to be versatile, and thus its inputs depend on the sensor setup and configuration defined by a particular mission or environment. The state estimator is therefore defined as a set of (optional) modules, deeply described in Section 9.2, grouped into three types: (1) world, (2) robot, and (3) sensors.

The proposed state estimator is based on an EKF-SLAM algorithm including the following advanced features:

- Inclusion of stochastic parameters to model uncertainty of fixed coefficients.
- Error-value (i.e. indirect) formulation, accumulating the noise over the error values.
- Quaternions for attitude representation, to avoid singularities.
- Multiplicative solution, to deal with quaternions accurately.
- Time-delayed measurement compensation, by means of a circular buffer.
- Iterative nature, to quickly converge to the real state when the estimated state is far from the real state.

A complete formulation of the EKF-SLAM algorithm with error-values and parameters can be consulted in Appendix D.3.

9.1.1. State and parameters

The presented algorithm considers two types of variables: On the one hand, the state is a random variable whose value changes over the time. On the other hand, parameters are random variables whose values do not change over the time. A normal distribution is assumed to represent both the state and the parameters. In the case in which the covariance of the normal distribution that represents a parameter is zero, then the parameter is deterministic. The use of both stochastic and deterministic parameters increases the accuracy of the state estimation without needing to inflate the state.

9.1.2. Observability of the state

The observability of the state must be satisfied for a particular state estimation setup. For example, if the estimator uses a single sensor that directly measures the pose of the sensor in world frame, the pose of the sensor in robot frame is not observable (cannot be considered as a state) and might be known in advance (considered as a parameter). For the sake of generality, an observability analysis is not detailed in the present thesis.

9.1.3. Formulation using error-values

The presented error-value formulation can distinguish between: (1) true-value ($\boldsymbol{v}$), (2) nominal-value ($\check{\boldsymbol{v}}$), and (3) error-value ($\boldsymbol{\delta v}$). The true-value is expressed as a composition of the nominal-value and the error-value (local perturbation: $\boldsymbol{v} = \check{\boldsymbol{v}} \oplus \boldsymbol{\delta v}$).

The idea to improve the stability and performance of the state estimator is to consider the nominal-value as large-signal (integrable in non-linear fashion) and the error-value as a small signal (linearly integrable and suitable for linear-Gaussian filtering).

9.1.4. Mapping of world elements

The mapping is carried out in a traditional EKF-SLAM fashion by augmenting the existing state with the state of the newly mapped element and therefore the covariance matrix. The performance of the estimator decreases with the number of mapped elements, nevertheless this is not a big inconvenience because the number of mapped elements is assumed to be reduced. The mapped elements are limited to the landmarks of high-level visual features (i.e. visual markers), the world reference frames of the positioning systems, or the ground plane, among others.

9.1.5. Buffer for time-delayed inputs

To be able to accurately incorporate time-delayed inputs, a buffer approach is used. In the buffer, all the inputs and states are stored organized by their timestamp (an example of the buffer is shown in Fig. 9.2).

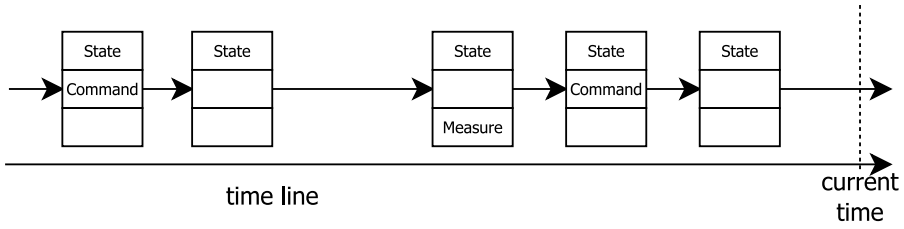
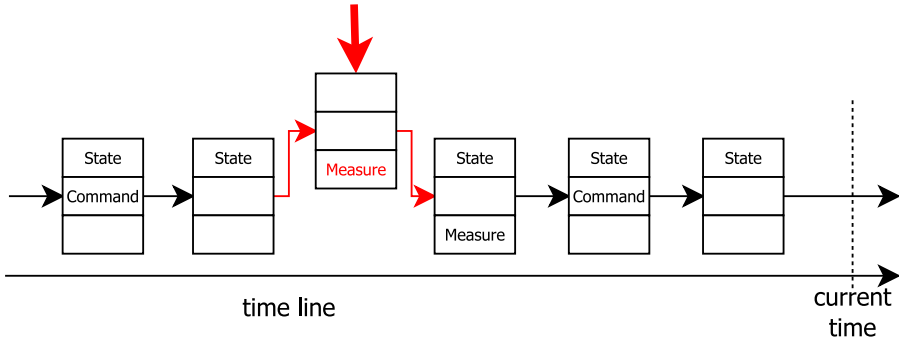


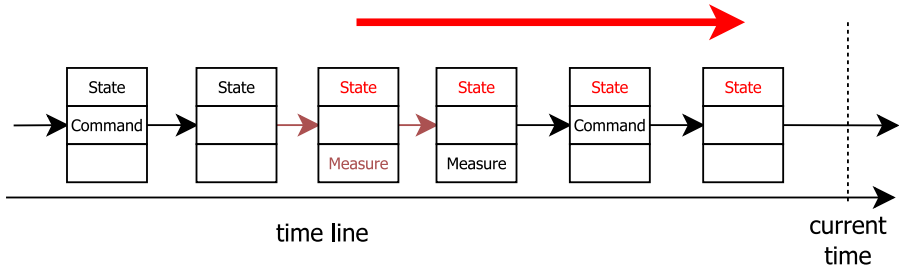
Figure 9.2: Example of the buffer of the MSF state estimator component.

Three different actions can take place in the buffer:

1. A new input (command or measurement) arrives (Fig. 9.3): The new input is added to the buffer in the correct place by its timestamp. After this, an estimation step is done in the newly added buffer element, adding to the newly added buffer element a new state estimation. Finally, the whole buffer (its state) is updated from the newly added buffer element to the newest (by timestamp) element.



(a) The measurement is added to the buffer by its timestamp.



(b) An estimation step is done in the newly added buffer element. Then, the estimated states of the newer (by timestamp) elements of the whole buffer are updated.

Figure 9.3: Process that takes places when a new measurement arrives.

2. New prediction step (Fig. 9.4): The prediction of the state is done synchronously at a constant rate. This allows to accurately integrate the prediction model, keeping the estimated state on the buffer. Additionally, the buffer is cleared from elements that contain only states if there are enough buffer elements with measurements.
3. Request to get the current state estimate (Fig. 9.5): When any component (e.g. the controller) requests an estimate of the state, a prediction is done based on the newest (by timestamp) buffer element.

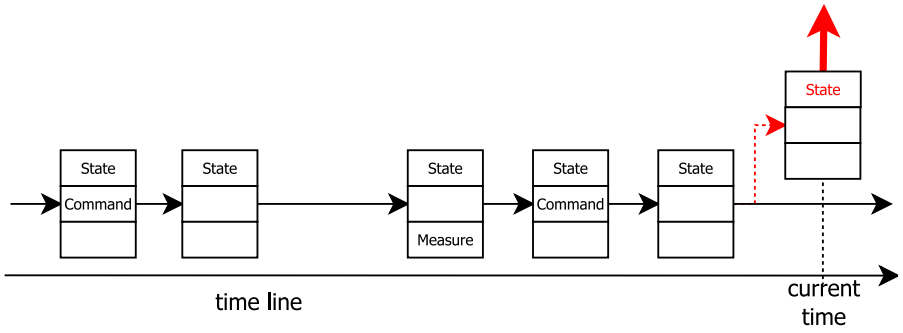


Figure 9.4: Process that takes place when a new prediction step is required.

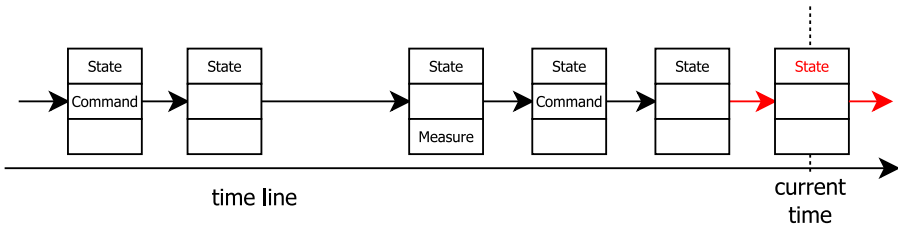


Figure 9.5: Process that takes place when the current state estimation is requested.

9.1.6. Iterative algorithm

The proposed algorithm uses an iterated approach to improving the convergence to the real state when the estimated state is far from the real state. The update stage is iterated until a maximum number of iterations is reached or until the estimate of the state has not changed more than a threshold.

9.1.7. Modularity

As stated before, the state estimator is defined as a set of (optional) modules, deeply described in Section 9.2, grouped into three types: (1) world, (2) robot, and (3) sensors.

Every module has its own contribution to the complete state estimator in terms of state, parameters, and process model. Moreover, the sensor modules have contributions in the measurements and measurement model, and some of them in the mapping model.

Therefore, the total state $\boldsymbol{x}$ is the combination of the state of all the modules, similarly than the total process model $\boldsymbol{f}$, as:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_{W1} \\ \cdots \\ \boldsymbol{x}_{Wi} \\ \cdots \\ \boldsymbol{x}_{R} \\ \boldsymbol{x}_{S1} \\ \cdots \\ \boldsymbol{x}_{Si} \\ \cdots \end{bmatrix} \quad \boldsymbol{f} = \begin{bmatrix} \boldsymbol{f}_{W1} \\ \cdots \\ \boldsymbol{f}_{Wi} \\ \cdots \\ \boldsymbol{f}_{R} \\ \boldsymbol{f}_{S1} \\ \cdots \\ \boldsymbol{f}_{Si} \\ \cdots \end{bmatrix} \tag{9.1}$$

Similarly, the available measurements $\boldsymbol{z}$ is the combination of the available measurements of all the modules, similarly than the total measurement model $\boldsymbol{h}$, as:

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{z}_{S1} \\ \cdots \\ \boldsymbol{z}_{Si} \\ \cdots \end{bmatrix} \quad \boldsymbol{h} = \begin{bmatrix} \boldsymbol{h}_{S1} \\ \cdots \\ \boldsymbol{h}_{Si} \\ \cdots \end{bmatrix} \tag{9.2}$$

9.1.8. Reference frames

All the reference frames involved in this solution, including the world reference frame and the robot reference can be arbitrarily defined, although is is recommended to follow the standards in robotics (for example in ROS[1]).

9.2. Multi-sensor fusion state estimation: modules

The following sections describe the different modules that can be used in the proposed state estimator. They are grouped into three types:

- World, analyzing the acceleration of gravity, the static generic reference frame, and the static plane.
- Robot, analyzing only the "free model"
- Sensors, analyzing the gyro, the accelerometer, the coded visual marker detector, the horizontal linear velocity, the vertical distance, the absolute pose, and the unscaled absolute pose.

All the sections have the same structure: first of all, the existing state and parameters are introduced. Following, the continuous-time true-value process model is presented. Then, the discrete-time true-value is calculated by integration. Finally, the existing Jacobian matrices of the discrete-time error-value process model are indicated. In the specific case of the sensors, three extra items are explored: the existing measurements, the true-value measurement model, and the Jacobian matrices of the error-value measurement model. Additionally, in the specific case of the sensors with world elements susceptible to be mapped, two extra items are explored: the true-value mapping model, and the Jacobian matrices of the error-value mapping model.

9.2.1. World: acceleration of gravity

This module includes the gravitational acceleration.

State and parameters

The state and parameters are presented in Table 9.1.

Magnitude	True-value	Nominal-value	Error-value	Composition
Gravity in world coordinates	$\boldsymbol{g}^W_{\vert W}$	$\check{\boldsymbol{g}}^W_{\vert W}$	$\delta\boldsymbol{g}^W_{\vert W}$	$\delta\boldsymbol{g}^W_{\vert W} = -\check{\boldsymbol{g}}^W_{\vert W} + \boldsymbol{g}^W_{\vert W}$

Table 9.1: State and parameters of the acceleration of gravity world module.

Continuous-time true-value process model

The gravity in world coordinates is modeled as a constant value, and therefore, the continuous-time true-value process model is described by the equations:

$$\dot{\boldsymbol{g}}^W_{\vert W} = \boldsymbol{0} \tag{9.3}$$

Discrete-time true-value process model

The discrete-time true-value process model is described by the equations:

$$\boldsymbol{g}^W_{\vert W}(k) = \boldsymbol{g}^W_{\vert W}(k-1) \tag{9.4}$$

Jacobian matrices of the discrete-time error-value process model

The existing Jacobian matrices of the discrete-time error-value process model are indicated in Table 9.2.

k \ $k-1$	$\partial\delta\boldsymbol{g}^W_{\vert W}$
$\partial\delta\boldsymbol{g}^W_{\vert W}$	$\boldsymbol{I}_{3\times3}$

Table 9.2: Jacobian matrices of the acceleration of gravity world error-value discrete-time process model: error-value in k vs. error-value in $k-1$.

[1] Online: `http://www.ros.org/reps/rep-0105.html`

9.2.2. World: static generic reference frame

This module describes a static generic reference frames, like the sensor world reference frames or the coded visual markers reference frames.

State and parameters

The state and parameters are presented in Table 9.3.

Magnitude		**True value**	**Nominal value**	**Error value**	**Composition** (local perturbation)
Pose of the reference frame in world coordinates	Position	$\boldsymbol{t}_{RF\star}^{W}$	$\check{\boldsymbol{t}}_{RF\star}^{W}$	$\delta\boldsymbol{t}_{RF\star}^{W}$	$\delta\boldsymbol{t}_{RF\star}^{W} = -\check{\boldsymbol{t}}_{RF\star}^{W} + \boldsymbol{t}_{RF\star}^{W}$
	Attitude(1)	$\boldsymbol{q}_{RF\star}^{W}$	$\check{\boldsymbol{q}}_{RF\star}^{W}$	$\delta\boldsymbol{q}_{RF\star}^{W}$	$\delta\boldsymbol{q}_{RF\star}^{W} = \left(\check{\boldsymbol{q}}_{RF\star}^{W}\right)^{*} \otimes \boldsymbol{q}_{RF\star}^{W}$

(1) Being: $\delta\boldsymbol{q}_{RF\star}^{W} \approx \begin{bmatrix} 1 \\ \frac{1}{2}\cdot\delta\boldsymbol{\theta}_{RF\star}^{W} \end{bmatrix}$

Table 9.3: State and parameters of the generic reference frame world module.

Continuous-time true-value process model

The reference frame is designed as a static element of the world, thus the pose of the reference frame in world coordinates is modeled as a constant value, and therefore, the continuous-time true-value process model is described by the equations:

$$\dot{\boldsymbol{t}}_{RF\star}^{W} = \boldsymbol{0} \tag{9.5}$$

$$\dot{\boldsymbol{q}}_{RF\star}^{W} = \boldsymbol{0} \tag{9.6}$$

Discrete-time true-value process model

The discrete-time true-value process model is described by the equations:

$$\boldsymbol{t}_{RF\star}^{W}(k) = \boldsymbol{t}_{RF\star}^{W}(k-1) \tag{9.7}$$

$$\boldsymbol{q}_{RF\star}^{W}(k) = \boldsymbol{q}_{RF\star}^{W}(k-1) \tag{9.8}$$

Jacobian matrices of the discrete-time error-value process model

The existing Jacobian matrices of the discrete-time error-value process model are indicated in Table 9.4.

k \ $k-1$	$\partial\delta\boldsymbol{t}_{RF\star}^{W}$	$\partial\delta\boldsymbol{\theta}_{RF\star}^{W}$
$\partial\delta\boldsymbol{t}_{RF\star}^{W}$	$\boldsymbol{I}_{3\times3}$	$\boldsymbol{0}$
$\partial\delta\boldsymbol{\theta}_{RF\star}^{W}$	$\boldsymbol{0}$	$\boldsymbol{I}_{3\times3}$

Table 9.4: Jacobian matrices of the generic reference frame world error-value discrete-time process model: error-value in k vs. error-value in $k-1$.

9.2.3. World: static plane

This module describes a static plane in the space, for example, the ground plane.

State and parameters

The state and parameters are presented in Table 9.5.

Magnitude		**True value**	**Nominal value**	**Error value**	**Composition** (local perturbation)
Plane descriptors in world coordinates	Normal Vector	$\boldsymbol{n}_{P_\star}^W$	$\check{\boldsymbol{n}}_{P_\star}^W$	$\delta\boldsymbol{n}_{P_\star}^W$	$\delta\boldsymbol{n}_{P_\star}^W = -\check{\boldsymbol{n}}_{P_\star}^W + \boldsymbol{n}_{P_\star}^W$
	Point	$\boldsymbol{x}_{P_\star}^W$	$\check{\boldsymbol{x}}_{P_\star}^W$	$\delta\boldsymbol{x}_{P_\star}^W$	$\delta\boldsymbol{x}_{P_\star}^W = -\check{\boldsymbol{x}}_{P_\star}^W + \boldsymbol{x}_{P_\star}^W$

Table 9.5: State and parameters of the plane world module.

Continuous-time true-value process model

The plane is designed as a static element of the world, thus the plane descriptors in world coordinates are modeled as a constant value, and therefore, the continuous-time true-value process model is described by the equations:

$$\dot{\boldsymbol{n}}_{P_\star}^W = \boldsymbol{0} \tag{9.9}$$
$$\dot{\boldsymbol{x}}_{P_\star}^W = \boldsymbol{0} \tag{9.10}$$

Discrete-time true-value process model

The discrete-time true-value process model is described by the equations:

$$\boldsymbol{n}_{P_\star}^W(k) = \boldsymbol{n}_{P_\star}^W(k-1) \tag{9.11}$$
$$\boldsymbol{x}_{P_\star}^W(k) = \boldsymbol{x}_{P_\star}^W(k-1) \tag{9.12}$$

Jacobian matrices of the discrete-time error-value process model

The existing Jacobian matrices of the discrete-time error-value process model are indicated in Table 9.6.

k \ $k-1$	$\partial\delta\boldsymbol{x}_{P_\star}^W$	$\partial\delta\boldsymbol{n}_{P_\star}^W$
$\partial\delta\boldsymbol{x}_{P_\star}^W$	$\boldsymbol{I}_{3\times3}$	$\boldsymbol{0}$
$\partial\delta\boldsymbol{n}_{P_\star}^W$	$\boldsymbol{0}$	$\boldsymbol{I}_{3\times3}$

Table 9.6: Jacobian matrices of the plane world error-value discrete-time process model: error-value in k vs. error-value in $k-1$.

9.2.4. Robot: free-model

This module includes the robot information (state and model). The reference frames involved in this module are shown in Fig. 9.6.

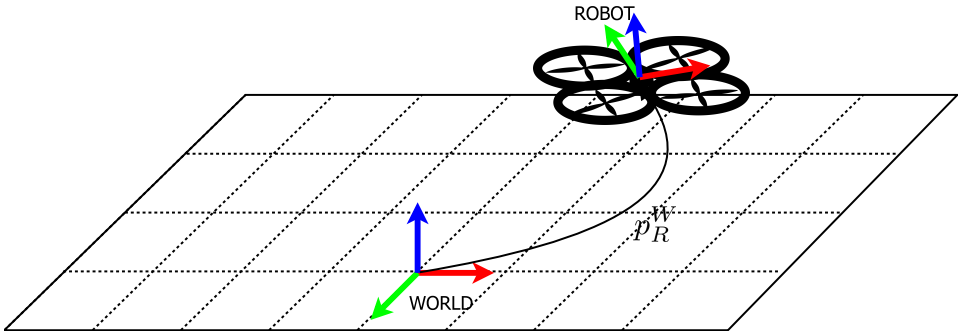


Figure 9.6: Robot module reference frames.

The used model does not include any input command, estimating the full state up to the acceleration. It neither includes any parameter. The main advantages of this model are: (1) it estimates the full robot state which

increases the compatibility with the sensor modules; (2) it does not depend on any input so it is more versatile (i.e. it might be used for any robot type); (3) it is faster because it represents only the minimal robot state. The main disadvantages are: (1) as it does not depend on any input, its prediction is only useful in the very short term, quickly diverging and being highly dependent on having some extra sensors to update it; (2) it does not represent any extra robot state that might be useful (e.g. battery level).

State and parameters

The state and parameters are presented in Table 9.7.

Magnitude		**True value**	**Nominal value**	**Error value**	**Composition** (local perturbation)
Pose of the robot in world coordinates	Position	$\boldsymbol{t}_R^W$	$\check{\boldsymbol{t}}_R^W$	$\delta\boldsymbol{t}_R^W$	$\delta\boldsymbol{t}_R^W = -\check{\boldsymbol{t}}_R^W + \boldsymbol{t}_R^W$
	Attitude(1)	$\boldsymbol{q}_R^W$	$\check{\boldsymbol{q}}_R^W$	$\delta\boldsymbol{q}_R^W$	$\delta\boldsymbol{q}_R^W = \left(\check{\boldsymbol{q}}_R^W\right)^* \otimes \boldsymbol{q}_R^W$
Velocity of the robot wrt. world in world coordinates	Linear	$\boldsymbol{v}_{R\|W}^W$	$\check{\boldsymbol{v}}_{R\|W}^W$	$\delta\boldsymbol{v}_{R\|W}^W$	$\delta\boldsymbol{v}_{R\|W}^W = -\check{\boldsymbol{v}}_{R\|W}^W + \boldsymbol{v}_{R\|W}^W$
	Angular	$\boldsymbol{\omega}_{R\|W}^W$	$\check{\boldsymbol{\omega}}_{R\|W}^W$	$\delta\boldsymbol{\omega}_{R\|W}^W$	$\delta\boldsymbol{\omega}_{R\|W}^W = -\check{\boldsymbol{\omega}}_{R\|W}^W + \boldsymbol{\omega}_{R\|W}^W$
Acceleration of the robot wrt. world in world coordinates	Linear	$\boldsymbol{a}_{R\|W}^W$	$\check{\boldsymbol{a}}_{R\|W}^W$	$\delta\boldsymbol{a}_{R\|W}^W$	$\delta\boldsymbol{a}_{R\|W}^W = -\check{\boldsymbol{a}}_{R\|W}^W + \boldsymbol{a}_{R\|W}^W$
	Angular	$\boldsymbol{\alpha}_{R\|W}^W$	$\check{\boldsymbol{\alpha}}_{R\|W}^W$	$\delta\boldsymbol{\alpha}_{R\|W}^W$	$\delta\boldsymbol{\alpha}_{R\|W}^W = -\check{\boldsymbol{\alpha}}_{R\|W}^W + \boldsymbol{\alpha}_{R\|W}^W$

(1) Note: $\delta\boldsymbol{q}_R^W \approx \begin{bmatrix} 1 \\ \frac{1}{2}\cdot\delta\boldsymbol{\theta}_R^W \end{bmatrix}$

Table 9.7: State and parameters of the model-free robot module.

Continuous-time true-value process model

The robot is modeled as a constant acceleration (both linear and angular), and therefore, its continuous-time true-value process model is described by the equations:

$$\dot{\boldsymbol{t}}_R^W = \boldsymbol{v}_{R|W}^W \tag{9.13}$$

$$\dot{\boldsymbol{v}}_{R|W}^W = \boldsymbol{a}_{R|W}^W \tag{9.14}$$

$$\dot{\boldsymbol{a}}_{R|W}^W = \boldsymbol{n}_{c_{a_{R|W}^W}} \tag{9.15}$$

$$\dot{\boldsymbol{q}}_R^W = \frac{1}{2}\cdot\boldsymbol{\omega}_{R|W}^W \otimes \boldsymbol{q}_R^W \tag{9.16}$$

$$\dot{\boldsymbol{\omega}}_{R|W}^W = \boldsymbol{\alpha}_{R|W}^W \tag{9.17}$$

$$\dot{\boldsymbol{\alpha}}_{R|W}^W = \boldsymbol{n}_{c_{\alpha_{R|W}^W}} \tag{9.18}$$

Note that a global perturbation for the quaternion has been used ($\boldsymbol{\omega}$ is defined in world coordinates), see Appendix C.

$\boldsymbol{n}_{c_{a_{R|W}^W}}$ and $\boldsymbol{n}_{c_{\alpha_{R|W}^W}}$ represent the continuous-time noise of the true-value process model of the linear acceleration and angular acceleration of the robot wrt. world in world coordinates, respectively.

Discrete-time true-value process model

The discrete-time true-value process model is described by the equations (see Appendix B):

$$\boldsymbol{t}_R^W(k) = \boldsymbol{t}_R^W(k-1) + \boldsymbol{v}_{R|W}^W(k-1)\cdot\Delta t + \frac{1}{2}\cdot\boldsymbol{a}_{R|W}^W(k-1)\cdot\Delta t^2 \tag{9.19}$$

$$\boldsymbol{v}_{R|W}^W(k) = \boldsymbol{v}_{R|W}^W(k-1) + \boldsymbol{a}_{R|W}^W(k-1)\cdot\Delta t \tag{9.20}$$

$$\boldsymbol{a}_{R|W}^W(k) = \boldsymbol{a}_{R|W}^W(k-1) + \boldsymbol{n}_{d_{a_{R|W}^W}} \tag{9.21}$$

$$\boldsymbol{q}_R^W(k) = \delta\boldsymbol{q} \otimes \boldsymbol{q}_R^W(k-1) \tag{9.22}$$

$$\boldsymbol{\omega}_{R|W}^W(k) = \boldsymbol{\omega}_{R|W}^W(k-1) + \boldsymbol{\alpha}_{R|W}^W(k-1)\cdot\Delta t \tag{9.23}$$

$$\boldsymbol{\alpha}_{R|W}^W(k) = \boldsymbol{\alpha}_{R|W}^W(k-1) + \boldsymbol{n}_{d_{\alpha_{R|W}^W}} \tag{9.24}$$

where:

$$\delta \boldsymbol{q} = \delta \boldsymbol{q}\left[\bar{\boldsymbol{\omega}} \cdot \Delta t\right] + \frac{\Delta t^2}{24} \cdot \delta \boldsymbol{q}[\boldsymbol{\alpha}] \tag{9.25}$$

being $\delta \boldsymbol{q}\left[\bar{\boldsymbol{\omega}} \cdot \Delta t\right]$ the operation rotation vector to quaternion of:

$$\bar{\boldsymbol{\omega}} \cdot \Delta t = \left(\boldsymbol{\omega}^W_{R|W}(k-1) + \frac{1}{2} \cdot \boldsymbol{\alpha}^W_{R|W}(k-1) \cdot \Delta t\right) \cdot \Delta t \tag{9.26}$$

and, being:

$$\delta \boldsymbol{q}[\boldsymbol{\alpha}] = \begin{bmatrix} 0 \\ \boldsymbol{\alpha}^W_{R|W}(k-1) \times \left(\boldsymbol{\omega}^W_{R|W}(k)\right) \end{bmatrix} \tag{9.27}$$

It is important to note that $\delta \boldsymbol{q}[\boldsymbol{\alpha}]$ is not a unitary quaternion. Therefore $\delta \boldsymbol{q}$ is not a unitary quaternion (additionally the $+$ operation does not preserve the unitary norm), so after getting $\boldsymbol{q}^W_R(k)$, it must be normalized.

$\boldsymbol{n}_{d\,a^W_{R|W}}$ and $\boldsymbol{n}_{d\,\alpha^W_{R|W}}$ represent the discrete-time noise of the true-value process model of the linear acceleration and angular acceleration of the robot wrt. world in world coordinates, respectively.

Jacobian matrices of the discrete-time error-value process model

The existing Jacobian matrices of the discrete-time error-value process model are indicated in Table 9.8.

k \ $k-1$	$\partial \delta t^W_R$	$\partial \delta v^W_{R\|W}$	$\partial \delta a^W_{R\|W}$	$\partial \delta \theta^W_R$	$\partial \delta \omega^W_{R\|W}$	$\partial \delta \alpha^W_{R\|W}$
$\partial \delta t^W_R$	$I_{3\times3}$	$\Delta t \cdot I_{3\times3}$	$\frac{1}{2}\Delta t^2 \cdot I_{3\times3}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$
$\partial \delta v^W_{R\|W}$	$\mathbf{0}$	$I_{3\times3}$	$\Delta t \cdot I_{3\times3}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$
$\partial \delta a^W_{R\|W}$	$\mathbf{0}$	$\mathbf{0}$	$I_{3\times3}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$
$\partial \delta \theta^W_R$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\left[\frac{\partial \delta \boldsymbol{\theta}^W_R(k)}{\partial \delta \boldsymbol{\theta}^W_R(k-1)}\right]_{3\times3}$	$\left[\frac{\partial \delta \boldsymbol{\theta}^W_R(k)}{\partial \delta \boldsymbol{\omega}^W_{R\|W}(k-1)}\right]_{3\times3}$	$\left[\frac{\partial \delta \boldsymbol{\theta}^W_R(k)}{\partial \delta \boldsymbol{\alpha}^W_{R\|W}(k-1)}\right]_{3\times3}$
$\partial \delta \omega^W_{R\|W}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$I_{3\times3}$	$\Delta t \cdot I_{3\times3}$
$\partial \delta \alpha^W_{R\|W}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$I_{3\times3}$

(a) Error-value in k vs. error-value in $k-1$.

k \ $k-1$	$\partial \boldsymbol{n}_{d\,\delta a^W_{R\|W}}$	$\partial \boldsymbol{n}_{d\,\delta \alpha^W_{R\|W}}$
$\partial \delta \boldsymbol{t}^W_R$	$\mathbf{0}$	$\mathbf{0}$
$\partial \delta \boldsymbol{v}^W_{R\|W}$	$\mathbf{0}$	$\mathbf{0}$
$\partial \delta \boldsymbol{a}^W_{R\|W}$	$\boldsymbol{I}_{3\times3}$	$\mathbf{0}$
$\partial \delta \boldsymbol{\theta}^W_R$	$\mathbf{0}$	$\mathbf{0}$
$\partial \delta \boldsymbol{\omega}^W_{R\|W}$	$\mathbf{0}$	$\mathbf{0}$
$\partial \delta \boldsymbol{\alpha}^W_{R\|W}$	$\mathbf{0}$	$\boldsymbol{I}_{3\times3}$

(b) Error-value in k vs. error-noise in $k-1$.

Table 9.8: Jacobian matrices of the free-model robot error-value discrete-time process model.

9.2.5. Sensor: common

This module includes the common descriptor that every sensor has. The reference frames involved in this module are shown in Fig. 9.7.

State and parameters

The state and parameters are presented in Table 9.9.

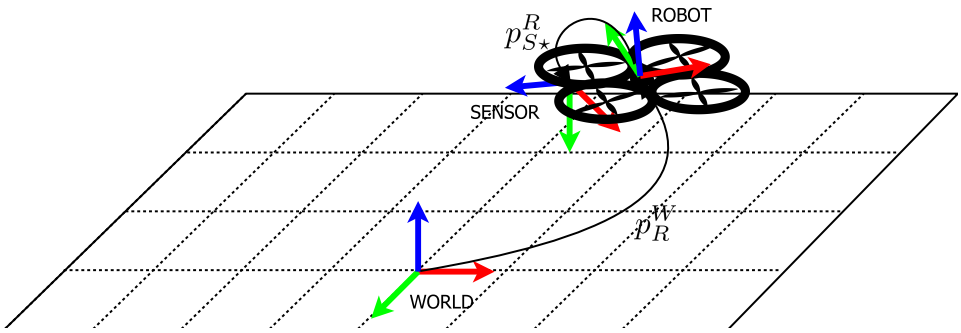


Figure 9.7: Common sensor reference frames.

Magnitude		**True value**	**Nominal value**	**Error value**	**Composition** (local perturbation)
Pose of the sensor in robot coordinates	Position	$\boldsymbol{t}^R_{S\star}$	$\check{\boldsymbol{t}}^R_{S\star}$	$\delta\boldsymbol{t}^R_{S\star}$	$\delta\boldsymbol{t}^R_{S\star} = -\check{\boldsymbol{t}}^R_{S\star} + \boldsymbol{t}^R_{S\star}$
	Attitude(1)	$\boldsymbol{q}^R_{S\star}$	$\check{\boldsymbol{q}}^R_{S\star}$	$\delta\boldsymbol{q}^R_{S\star}$	$\delta\boldsymbol{q}^R_{S\star} = \left(\check{\boldsymbol{q}}^R_{S\star}\right)^* \otimes \boldsymbol{q}^R_{S\star}$

(1) Being: $\delta\boldsymbol{q}^R_{S\star} \approx \begin{bmatrix} 1 \\ \frac{1}{2}\cdot\delta\boldsymbol{\theta}^R_{S\star} \end{bmatrix}$

Table 9.9: State and parameters of the sensor common module.

Continuous-time true-value process model

The sensors are designed as static elements of the robot, thus the pose of the sensor in world coordinates is modeled as a constant value, and therefore, the continuous-time true-value process model is described by the equations:

$$\dot{\boldsymbol{t}}^R_{S\star} = \boldsymbol{0} \tag{9.28}$$
$$\dot{\boldsymbol{q}}^R_{S\star} = \boldsymbol{0} \tag{9.29}$$

Discrete-time true-value process model

The discrete-time true-value process model is described by the equations:

$$\boldsymbol{t}^R_{S\star}(k) = \boldsymbol{t}^R_{S\star}(k-1) \tag{9.30}$$
$$\boldsymbol{q}^R_{S\star}(k) = \boldsymbol{q}^R_{S\star}(k-1) \tag{9.31}$$

Jacobian matrices of the discrete-time error-value process model

The existing Jacobian matrices of the discrete-time error-value process model are indicated in Table 9.10.

k \ $k-1$	$\partial\delta\boldsymbol{t}^R_{S\star}$	$\partial\delta\boldsymbol{\theta}^R_{S\star}$
$\partial\delta\boldsymbol{t}^R_{S\star}$	$\boldsymbol{I}_{3\times3}$	$\boldsymbol{0}$
$\partial\delta\boldsymbol{\theta}^R_{S\star}$	$\boldsymbol{0}$	$\left[\frac{\partial\delta\boldsymbol{\theta}^R_{S\star}(k)}{\partial\delta\boldsymbol{\theta}^R_{S\star}(k-1)}\right]_{3\times3}$

Table 9.10: Jacobian matrices of the common sensor error-value discrete-time process model: error-value in k vs. error-value in $k-1$.

Useful relationships

The following relationship with the robot pose can be found:

$$\boldsymbol{t}_{S\star}^{W} = \boldsymbol{q}_{R}^{W} \otimes \boldsymbol{t}_{S\star}^{R} \otimes \left(\boldsymbol{q}_{R}^{W}\right)^{*} + \boldsymbol{t}_{R}^{W} \tag{9.32}$$

$$\boldsymbol{q}_{S\star}^{W} = \boldsymbol{q}_{R}^{W} \otimes \boldsymbol{q}_{S\star}^{R} \tag{9.33}$$

9.2.6. Sensor: gyro

This module describes the gyro sensor.

State and parameters

It uses the common sensor state and parameters of Table 9.9 and the ones presented in Table 9.11 and Table 9.12.

Magnitude	**True value**	**Nominal value**	**Error value**	**Composition** (local perturbation)
Bias of the gyro	$\boldsymbol{b}_{gyro}$	$\check{\boldsymbol{b}}_{gyro}$	$\delta\boldsymbol{b}_{gyro}$	$\delta\boldsymbol{b}_{gyro} = -\check{\boldsymbol{b}}_{gyro} + \boldsymbol{b}_{gyro}$

Table 9.11: State and parameters of the gyro sensor module.

Magnitude	**Value**
Sensitivity of the gyro	$\boldsymbol{S}_{gyro}$

Table 9.12: Parameters with zero covariance of the gyro sensor module.

Continuous-time true-value process model

The continuous-time true-value process model of the gyro is described by the common sensor model, together with the equations of the dynamics of the non-static biases, that are modeled as a random process, because they randomly change their value with the time, and therefore, their associated continuous-time true-value process model is given by the equations:

$$\dot{\boldsymbol{b}}_{gyro} = \boldsymbol{n}_{cb_{gyro}} \tag{9.34}$$

$\boldsymbol{n}_{cb_{gyro}}$ represents the continuous-time noise of the true-value process model of the bias of the gyro.

Discrete-time true-value process model

The discrete-time true-value process model is described by the common sensor together with the equations:

$$\boldsymbol{b}_{gyro}(k) = \boldsymbol{b}_{gyro}(k-1) + \boldsymbol{n}_{db_{gyro}} \tag{9.35}$$

$\boldsymbol{n}_{db_{gyro}}$ represents the discrete-time noise of the true-value process model of the bias of the gyro.

Jacobian matrices of the discrete-time error-value process model

The existing Jacobian matrices of the discrete-time error-value process model are the ones shown in the common sensor together with the ones indicated in Table 9.13.

Measurements

The measurements are presented in Table 9.14.

k \ $k-1$	$\partial\delta b_{gyro}$
$\partial\delta b_{gyro}$	$I_{3\times3}$

(a) Error-value in k vs. error-value in $k-1$.

k \ $k-1$	$\partial n_{d\,\delta b_{gyro}}$
$\partial\delta b_{gyro}$	$I_{3\times3}$

(b) Error-value in k vs. error-noise in $k-1$.

Table 9.13: Jacobian matrices of the gyro sensor error-value discrete-time process model.

Magnitude	**True value**	**Nominal value**	**Error value**	**Composition** (local perturbation)
Measurement of the gyro	z_{gyro}	$\check{z}_{gyro}$	δz_{gyro}	$\delta z_{gyro} = -\check{z}_{gyro} + z_{gyro}$

Table 9.14: Measurements of the gyro sensor module.

True-value measurement model

The true-value measurement model is described by the equations:

$$z_{gyro} = S_{gyro} \cdot \omega_{S_G|W}^{S_G} + b_{gyro} + n_{gyro} \tag{9.36}$$

being:

$$\omega_{S_G|W}^{S_G} = (q_{S_G}^W)^* \otimes \omega_{R|W}^W \otimes q_{S_G}^W \tag{9.37}$$

$$= (q_R^W \otimes q_{S_G}^R)^* \otimes \omega_{R|W}^W \otimes (q_R^W \otimes q_{S_G}^R) \tag{9.38}$$

n_{gyro} is the noise of the true-value measurement model of the gyro sensor module.

Jacobian matrices of the error-value measurement model

The existing Jacobian matrices of the error-value measurement model are indicated in Table 9.15.

	$\partial\delta\theta_R^W$	$\partial\delta\omega_{R|W}^W$
$\partial\delta z_{gyro}$	$\left[\frac{\partial\delta z_{gyro}}{\partial\delta\theta_R^W}\right]_{3\times3}$	$\left[\frac{\partial\delta z_{gyro}}{\partial\delta\omega_{R|W}^W}\right]_{3\times3}$

(a) Error-value vs. robot error-value.

	$\partial\delta t_{S_G}^R$	$\partial\delta\theta_{S_G}^R$	$\partial\delta b_{gyro}$
$\partial\delta z_{gyro}$	$\mathbf{0}$	$\left[\frac{\partial\delta z_{gyro}}{\partial\delta\theta_{S_G}^R}\right]_{3\times3}$	$I_{3\times3}$

(b) Error-value vs. sensor error-value.

	$\partial n_{\delta gyro}$
$\partial\delta z_{gyro}$	$I_{3\times3}$

(c) Error-value vs. error-noise.

Table 9.15: Jacobian matrices of the gyro sensor error-value measurement model.

9.2.7. Sensor: accelerometer

This module describes the accelerometer sensor.

Magnitude	**True value**	**Nominal value**	**Error value**	**Composition** (local perturbation)
Bias of the accelerometer	$\boldsymbol{b}_{accel}$	$\check{\boldsymbol{b}}_{accel}$	$\delta\boldsymbol{b}_{accel}$	$\delta\boldsymbol{b}_{accel} = -\check{\boldsymbol{b}}_{accel} + \boldsymbol{b}_{accel}$

Table 9.16: State and parameters of the accelerometer sensor module.

Magnitude	**Value**
Sensitivity of the accelerometer	$\boldsymbol{S}_{accel}$

Table 9.17: Parameters with zero covariance of the accelerometer sensor module.

State and parameters

It uses the common sensor state and parameters of Table 9.9 and the ones presented in Table 9.16 and Table 9.17.

Continuous-time true-value process model

The continuous-time true-value process model of the accelerometer is described by the common sensor model, together with the equations of the dynamics of the non-static biases, that are modeled as a random process, because they randomly change their value with the time, and therefore, their associated continuous-time true-value process model is given by the equations:

$$\dot{\boldsymbol{b}}_{accel} = \boldsymbol{n}_{cb_{accel}} \tag{9.39}$$

$\boldsymbol{n}_{cb_{accel}}$ represents the continuous-time noise of the true-value process model of the bias of the accelerometer.

Discrete-time true-value process model

The discrete-time true-value process model is described by the common sensor together with the equations:

$$\boldsymbol{b}_{accel}(k) = \boldsymbol{b}_{accel}(k-1) + \boldsymbol{n}_{db_{accel}} \tag{9.40}$$

$\boldsymbol{n}_{db_{accel}}$ represents the discrete-time noise of the true-value process model of the bias of the accelerometer.

Jacobian matrices of the discrete-time error-value process model

The existing Jacobian matrices of the discrete-time error-value process model are the ones shown in the common sensor together with the ones indicated in Table 9.18.

k \ $k-1$	$\partial\delta\boldsymbol{b}_{accel}$
$\partial\delta\boldsymbol{b}_{accel}$	$\boldsymbol{I}_{3\times3}$

(a) Error-value in k vs. error-value in $k-1$.

k \ $k-1$	$\partial\boldsymbol{n}_{d\delta b_{accel}}$
$\partial\delta\boldsymbol{b}_{accel}$	$\boldsymbol{I}_{3\times3}$

(b) Error-value in k vs. error-noise in $k-1$.

Table 9.18: Jacobian matrices of the accelerometer sensor error-value discrete-time process model.

Measurements

The measurements are presented in Table 9.19.

Magnitude	**True value**	**Nominal value**	**Error value**	**Composition** (local perturbation)
Measurement of the accelerometer	z_{accel}	$\check{z}_{accel}$	δz_{accel}	$\delta z_{accel} = -\check{z}_{accel} + z_{accel}$

Table 9.19: Measurements of the accelerometer sensor module.

True-value measurement model

The true-value measurement model is described by the equations:

$$\boldsymbol{z}_{accel} = \boldsymbol{S}_{accel} \cdot \boldsymbol{a}^{S_A}_{sp-S_A|W} + \boldsymbol{b}_{accel} + \boldsymbol{n}_{accel} \tag{9.41}$$

being the proper acceleration, also called specific acceleration:

$$\boldsymbol{a}^{S_A}_{sp-S_A|W} = \boldsymbol{a}^{S_A}_{S_A|W} - \boldsymbol{g}^{S_A}_{|W} \tag{9.42}$$

being:

$$\boldsymbol{g}^{S_A}_{|W} = \left(\boldsymbol{q}^W_{S_A}\right)^* \otimes \boldsymbol{g}^W_{|W} \otimes \boldsymbol{q}^W_{S_A} \tag{9.43}$$

$$= \left(\boldsymbol{q}^W_R \otimes \boldsymbol{q}^R_{S_A}\right)^* \otimes \boldsymbol{g}^W_{|W} \otimes \left(\boldsymbol{q}^W_R \otimes \boldsymbol{q}^R_{S_A}\right) \tag{9.44}$$

and being:

$$\boldsymbol{a}^{S_A}_{S_A|W} = \left(\boldsymbol{q}^W_{S_A}\right)^* \otimes \boldsymbol{a}^W_{R|W} \otimes \boldsymbol{q}^W_{S_A} + \left(\boldsymbol{q}^R_{S_A}\right)^* \otimes \boldsymbol{a}^R_{Fict|W} \otimes \boldsymbol{q}^R_{S_A} \tag{9.45}$$

$$= \left(\boldsymbol{q}^W_R \otimes \boldsymbol{q}^R_{S_A}\right)^* \otimes \boldsymbol{a}^W_{R|W} \otimes \left(\boldsymbol{q}^W_R \otimes \boldsymbol{q}^R_{S_A}\right) + \left(\boldsymbol{q}^R_{S_A}\right)^* \otimes \boldsymbol{a}^R_{Fict|W} \otimes \boldsymbol{q}^R_{S_A} \tag{9.46}$$

where:

$$\boldsymbol{a}^R_{Fict|W} = \boldsymbol{a}^R_{Fict-Norm|W} + \boldsymbol{a}^R_{Fict-Tan|W} \tag{9.47}$$

being:

$$\boldsymbol{a}^R_{Fict-Norm|W} = \boldsymbol{\omega}^R_{R|W} \times \boldsymbol{\omega}^R_{R|W} \times \boldsymbol{t}^R_{S_A} \tag{9.48}$$

$$\boldsymbol{a}^R_{Fict-Tan|W} = \boldsymbol{\alpha}^R_{R|W} \times \boldsymbol{t}^R_{S_A} \tag{9.49}$$

where:

$$\boldsymbol{\omega}^R_{R|W} = \left(\boldsymbol{q}^W_R\right)^* \otimes \boldsymbol{\omega}^W_{R|W} \otimes \boldsymbol{q}^W_R \tag{9.50}$$

$$\boldsymbol{\alpha}^R_{R|W} = \left(\boldsymbol{q}^W_R\right)^* \otimes \boldsymbol{\alpha}^W_{R|W} \otimes \boldsymbol{q}^W_R \tag{9.51}$$

$\boldsymbol{n}_{accel}$ is the noise of the true-value measurement model of the accelerometer sensor module.

As can be seen, the measurements depend on the acceleration of Gravity world module, that has the information of the acceleration of Gravity in world coordinates.

Jacobian matrices of the error-value measurement model

The existing Jacobian matrices of the error-value measurement model are indicated in Table 9.20.

9.2.8. Sensor: coded visual markers detector

This module describes the coded visual markers detector sensor. The reference frames involved in this module are shown in Fig. 9.8.

State and parameters

It uses only the common sensor state and parameters of Table 9.9.

Continuous-time true-value process model

It uses only the common sensor model. No extra model is needed.

	$\partial\delta\boldsymbol{a}^W_{R\|W}$	$\partial\delta\boldsymbol{\theta}^W_R$	$\partial\delta\boldsymbol{\omega}^W_{R\|W}$	$\partial\delta\boldsymbol{\alpha}^W_{R\|W}$
$\partial\delta\boldsymbol{z}_{accel}$	$\left[\frac{\partial\delta\boldsymbol{z}_{accel}}{\partial\delta\boldsymbol{a}^W_{R\|W}}\right]_{3\times3}$	$\left[\frac{\partial\delta\boldsymbol{z}_{accel}}{\partial\delta\boldsymbol{\theta}^W_R}\right]_{3\times3}$	$\left[\frac{\partial\delta\boldsymbol{z}_{accel}}{\partial\delta\boldsymbol{\omega}^W_{R\|W}}\right]_{3\times3}$	$\left[\frac{\partial\delta\boldsymbol{z}_{accel}}{\partial\delta\boldsymbol{\alpha}^W_{R\|W}}\right]_{3\times3}$

(a) Error-value vs. robot error-value.

	$\partial\delta\boldsymbol{g}^W_{\|W}$
$\partial\delta\boldsymbol{z}_{accel}$	$\left[\frac{\partial\delta\boldsymbol{z}_{accel}}{\partial\delta\boldsymbol{g}^W_{\|W}}\right]_{3\times3}$

(b) Error-value vs. acceleration of Gravity world error-value.

	$\partial\delta\boldsymbol{t}^R_{S_A}$	$\partial\delta\boldsymbol{\theta}^R_{S_A}$	$\partial\delta\boldsymbol{b}_{accel}$
$\partial\delta\boldsymbol{z}_{accel}$	$\left[\frac{\partial\delta\boldsymbol{z}_{accel}}{\partial\delta\boldsymbol{t}^R_{S_A}}\right]_{3\times3}$	$\left[\frac{\partial\delta\boldsymbol{z}_{accel}}{\partial\delta\boldsymbol{\theta}^R_{S_A}}\right]_{3\times3}$	$\boldsymbol{I}_{3\times3}$

(c) Error-value vs. sensor error-value.

	$\partial\boldsymbol{n}_{\delta accel}$
$\partial\delta\boldsymbol{z}_{accel}$	$\boldsymbol{I}_{3\times3}$

(d) Error-value vs. error-noise.

Table 9.20: Jacobian matrices of the accelerometer sensor error-value measurement model.

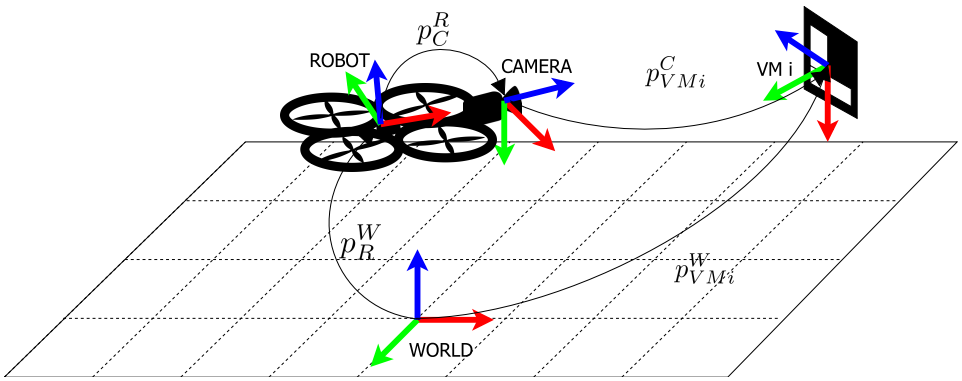


Figure 9.8: Camera and coded visual markers reference frames.

Discrete-time true-value process model

It uses only the common sensor model. No extra model is needed.

Jacobian matrices of the discrete-time error-value process model

It uses only the Jacobian matrices of the common sensor. No extra Jacobian matrices are needed.

Measurements

The measurements are presented in Table 9.21.

Magnitude		**True value**	**Nominal value**	**Error value**	**Composition** (local perturbation)
Measured pose of the visual marker	Position	$\boldsymbol{z}_{t-VMi}$	$\check{\boldsymbol{z}}_{t-VMi}$	$\delta\boldsymbol{z}_{t-VMi}$	$\delta\boldsymbol{z}_{t-VMi} = -\check{\boldsymbol{z}}_{t-VMi} + \boldsymbol{z}_{t-VMi}$
	Attitude(1)	$\boldsymbol{z}_{q-VMi}$	$\check{\boldsymbol{z}}_{q-VMi}$	$\delta\boldsymbol{z}_{q-VMi}$	$\delta\boldsymbol{z}_{q-VMi} = (\check{\boldsymbol{z}}_{q-VMi})^* \otimes \boldsymbol{z}_{q-VMi}$

(1) Being: $\delta\boldsymbol{z}_{q-VMi} \approx \begin{bmatrix} 1 \\ \frac{1}{2}\cdot\delta\boldsymbol{z}_{\theta-VMi} \end{bmatrix}$

Table 9.21: Measurements of the coded visual marker detector sensor module.

Additionally, the measurements depend on a reference frame world module that has the world information of the measured coded visual marker. If this world module does not exist, it has to be added to the estimator in the

mapping stage.

True-value measurement model

The true-value measurement model is described by the equations:

$$\boldsymbol{z}_{t-VMi} = \boldsymbol{t}_{VMi}^{S_C} + \boldsymbol{n}_{z_{t-VMi}} \tag{9.52}$$

$$\boldsymbol{z}_{q-VMi} = \boldsymbol{q}_{VMi}^{S_C} \otimes \boldsymbol{q}[\boldsymbol{n}_{z_{q-VMi}}] \tag{9.53}$$

where:

$$p_{VMi}^{S_C} = \ominus(p_R^W \oplus p_{S_C}^R) \oplus p_{VMi}^W \tag{9.54}$$

being:

$$\boldsymbol{t}_{VMi}^{S_C} = (\boldsymbol{q}_{S_C}^W)^* \otimes (\boldsymbol{t}_{VMi}^W - \boldsymbol{t}_{S_C}^W) \otimes \boldsymbol{q}_{S_C}^W \tag{9.55}$$

$$\boldsymbol{q}_{VMi}^{S_C} = (\boldsymbol{q}_{S_C}^W)^* \otimes \boldsymbol{q}_{VMi}^W \tag{9.56}$$

$\boldsymbol{n}_{z_{t-VMi}}$ and $\boldsymbol{n}_{z_{q-VMi}}$ are the noises of the true-value measurement model of the position and attitude of the coded visual marker detector sensor module, respectively. Note that the noise of the attitude measurement is defined as a local perturbation.

Jacobian matrices of the error-value measurement model

The existing Jacobian matrices of the error-value measurement model are indicated in Table 9.22.

	$\partial\delta\boldsymbol{t}_R^W$	$\partial\delta\boldsymbol{\theta}_R^W$
$\partial\delta\boldsymbol{z}_{t-VMi}$	$\left[\frac{\partial\delta\boldsymbol{z}_{t-VMi}}{\partial\delta\boldsymbol{t}_R^W}\right]_{3\times3}$	$\left[\frac{\partial\delta\boldsymbol{z}_{t-VMi}}{\partial\delta\boldsymbol{\theta}_R^W}\right]_{3\times3}$
$\partial\delta\boldsymbol{z}_{\theta-VMi}$	$\boldsymbol{0}$	$\left[\frac{\partial\delta\boldsymbol{z}_{\theta-VMi}}{\partial\delta\boldsymbol{\theta}_R^W}\right]_{3\times3}$

(a) Error-value vs. robot error-value.

	$\partial\delta\boldsymbol{t}_{S_C}^R$	$\partial\delta\boldsymbol{\theta}_{S_C}^R$
$\partial\delta\boldsymbol{z}_{t-VMi}$	$\left[\frac{\partial\delta\boldsymbol{z}_{t-VMi}}{\partial\delta\boldsymbol{t}_{S_C}^R}\right]_{3\times3}$	$\left[\frac{\partial\delta\boldsymbol{z}_{t-VMi}}{\partial\delta\boldsymbol{\theta}_{S_C}^R}\right]_{3\times3}$
$\partial\delta\boldsymbol{z}_{\theta-VMi}$	$\boldsymbol{0}$	$\left[\frac{\partial\delta\boldsymbol{z}_{\theta-VMi}}{\partial\delta\boldsymbol{\theta}_{S_C}^R}\right]_{3\times3}$

(b) Error-value vs. sensor error-value.

	$\partial\delta\boldsymbol{t}_{VMi}^W$	$\partial\delta\boldsymbol{\theta}_{VMi}^W$
$\partial\delta\boldsymbol{z}_{t-VMi}$	$\left[\frac{\partial\delta\boldsymbol{z}_{t-VMi}}{\partial\delta\boldsymbol{t}_{VMi}^W}\right]_{3\times3}$	$\boldsymbol{0}$
$\partial\delta\boldsymbol{z}_{\theta-VMi}$	$\boldsymbol{0}$	$\left[\frac{\partial\delta\boldsymbol{z}_{\theta-VMi}}{\partial\delta\boldsymbol{\theta}_{VMi}^W}\right]_{3\times3}$

(c) Error-value vs. coded visual marker reference frame world error-value.

	$\partial\boldsymbol{n}_{\delta t-VMi}$	$\partial\boldsymbol{n}_{\delta\theta-VMi}$
$\partial\delta\boldsymbol{z}_{t-VMi}$	$\boldsymbol{I}_{3\times3}$	$\boldsymbol{0}$
$\partial\delta\boldsymbol{z}_{\theta-VMi}$	$\boldsymbol{0}$	$\left[\frac{\partial\delta\boldsymbol{z}_{\theta-VMi}}{\partial\boldsymbol{n}_{\delta\theta-VMi}}\right]_{3\times3}$

(d) Error-value vs. error-noise.

Table 9.22: Jacobian matrices of the coded visual marker detector sensor error-value measurement model.

True-value mapping model

The true-value mapping model is described by the equations:

$$p_{VMi}^W = p_R^W \oplus p_{S_C}^R \oplus p_{VMi}^{S_C} \tag{9.57}$$

Being:

$$\begin{aligned} \boldsymbol{t}_{VMi}^{W} =& \boldsymbol{q}_{R}^{W} \otimes \boldsymbol{t}_{VMi}^{R} \otimes \left(\boldsymbol{q}_{R}^{W}\right)^{*} + \boldsymbol{t}_{R}^{W} \\ =& \boldsymbol{q}_{R}^{W} \otimes \left(\boldsymbol{q}_{S_C}^{R} \otimes \boldsymbol{q}_{VMi}^{S_C} \otimes (\boldsymbol{q}_{S_C}^{R})^{*} + \boldsymbol{t}_{S_C}^{R}\right) \otimes \left(\boldsymbol{q}_{R}^{W}\right)^{*} + \boldsymbol{t}_{R}^{W} \end{aligned} \tag{9.58}$$

$$\boldsymbol{q}_{VMi}^{W} = \boldsymbol{q}_{R}^{W} \otimes \boldsymbol{q}_{S_C}^{R} \otimes \boldsymbol{q}_{VMi}^{S_C} \tag{9.59}$$

It is important to note that the complete measurement (position and attitude) is required, to be able to perform the mapping of the world module.

Jacobian matrices of the error-value mapping model

The existing Jacobian matrices of the error-value mapping model are indicated in Table 9.23.

	$\partial\delta\boldsymbol{t}_{R}^{W}$	$\partial\delta\boldsymbol{\theta}_{R}^{W}$
$\partial\delta\boldsymbol{t}_{VMi}^{W}$	$\boldsymbol{I}_{3\times3}$	$\left[\frac{\partial\delta\boldsymbol{t}_{VMi}^{W}}{\partial\delta\boldsymbol{\theta}_{R}^{W}}\right]_{3\times3}$
$\partial\delta\boldsymbol{\theta}_{VMi}^{W}$	$\boldsymbol{0}$	$\left[\frac{\partial\delta\boldsymbol{\theta}_{VMi}^{W}}{\partial\delta\boldsymbol{\theta}_{R}^{W}}\right]_{3\times3}$

(a) Error-value vs. robot error-value.

	$\partial\delta\boldsymbol{t}_{S_C}^{R}$	$\partial\delta\boldsymbol{\theta}_{S_C}^{R}$
$\partial\delta\boldsymbol{t}_{VMi}^{W}$	$\left[\frac{\partial\delta\boldsymbol{t}_{VMi}^{W}}{\partial\delta\boldsymbol{t}_{S_C}^{R}}\right]_{3\times3}$	$\left[\frac{\partial\delta\boldsymbol{t}_{VMi}^{W}}{\partial\delta\boldsymbol{\theta}_{S_C}^{R}}\right]_{3\times3}$
$\partial\delta\boldsymbol{\theta}_{VMi}^{W}$	$\boldsymbol{0}$	$\left[\frac{\partial\delta\boldsymbol{\theta}_{VMi}^{W}}{\partial\delta\boldsymbol{\theta}_{S_C}^{R}}\right]_{3\times3}$

(b) Error-value vs. sensor error-value.

	$\partial\delta\boldsymbol{t}_{VMi}^{S_C}$	$\partial\delta\boldsymbol{\theta}_{VMi}^{S_C}$
$\partial\delta\boldsymbol{t}_{VMi}^{W}$	$\left[\frac{\partial\delta\boldsymbol{t}_{VMi}^{W}}{\partial\delta\boldsymbol{t}_{VMi}^{S_C}}\right]_{3\times3}$	$\boldsymbol{0}$
$\partial\delta\boldsymbol{\theta}_{VMi}^{W}$	$\boldsymbol{0}$	$\left[\frac{\partial\delta\boldsymbol{\theta}_{VMi}^{W}}{\partial\delta\boldsymbol{\theta}_{VMi}^{S_C}}\right]_{3\times3}$

(c) Error-value vs. sensor measurement error-value.

Table 9.23: Jacobian matrices of the coded visual marker detector sensor error-value mapping model.

9.2.9. Sensor: horizontal linear velocity

This module describes the horizontal linear velocity sensor, as for example the px4flow sensor, (Honegger et al., 2013).

State and parameters

It uses only the common sensor state and parameters of Table 9.9.

Continuous-time true-value process model

It uses only the common sensor model. No extra model is needed.

Discrete-time true-value process model

It uses only the common sensor model. No extra model is needed.

Jacobian matrices of the discrete-time error-value process model

It uses only the Jacobian matrices of the common sensor. No extra Jacobian matrices are needed.

Measurements

The measurements are presented in Table 9.24.

Magnitude	**True value**	**Nominal value**	**Error value**	**Composition** (local perturbation)
Measured horizontal linear velocity	$\boldsymbol{z}_v$	$\check{\boldsymbol{z}}_v$	$\delta\boldsymbol{z}_v$	$\delta\boldsymbol{z}_v = -\check{\boldsymbol{z}}_v + \boldsymbol{z}_v$

Table 9.24: Measurements of the horizontal linear velocity sensor module.

True-value measurement model

The true-value measurement model is described by the equations:

$$\boldsymbol{z}_v = \boldsymbol{S}_{v-sens} \cdot \boldsymbol{v}_{S|W}^S + \boldsymbol{n}_{z_v} \tag{9.60}$$

being the sensitivity matrix in the speed measurement:

$$\boldsymbol{S}_{v-sens} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{9.61}$$

and:

$$\boldsymbol{v}_{S|W}^S = \left(\boldsymbol{q}_S^W\right)^* \otimes \boldsymbol{v}_{S|W}^W \otimes \boldsymbol{q}_S^W \tag{9.62}$$

$$\boldsymbol{v}_{S|W}^W = \boldsymbol{v}_{R|W}^W + \boldsymbol{q}_R^W \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{R|W}^R \times \boldsymbol{t}_S^R \end{bmatrix} \otimes \left(\boldsymbol{q}_R^W\right)^* \tag{9.63}$$

$$\boldsymbol{\omega}_{R|W}^R = \left(\boldsymbol{q}_R^W\right)^* \otimes \boldsymbol{\omega}_{R|W}^W \otimes \boldsymbol{q}_R^W \tag{9.64}$$

$\boldsymbol{n}_{z_v}$ is the noise of the true-value measurement model of the horizontal linear velocity sensor module.

Jacobian matrices of the error-value measurement model

The existing Jacobian matrices of the error-value measurement model are indicated in Table 9.25.

	$\partial\delta\boldsymbol{v}_{R\|W}^W$	$\partial\delta\boldsymbol{\theta}_R^W$	$\partial\delta\boldsymbol{\omega}_{R\|W}^W$
$\partial\delta\boldsymbol{z}_v$	$\left[\frac{\partial\delta\boldsymbol{z}_v}{\partial\delta\boldsymbol{v}_{R\|W}^W}\right]_{2\times3}$	$\left[\frac{\partial\delta\boldsymbol{z}_v}{\partial\delta\boldsymbol{\theta}_R^W}\right]_{2\times3}$	$\left[\frac{\partial\delta\boldsymbol{z}_v}{\partial\delta\boldsymbol{\omega}_{R\|W}^W}\right]_{2\times3}$

(a) Error-value vs. robot error-value.

	$\partial\delta\boldsymbol{t}_S^R$	$\partial\delta\boldsymbol{\theta}_S^R$
$\partial\delta\boldsymbol{z}_v$	$\left[\frac{\partial\delta\boldsymbol{z}_v}{\partial\delta\boldsymbol{t}_S^R}\right]_{2\times3}$	$\left[\frac{\partial\delta\boldsymbol{z}_v}{\partial\delta\boldsymbol{\theta}_S^R}\right]_{2\times3}$

(b) Error-value vs. sensor error-value.

	$\partial\delta\boldsymbol{n}_{z_v}$
$\partial\delta\boldsymbol{z}_v$	$\boldsymbol{I}_{2\times2}$

(c) Error-value vs. error-noise.

Table 9.25: Jacobian matrices of the horizontal linear velocity sensor error-value measurement model.

9.2.10. Sensor: vertical distance

This module describes the vertical distance sensor, as for example an ultrasound sensor or a 1D LIDAR.

State and parameters

It uses only the common sensor state and parameters of Table 9.9.

Continuous-time true-value process model

It uses only the common sensor model. No extra model is needed.

Discrete-time true-value process model

It uses only the common sensor model. No extra model is needed.

Jacobian matrices of the discrete-time error-value process model

It uses only the Jacobian matrices of the common sensor. No extra Jacobian matrices are needed.

Measurement

The measurements are presented in Table 9.26.

Magnitude	**True value**	**Nominal value**	**Error value**	**Composition** (local perturbation)
Measured vertical distance	z_d	$\check{z}_d$	δz_d	$\delta z_d = -\check{z}_d + z_d$

Table 9.26: Measurements of the vertical distance sensor module.

Additionally, the measurements depend on a plane world module that has the world information of the ground plane.

True-value measurement model

The true-value measurement model is described by the equations:

$$z_d = d_{P-S} + n_{z_d} \tag{9.65}$$

and being:

$$d_{P-S} = \left| \frac{\boldsymbol{n}_P^W \circ \left(\boldsymbol{x}_P^W - \boldsymbol{t}_S^W\right)}{\boldsymbol{n}_P^W \circ \boldsymbol{u}} \right| \tag{9.66}$$

where:

$$\boldsymbol{t}_S^W = \boldsymbol{q}_R^W \otimes \boldsymbol{t}_S^R \otimes \left(\boldsymbol{q}_R^W\right)^* + \boldsymbol{t}_R^W \tag{9.67}$$

$$\boldsymbol{u} = \boldsymbol{q}_S^W \otimes \boldsymbol{s}_{d-sens} \otimes \left(\boldsymbol{q}_S^W\right)^* \tag{9.68}$$

being the sensitivity vector in the distance measurement:

$$\boldsymbol{s}_{d-sens} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{9.69}$$

n_{z_d} is the noise of the true-value measurement model of the vertical distance sensor module.

Jacobian matrices of the error-value measurement model

The existing Jacobian matrices of the error-value measurement model are indicated in Table 9.27.

9.2.11. Sensor: absolute pose

This module describes the absolute pose sensor, such as a Motion Capture System (MoCap), a 3D LIDAR SLAM, a Stereo Visual SLAM or a Global Navigation Satellite System (GNSS). The reference frames involved in this module are shown in Fig. 9.9.

State and parameters

It uses only the common sensor state and parameters of Table 9.9.

Continuous-time true-value process model

It uses only the common sensor model. No extra model is needed.

Discrete-time true-value process model

It uses only the common sensor model. No extra model is needed.

	$\partial\delta t_R^W$	$\partial\delta\theta_R^W$
$\partial\delta z_d$	$\left[\frac{\partial\delta z_d}{\partial\delta t_S^R}\right]_{1\times3}$	$\left[\frac{\partial\delta z_d}{\partial\delta\theta_R^W}\right]_{1\times3}$

(a) Error-value vs. robot error-value.

	$\partial\delta t_S^R$	$\partial\delta\theta_S^R$
$\partial\delta z_d$	$\left[\frac{\partial\delta z_d}{\partial\delta t_S^R}\right]_{1\times3}$	$\left[\frac{\partial\delta z_d}{\partial\delta\theta_S^R}\right]_{1\times3}$

(b) Error-value vs. sensor error-value.

	$\partial\delta n_P^W$	$\partial\delta x_P^W$
$\partial\delta z_d$	$\left[\frac{\partial\delta z_d}{\partial\delta n_P^W}\right]_{1\times3}$	$\left[\frac{\partial\delta z_d}{\partial\delta x_P^W}\right]_{1\times3}$

(c) Error-value vs. ground plane world error-value.

	$\partial\delta n_{z_d}$
$\partial\delta z_d$	1

(d) Error-value vs. error-noise.

Table 9.27: Jacobian matrices of the horizontal linear velocity sensor error-value measurement model.

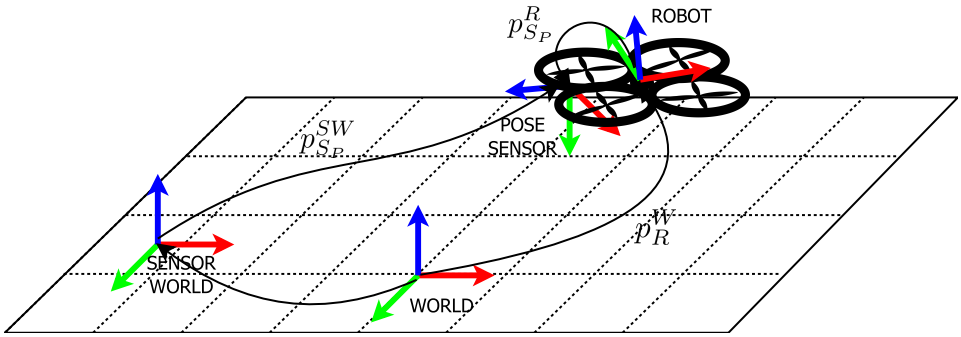


Figure 9.9: Absolute pose reference frames.

Jacobian matrices of the discrete-time error-value process model

It uses only the Jacobian matrices of the common sensor. No extra Jacobian matrices are needed.

Measurements

The measurements are presented in Table 9.28.

Magnitude		**True value**	**Nominal value**	**Error value**	**Composition** (local perturbation)
Pose measurement given by the sensor	Position	z_t	$\check{z}_t$	δz_t	$\delta z_t = -\check{z}_t + z_t$
	Attitude(1)	z_q	$\check{z}_q$	δz_q	$\delta z_q = (\check{z}_q)^* \otimes z_q$

(1) Being: $\delta z_q \approx \begin{bmatrix} 1 \\ \frac{1}{2}\cdot\delta z_\theta \end{bmatrix}$

Table 9.28: Measurements of the absolute pose sensor module.

Additionally, the measurements depend on a reference frame world module that has the information of the world reference frame of the measurement given by the sensor. If this world module does not exist, it has to be added to the estimator in the mapping stage.

True-value measurement model

The true-value measurement model is described by the equations:

$$\boldsymbol{z}_t = \boldsymbol{t}_{S_P}^{SW} + \boldsymbol{n}_{z_t} \tag{9.70}$$

$$\boldsymbol{z}_q = \boldsymbol{q}_{S_P}^{SW} \otimes \boldsymbol{q}\left\{\boldsymbol{n}_{z_q}\right\} \tag{9.71}$$

being:

$$p_{S_P}^{SW} = \ominus p_{SW}^{W} \oplus p_R^W \oplus p_{S_P}^R \tag{9.72}$$

where:

$$\boldsymbol{t}_{S_P}^{SW} = \left(\boldsymbol{q}_{SW}^{W}\right)^* \otimes \left(\boldsymbol{t}_{S_P}^{W} - \boldsymbol{t}_{SW}^{W}\right) \otimes \boldsymbol{q}_{SW}^{W} \tag{9.73}$$

$$\boldsymbol{q}_{S_P}^{SW} = \left(\boldsymbol{q}_{SW}^{W}\right)^* \otimes \boldsymbol{q}_R^W \otimes \boldsymbol{q}_{S_P}^R \tag{9.74}$$

$\boldsymbol{n}_{z_t}$ and $\boldsymbol{n}_{z_q}$ are the noises of the true-value measurement model of the position and attitude of the absolute pose sensor module, respectively. Note that the noise of the attitude measurement is defined as a local perturbation.

Jacobian matrices of the error-value measurement model

The existing Jacobian matrices of the error-value measurement model are indicated in Table 9.29.

	$\partial\delta\boldsymbol{t}_R^W$	$\partial\delta\boldsymbol{\theta}_R^W$
$\partial\delta\boldsymbol{z}_t$	$\left[\frac{\partial\delta\boldsymbol{z}_t}{\partial\delta\boldsymbol{t}_R^W}\right]_{3\times3}$	$\left[\frac{\partial\delta\boldsymbol{z}_t}{\partial\delta\boldsymbol{\theta}_R^W}\right]_{3\times3}$
$\partial\delta\boldsymbol{z}_q$	$\mathbf{0}$	$\left[\frac{\partial\delta\boldsymbol{z}_q}{\partial\delta\boldsymbol{\theta}_R^W}\right]_{3\times3}$

(a) Error-value vs. robot error-value.

	$\partial\delta\boldsymbol{t}_{S_P}^R$	$\partial\delta\boldsymbol{\theta}_{S_P}^R$
$\partial\delta\boldsymbol{z}_t$	$\left[\frac{\partial\delta\boldsymbol{z}_t}{\partial\delta\boldsymbol{t}_{S_P}^R}\right]_{3\times3}$	$\mathbf{0}$
$\partial\delta\boldsymbol{z}_q$	$\mathbf{0}$	$\left[\frac{\partial\delta\boldsymbol{z}_q}{\partial\delta\boldsymbol{\theta}_{S_P}^R}\right]_{3\times3}$

(b) Error-value vs. sensor error-value.

	$\partial\delta\boldsymbol{t}_{SW}^W$	$\partial\delta\boldsymbol{\theta}_{SW}^W$
$\partial\delta\boldsymbol{z}_t$	$\left[\frac{\partial\delta\boldsymbol{z}_t}{\partial\delta\boldsymbol{t}_{SW}^W}\right]_{3\times3}$	$\left[\frac{\partial\delta\boldsymbol{z}_t}{\partial\delta\boldsymbol{\theta}_{SW}^W}\right]_{3\times3}$
$\partial\delta\boldsymbol{z}_q$	$\mathbf{0}$	$\left[\frac{\partial\delta\boldsymbol{z}_q}{\partial\delta\boldsymbol{\theta}_{SW}^W}\right]_{3\times3}$

(c) Error-value vs. reference frame world error-value.

	$\partial\boldsymbol{n}_{\delta t}$	$\partial\boldsymbol{n}_{\delta\theta}$
$\partial\delta\boldsymbol{z}_t$	$\boldsymbol{I}_{3\times3}$	$\mathbf{0}$
$\partial\delta\boldsymbol{z}_\theta$	$\mathbf{0}$	$\left[\frac{\partial\delta\boldsymbol{z}_\theta}{\partial\boldsymbol{n}_{\delta\theta}}\right]_{3\times3}$

(d) Error-value vs. error-noise.

Table 9.29: Jacobian matrices of the absolute pose sensor error-value measurement model.

True-value mapping model

The true-value mapping model is described by the equations:

$$p_{SW}^W = p_R^W \oplus p_{S_P}^R \ominus p_{S_P}^{SW} \tag{9.75}$$

And considering Eq. 9.70 and 9.71 is:

$$\begin{aligned}\boldsymbol{t}_{SW}^W = &-\boldsymbol{q}_{S_P}^W \otimes \left(\left(\boldsymbol{q}_{S_P}^{SW}\right)^* \otimes \boldsymbol{t}_{S_P}^{SW} \otimes \boldsymbol{q}_{S_P}^{SW}\right) \otimes \left(\boldsymbol{q}_{S_P}^W\right)^* \\ &+ \boldsymbol{q}_R^W \otimes \boldsymbol{t}_{S_P}^R \otimes \left(\boldsymbol{q}_R^W\right)^* + \boldsymbol{t}_R^W\end{aligned} \tag{9.76}$$

$$\boldsymbol{q}_{SW}^W = \boldsymbol{q}_R^W \otimes \boldsymbol{q}_{S_P}^R \otimes \left(\boldsymbol{q}_{S_P}^{SW}\right)^* \tag{9.77}$$

It is important to note that the complete measurement (position and attitude) is needed to be able to do the mapping of the world module.

Jacobian matrices of the error-value mapping model

The existing Jacobian matrices of the error-value mapping model are indicated in Table 9.30.

	$\partial \delta \boldsymbol{t}_R^W$	$\partial \delta \boldsymbol{\theta}_R^W$
$\partial \delta \boldsymbol{t}_{SW}^W$	$\left[\frac{\partial \delta \boldsymbol{t}_{SW}^W}{\partial \delta \boldsymbol{t}_R^W}\right]_{3\times3}$	$\left[\frac{\partial \delta \boldsymbol{t}_{SW}^W}{\partial \delta \boldsymbol{\theta}_R^W}\right]_{3\times3}$
$\partial \delta \boldsymbol{\theta}_{SW}^W$	$\mathbf{0}$	$\left[\frac{\partial \delta \boldsymbol{\theta}_{SW}^W}{\partial \delta \boldsymbol{\theta}_R^W}\right]_{3\times3}$

(a) Error-value vs. robot error-value.

	$\partial \delta \boldsymbol{t}_{S_P}^R$	$\partial \delta \boldsymbol{\theta}_{S_P}^R$
$\partial \delta \boldsymbol{t}_{SW}^W$	$\left[\frac{\partial \delta \boldsymbol{t}_{SW}^W}{\partial \delta \boldsymbol{t}_{S_P}^R}\right]_{3\times3}$	$\left[\frac{\partial \delta \boldsymbol{t}_{SW}^W}{\partial \delta \boldsymbol{\theta}_{S_P}^R}\right]_{3\times3}$
$\partial \delta \boldsymbol{\theta}_{SW}^W$	$\mathbf{0}$	$\left[\frac{\partial \delta \boldsymbol{\theta}_{SW}^W}{\partial \delta \boldsymbol{\theta}_{S_P}^R}\right]_{3\times3}$

(b) Error-value vs. sensor error-value.

	$\partial \delta \boldsymbol{t}_{S_P}^{SW}$	$\partial \delta \boldsymbol{\theta}_{S_P}^{SW}$
$\partial \delta \boldsymbol{t}_{SW}^W$	$\left[\frac{\partial \delta \boldsymbol{t}_{SW}^W}{\partial \delta \boldsymbol{t}_{S_P}^{SW}}\right]_{3\times3}$	$\left[\frac{\partial \delta \boldsymbol{t}_{SW}^W}{\partial \delta \boldsymbol{\theta}_{S_P}^{SW}}\right]_{3\times3}$
$\partial \delta \boldsymbol{\theta}_{SW}^W$	$\mathbf{0}$	$\left[\frac{\partial \delta \boldsymbol{\theta}_{SW}^W}{\partial \delta \boldsymbol{\theta}_{S_P}^{SW}}\right]_{3\times3}$

(c) Error-value vs. sensor measurement error-value.

Table 9.30: Jacobian matrices of the absolute pose sensor error-value mapping model.

9.2.12. Sensor: unscaled absolute pose

This module describes the unscaled absolute pose sensor, like a monocular visual SLAM. The reference frames involved in this module are similar to those shown in Fig. 9.9.

State and parameters

It uses the common sensor state and parameters of Table 9.9 and the ones presented in Table 9.31.

Magnitude	**True value**	**Nominal value**	**Error value**	**Composition** (local perturbation)
Scale on the measurement	λ_{meas}	$\check{\lambda}_{meas}$	$\delta\lambda_{meas}$	$\delta\lambda_{meas} = -\check{\lambda}_{meas} + \lambda_{meas}$

Table 9.31: State and parameters of the unscaled absolute pose sensor module.

Continuous-time true-value process model

Assuming that the scale drifts spatially and not temporary, the continuous-time true-value process model is described by the common sensor model together with the equations:

$$\dot{\lambda}_{meas} = 0 \tag{9.78}$$

Discrete-time true-value process model

The discrete-time true-value process model is described by the common sensor together with the equations:

$$\lambda_{meas}(k) = \lambda_{meas}(k-1) \tag{9.79}$$

Jacobian matrices of the discrete-time error-value process model

The existing Jacobian matrices of the discrete-time error-value process model are the ones shown in the common sensor together with the ones indicated in Table 9.32.

k \ $k-1$	$\partial\delta\boldsymbol{\lambda}_{meas}$
$\partial\delta\boldsymbol{\lambda}_{meas}$	1

(a) Error-value in k vs. error-value in $k-1$.

Table 9.32: Jacobian matrices of the accelerometer sensor error-value discrete-time process model.

Measurements

The measurements are presented in Table 9.33.

Magnitude		**True value**	**Nominal value**	**Error value**	**Composition** (local perturbation)
Unscaled pose measurement given by the sensor	Position	$\boldsymbol{z}_t$	$\check{\boldsymbol{z}}_t$	$\delta\boldsymbol{z}_t$	$\delta\boldsymbol{z}_t = -\check{\boldsymbol{z}}_t + \boldsymbol{z}_t$
	Attitude(1)	$\boldsymbol{z}_q$	$\check{\boldsymbol{z}}_q$	$\delta\boldsymbol{z}_q$	$\delta\boldsymbol{z}_q = (\check{\boldsymbol{z}}_q)^* \otimes \boldsymbol{z}_q$

(1) Being: $\delta z_q \approx \begin{bmatrix} 1 \\ \frac{1}{2}\cdot\delta z_\theta \end{bmatrix}$

Table 9.33: Measurements of the unscaled absolute pose sensor module.

True-value measurement model

The true-value measurement model is described by the equations:

$$\boldsymbol{z}_t = \boldsymbol{\lambda}_{meas}\cdot\boldsymbol{t}_{S_P}^{SW} + \boldsymbol{n}_{z_t} \tag{9.80}$$

$$\boldsymbol{z}_q = \boldsymbol{q}_{S_P}^{SW} \otimes \boldsymbol{q}\left\{\boldsymbol{n}_{z_q}\right\} \tag{9.81}$$

being:

$$p_{S_P}^{SW} = \ominus p_{SW}^{W} \oplus p_R^W \oplus p_{S_P}^R \tag{9.82}$$

where:

$$\boldsymbol{t}_{S_P}^{SW} = \left(\boldsymbol{q}_{SW}^{W}\right)^* \otimes \left(\boldsymbol{t}_{S_P}^{W} - \boldsymbol{t}_{SW}^{W}\right) \otimes \boldsymbol{q}_{SW}^{W} \tag{9.83}$$

$$\boldsymbol{q}_{S_P}^{SW} = \left(\boldsymbol{q}_{SW}^{W}\right)^* \otimes \boldsymbol{q}_R^W \otimes \boldsymbol{q}_{S_P}^R \tag{9.84}$$

$\boldsymbol{n}_{z_t}$ and $\boldsymbol{n}_{z_q}$ are the noises of the true-value measurement model of the position and attitude of the unscaled absolute pose sensor module, respectively. Note that the noise of the attitude measurement is defined as a local perturbation.

Jacobian matrices of the error-value measurement model

The existing Jacobian matrices of the error-value measurement model are indicated in Table 9.34.

True-value mapping model

The true-value mapping model is described by the equations:

$$p_{SW}^{W} = p_R^W \oplus p_{S_P}^R \ominus p_{S_P}^{SW} \tag{9.85}$$

And considering Eq. 9.80 and 9.81 is:

$$\begin{aligned}\boldsymbol{t}_{SW}^{W} =& -\boldsymbol{q}_{S_P}^{W} \otimes \left(\left(\boldsymbol{q}_{S_P}^{SW}\right)^* \otimes \boldsymbol{t}_{S_P}^{SW} \otimes \boldsymbol{q}_{S_P}^{SW}\right) \otimes \left(\boldsymbol{q}_{S_P}^{W}\right)^* \\ &+ \boldsymbol{q}_R^W \otimes \boldsymbol{t}_{S_P}^R \otimes \left(\boldsymbol{q}_R^W\right)^* + \boldsymbol{t}_R^W \end{aligned} \tag{9.86}$$

$$\boldsymbol{q}_{SW}^{W} = \boldsymbol{q}_R^W \otimes \boldsymbol{q}_{S_P}^R \otimes \left(\boldsymbol{q}_{S_P}^{SW}\right)^* \tag{9.87}$$

Jacobian matrices of the error-value mapping model

The existing Jacobian matrices of the error-value mapping model are indicated in Table 9.35.

	$\partial\delta t_R^W$	$\partial\delta\theta_R^W$
$\partial\delta z_t$	$\left[\frac{\partial\delta z_t}{\partial\delta t_R^W}\right]_{3\times3}$	$\left[\frac{\partial\delta z_t}{\partial\delta\theta_R^W}\right]_{3\times3}$
$\partial\delta z_q$	$\mathbf{0}$	$\left[\frac{\partial\delta z_q}{\partial\delta\theta_R^W}\right]_{3\times3}$

(a) Error-value vs. robot error-value.

	$\partial\delta t_{S_P}^R$	$\partial\delta\theta_{S_P}^R$	$\partial\delta\lambda_{meas}$
$\partial\delta z_t$	$\left[\frac{\partial\delta z_t}{\partial\delta t_{S_P}^R}\right]_{3\times3}$	$\mathbf{0}$	$\left[\frac{\partial\delta z_t}{\partial\delta\lambda_{meas}}\right]_{3\times1}$
$\partial\delta z_q$	$\mathbf{0}$	$\left[\frac{\partial\delta z_q}{\partial\delta\theta_{S_P}^R}\right]_{3\times3}$	$\mathbf{0}$

(b) Error-value vs. sensor error-value.

	$\partial\delta t_{SW}^W$	$\partial\delta\theta_{SW}^W$
$\partial\delta z_t$	$\left[\frac{\partial\delta z_t}{\partial\delta t_{SW}^W}\right]_{3\times3}$	$\left[\frac{\partial\delta z_t}{\partial\delta\theta_{SW}^W}\right]_{3\times3}$
$\partial\delta z_q$	$\mathbf{0}$	$\left[\frac{\partial\delta z_q}{\partial\delta\theta_{SW}^W}\right]_{3\times3}$

(c) Error-value vs. reference frame world error-value.

	$\partial n_{\delta t}$	$\partial n_{\delta\theta}$
$\partial\delta z_t$	$I_{3\times3}$	$\mathbf{0}$
$\partial\delta z_\theta$	$\mathbf{0}$	$\left[\frac{\partial\delta z_\theta}{\partial n_{\delta\theta}}\right]_{3\times3}$

(d) Error-value vs. error-noise.

Table 9.34: Jacobian matrices of the absolute pose sensor error-value measurement model.

	$\partial\delta t_R^W$	$\partial\delta\theta_R^W$
$\partial\delta t_{SW}^W$	$\left[\frac{\partial\delta t_{SW}^W}{\partial\delta t_R^W}\right]_{3\times3}$	$\left[\frac{\partial\delta t_{SW}^W}{\partial\delta\theta_R^W}\right]_{3\times3}$
$\partial\delta\theta_{SW}^W$	$\mathbf{0}$	$\left[\frac{\partial\delta\theta_{SW}^W}{\partial\delta\theta_R^W}\right]_{3\times3}$

(a) Error-value vs. robot error-value.

	$\partial\delta t_{S_P}^R$	$\partial\delta\theta_{S_P}^R$	$\partial\delta\lambda_{meas}$
$\partial\delta t_{SW}^W$	$\left[\frac{\partial\delta t_{SW}^W}{\partial\delta t_{S_P}^R}\right]_{3\times3}$	$\left[\frac{\partial\delta t_{SW}^W}{\partial\delta\theta_{S_P}^R}\right]_{3\times3}$	$\left[\frac{\partial\delta t_{SW}^W}{\partial\delta\lambda_{meas}}\right]_{3\times1}$
$\partial\delta\theta_{SW}^W$	$\mathbf{0}$	$\left[\frac{\partial\delta\theta_{SW}^W}{\partial\delta\theta_{S_P}^R}\right]_{3\times3}$	$\mathbf{0}$

(b) Error-value vs. sensor error-value.

	$\partial\delta t_{S_P}^{SW}$	$\partial\delta\theta_{S_P}^{SW}$
$\partial\delta t_{SW}^W$	$\left[\frac{\partial\delta t_{SW}^W}{\partial\delta t_{S_P}^{SW}}\right]_{3\times3}$	$\left[\frac{\partial\delta t_{SW}^W}{\partial\delta\theta_{S_P}^{SW}}\right]_{3\times3}$
$\partial\delta\theta_{SW}^W$	$\mathbf{0}$	$\left[\frac{\partial\delta\theta_{SW}^W}{\partial\delta\theta_{S_P}^{SW}}\right]_{3\times3}$

(c) Error-value vs. sensor measurement error-value.

Table 9.35: Jacobian matrices of the unscaled absolute pose sensor error-value mapping model.

9.3. Environment reconstruction

Similarly to the environment reconstruction component described in Section 9.3, this component generates a map of the environment with a usable format formed by the geometric primitives of the objects of the environment. To generate these geometric primitives of the objects, it uses the map of landmarks estimated by the MSF state estimation component, previously analyzed, and an a priori knowledge describing some parameters of the objects and the relationships between the objects and the landmarks.

9.3.1. 3D geometric primitives

This component generates 3D geometric primitives (cubes, cylinders, and spheres) of the objects of the environment. It estimates the pose of these primitives taking into account the estimated pose of the mapped landmarks.

The following a priori knowledge is required:

- kind of 3D object (cube, cylinder or sphere);
- parameters of the 3D object (sizes);
- pose of the landmarks with respect to the center of the 3D object $p^{CO\star}_{VMi}$.

The pose of the center of the 3D object in world coordinates, $p^{W}_{CO\star}$, is estimated performing the average of the individual contribution of every landmark as:

$$\boldsymbol{t}^{W}_{CO\star} = \sum_{i=1}^{n} \left(w_i \cdot \boldsymbol{t}^{W}_{CO\star}(i) \right) \tag{9.88}$$

$$\boldsymbol{q}^{W}_{CO\star} = \boldsymbol{Q}_{av} \cdot \boldsymbol{Q}^{T}_{av} \tag{9.89}$$

where:

$$[\boldsymbol{Q}_{av}]_{4\times n} = \begin{bmatrix} w_1 \cdot \boldsymbol{q}^{W}_{CO\star}(1) & w_2 \cdot \boldsymbol{q}^{W}_{CO\star}(2) & \dots & w_n \cdot \boldsymbol{q}^{W}_{CO\star}(n) \end{bmatrix} \tag{9.90}$$

being $w_i = 1/n$ and n is the number of mapped landmarks associated to the 3D object.

The contribution of the landmark i to the estimation of the pose of the center of the 3D object in world coordinates, $p^{W}_{CO\star}(i)$, is calculated using the estimation of the pose of the landmark i in world coordinates, p^{W}_{VMi}, following the equation:

$$p^{W}_{CO\star}(i) = p^{W}_{VMi} \ominus p^{CO\star}_{VMi} \tag{9.91}$$

and therefore:

$$\boldsymbol{t}^{W}_{CO\star}(i) = -\boldsymbol{q}^{W}_{VMi} \otimes \left(\boldsymbol{q}^{CO\star}_{VMi}\right)^{*} \otimes \boldsymbol{t}^{CO\star}_{VMi} \otimes \boldsymbol{q}^{CO\star}_{VMi} \otimes \left(\boldsymbol{q}^{W}_{VMi}\right)^{*} + \boldsymbol{t}^{W}_{VMi} \tag{9.92}$$

$$\boldsymbol{q}^{W}_{CO\star}(i) = \boldsymbol{q}^{W}_{VMi} \otimes \left(\boldsymbol{q}^{CO\star}_{VMi}\right)^{*} \tag{9.93}$$

A future work to improve the estimation of the pose of the center of the 3D object, the estimated covariance of the landmark i might be used when calculating the weight w_i.

9.3.2. 2D geometric primitives

This component generates 2D geometric primitives (rectangles and ellipses), based on the 3D geometrical primitives calculated in the previous section. To provide a similar output than the environment reconstruction component described in Section 9.3, only rectangles, and ellipses are used as possible 2D geometrical primitives.

These 2D geometric primitives are calculated by following Algorithm 1.

Algorithm 1 2D geometric primitives generation algorithm.

```
for All 3D geometric primitives do
    Calculate intersection between the 3D geometric primitives and the plane for the 2D representation
    if Intersection is a point or a line then
        No 2D primitive is generated
    else if Intersection is an ellipse then
        The 2D primitive is directly the intersected ellipse
    else if Intersection is a polygon then
        The 2D primitive is the rectangle that circumscribe the intersected polygon
    end if
end for
```

9.4. Evaluation and results

9.4.1. Methodology

The proposed perception solution, incorporated as several open-source components in Aerostack, is tested through real experiments controlling a quadrotor performing a semi-autonomous flight. A semi-autonomous flight, using the minimum number of components of Aerostack, is preferred instead of a fully autonomous mission, with the objective of isolating the performance of the proposed perception solution and simplifying the experiments without losing realism.

Additionally, in the experiments presented bellow, a basic set of hardware elements were employed, using only some of the available modules presented in Section 9.2. This simplification allows to efficiently analyze the performance of the proposed complete solution, and their main features (presented in Section 9.1), avoiding a higher complexity in the experiments.

9.4.2. System setup

As shown in Fig. 9.10, a Mikrokopter[2] quadrotor is used as the aerial platform. Its Flight Controller board is only used to acquire its embedded IMU measurements at 1 [kHz] and as the interface with its motor controllers. The quadrotor is equipped only with an extra IMU Phidgets 1044[3] (250 Hz) and a UI-3241-LE-C-HQ camera with a wide angle lens (set to 640×480 @ 30 Hz). An Intel NUC5i7RYH[4] is mounted onboard in which all the software runs. Additionally, a desktop computer is used as the Ground Control Station (GCS) for user visualization and mission commanding purposes.

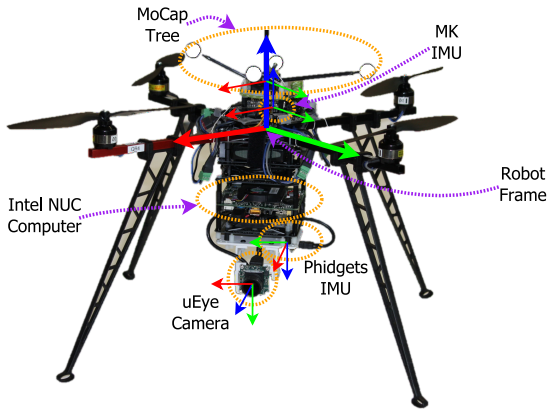


Figure 9.10: Hardware used for the experiments. The markers for the MOCAP are used only for the ground truth.

The simplified system architecture used for the experiment is represented in Fig. 9.11 and has the following components:

- Feature Extraction System. The *Aruco Eye* component described in Section 7.1.
- Motor System. It is formed by two components:
  - *Controller*: A complete controller (attitude, velocity and position) for the quadrotor, based on (Lee et al., 2011). It requires to have measurements at a rate of 1 kHz.
  - *Trajectory Commander*: Generates a smooth trajectory from the commanded motion reference.
- Situation Awareness System. It includes the two components described in the present section, the *MSF localization* and the *Environment Map Generation*. Additionally, it has another extra component:
  - *POM*[5]: Based on (Crassidis and Markley, 2003), it fuses the MK-IMU measurements with the proposed state estimator output. This cascaded approach confers robustness to the system when closing the hard-real time control loop at 1 kHz.

[2]Online: `http://www.mikrokopter.de`
[3]Online: `http://www.phidgets.com/products.php?product_id=1044_0`
[4]Online: `http://www.intel.eu/content/www/eu/en/nuc/nuc-kit-nuc5i7ryh.html`
[5]Online: `https://git.openrobots.org/projects/pom-genom3`

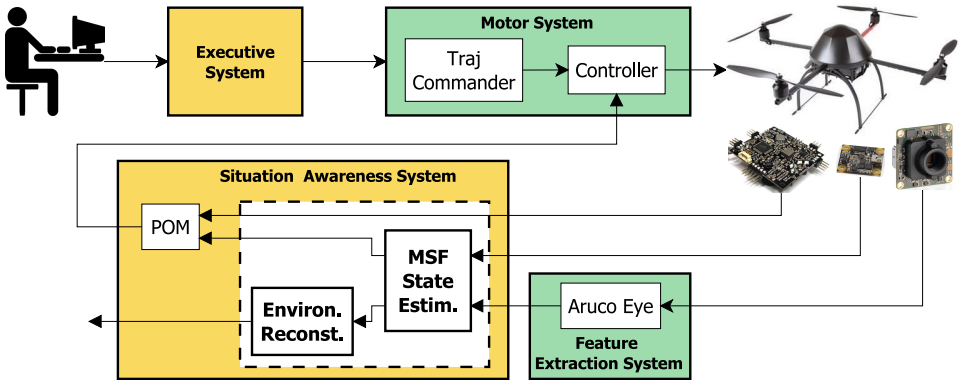


Figure 9.11: System architecture used for the experiments.

9.4.3. Results

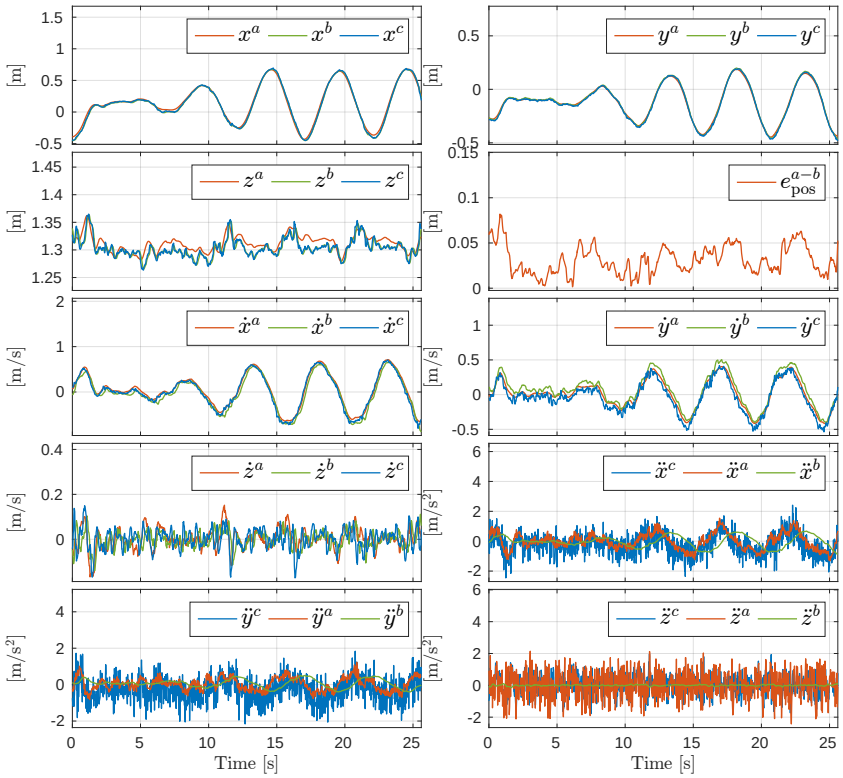
The goal of this experiment is to perform a simple trajectory using only onboard sensors showing the performances of the proposed estimator with the minimal sensor setup. After the take-off the robot is required to follow a circular trajectory on a horizontal plane at height 1.3 m, oscillating along the x-axis of the inertial frame between -0.5 and 0.6 m, and along the y-axis between -0.5 and 0.25 m, with a frequency of 0.8 Hz. The trajectory is performed while controlling the yaw angle to be equal to 45° in order to always head toward the wall developed along the x-axis of the inertial frame and placed at $y = 1.32$ m, on which 5 markers are placed.

Although this is a very simple trajectory, it is rich enough to show the capabilities of the proposed method for estimating the pose of the robot and its first and second derivatives, as it is shown in Fig. 9.12. In particular, this figure shows the estimated variables related to the translational and rotational dynamics respectively, for three different approaches:

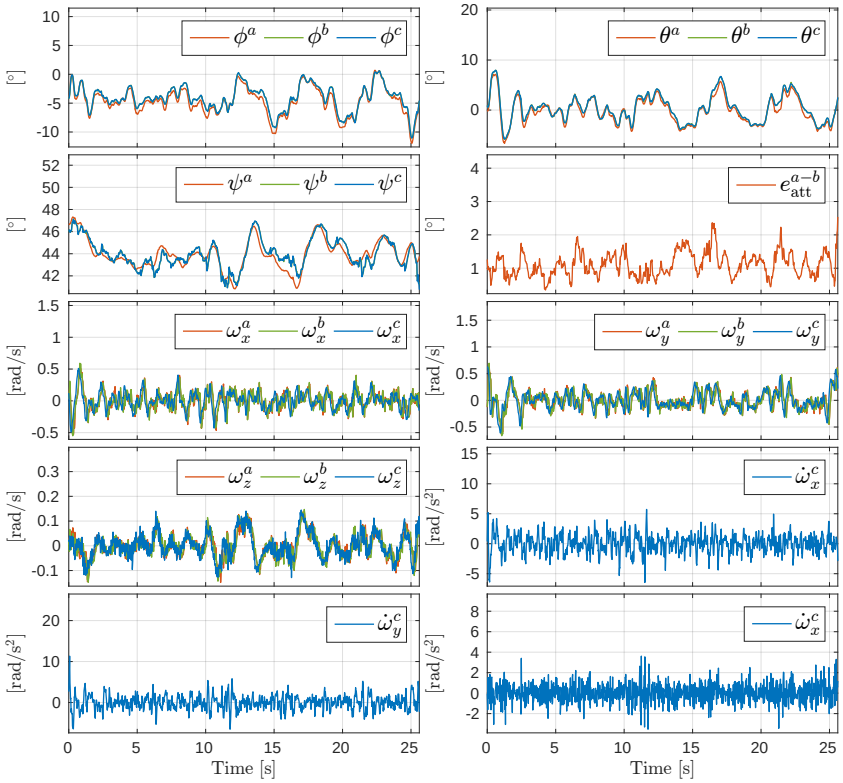
1. *POM+MOCAP*: POM (described before) is used to fuse the MK-IMU measurement with the one coming from a motion capture system (MOCAP). This estimated state can be used as the ground truth to evaluate the performances of the proposed MSF localization component;
2. *POM+MSF*: as said before, the use of POM on top of the proposed state estimator gives robustness to the final estimation;
3. *MSF*: the estimated state provided by the proposed MSF localization component without any additional filtering.

Furthermore, in the plots presented in Fig. 9.12 the mean estimation error on the position and on the orientation of the robot with respect to the methods Item 1) and 2) are shown, i.e., e_{pos}^{a-b} and e_{att}^{a-b}, respectively. Looking at these quantities can be seen how the estimation errors remain always bounded. In particular, the error stays between a minimum value less than 1 cm and a maximum value about 7 cm for the estimation of the position; and between a minimum value less than 1° and a maximum value about 2° for the estimation of the attitude. However, the reader must notice that the peaks on the errors coincide with the moments in which the aerial vehicle is at the farthest position from the markers. Indeed the more the distance between the camera and the marker, the higher the noise on the Coded Visual Markers Detector (Aruco Eye). Furthermore, the intensity of the noise depends on the type of the camera, its sensing parameters, and the light conditions as well.

From Fig. 9.12, comparing methods Item 2) and Item 3) with method Item 1), one can see that the linear and angular velocity and acceleration are well estimated, except for some small biases due to calibration errors. It is important to highlight the fact that the additional filtered method Item 2), as expected, reduces the noise, in particular on the estimated linear acceleration, but, on the other hand, introduces a small delay. The last origin of estimation errors, and in particular of biases, consists on the calibration errors that have to be minimized in order to obtain the best performances of the estimator.



(a) Estimated variables related to the translational dynamics.



(b) Estimated variables related to the rotational dynamics. Roll-pitch-yaw are used only for visualization purposes, all the estimator computation are done using unit quaternions.

Figure 9.12: Experimental results following a circular trajectory at a constant altitude.

9.5. Considerations and further improvements

This section explores some of the limitations of the presented perception solution and points out how to overcome them.

Limited kinds of sensors

Despite having the capability of including seven different kinds of sensors (gyro, accelerometer, coded visual marker detector, horizontal linear velocity, vertical distance, absolute pose, and unscaled absolute pose), there exist a large number of sensors that are widely used in aerial robotics, such as magnetometers, barometers or drifting pose sensors, in the absolute and unscaled versions (for odometry algorithms like mono-VO and stereo-VO), that need to be added as modules of the MSF state estimation component.

Mapping algorithm of world elements

As stated before, the proposed state estimator has mapping capabilities, being able to map some elements like the landmarks of high-level visual features (i.e. visual markers), or the world reference frames of the positioning systems, among others.

The current mapping approach is based on an EKF-SLAM algorithm as explained in Appendix D.3. The limitation of this mapping algorithm is that the performance of the estimator decreases with the number of mapped elements.

A planned future work is to explore other more efficient approaches for the mapping, like Sparse Extended Information Filters, which better scales up when the number of mapped elements grow significantly.

Sensors failures and anomalies

An important feature required by a sensor fusion algorithm is the capability of dealing with sensor failures and anomalies. The currently proposed algorithm relies on the Mahalanobis distance to detect and filter these anomalies, as explained in Appendix D.3.

A planned line of future work is to explore other more advanced approaches for the detection and filtering of sensor anomalies, such as the fuzzy algorithm presented in (Caron et al., 2006).

Full map estimation and fusion

As stated before, the proposed system is not performing a full map estimation and fusion from the used SLAM system. The full map estimation and the fusion from the used mapping components is a remaining future work that would increase the accuracy of the state estimation, together with the versatility on the environment reconstruction.

Environment reconstruction

As remarked in Section 9.3, the estimated covariance of the landmark i should be used when calculating the weight w_i for an improved estimation of the pose of the center of the 3D object.

Experiments

In Section 9.4, an experiment with a reduced number of hardware elements has been carried out to analyze the performance of the proposed solution, avoiding increasing the complexity when introducing a large and variate number of hardware elements. Nevertheless, a complete evaluation of all the proposed modules remains pending as future work, requiring to design a more complex battery of experiments.

9.6. Summary

In this chapter, a perception solution based on multi-sensor fusion with environment reconstruction has been presented.

The proposed solution combines in a modular way the information provided by a variable optional number of sensors and positioning algorithms in a loosely-coupling fashion. Moreover, the proposed solution is able to map some elements like the landmarks of high-level visual features (i.e. visual markers), or the world reference frames of the positioning systems and use them for the estimation of the robot state, that is, it is performing internally a simultaneous localization and mapping (SLAM). Thanks to this, the presented work enables the navigation of an aerial robot in different environments and carrying out different missions with the same perception architecture, exploiting the best from every sensor.

The key component with the responsibility of the combination of all the available information is an EKF-based state estimator the following features: 1) Inclusion of stochastic parameters to model uncertainty of fixed coefficients; 2) Error-value (indirect) method, accumulating the noise over the error state; 3) Quaternions for attitude representation, to avoid singularities; 4) Multiplicative update, to deal with quaternions accurately; 5) Time-delayed measurement compensation, by means of a circular buffer; 6) Iterative method, to quickly converge to the real state when the estimated state is far from the real state.

Furthermore, the proposed solution provides an environment reconstruction formed by geometric primitives, both for 3D and 2D environments.

The performance of the proposed perception solution has been evaluated in real experiments by means of a semi-autonomous flight with a minimal hardware setup. A quadrotor equipped with an extra IMU and an RGB camera used only to detect visual markers was autonomously controlled to track a simple, but rich enough trajectory. The full system architecture for the experiment, including the presented perception components, was running onboard. The state estimation output is comparable to the ground truth (given by a motion capture system) with average errors less than 4 cm and 1°.

An analysis of the main limitations of the proposed perception solution, together with the proposed future work to overcome them has been carried out.

The proposed perception solution is completely available to the scientific community as an open-source software integrated into Aerostack.

The proposed perception solution based on multi-sensor fusion with environment reconstruction, presented in this chapter, can be found on (Sanchez-Lopez et al., 2016a), and (Sanchez-Lopez et al., 2017a).

Chapter 10

Collision-Free Path Planning for Dynamic Environments

This chapter presents a solution for the fast generation of collision-free paths, able to interact with dynamic environments.

As analyzed in Section 4.5, for typical navigation tasks, like the path following, of common under-actuated multirotor aerial robots, the following simplifications can be done:

- No time restrictions are considered.
- Only the position and heading of the aerial robot have planning restrictions, being the other degrees of freedom, like their velocity and acceleration, imposed by its dynamic model. This allows considering the aerial robot as an holonomic vehicle.
- Symmetry in the horizontal dimensions of the aerial robot, decoupling the position path planning and the heading path planning.

An important requirement of the path planner is that it has to be able to generate collision-free paths fast enough to make possible an efficient reaction to changes in the mission, in the environment, or in the state of the robot, what is specially critical in aerial robots, since their unstable nature disqualifies them to passively wait for a slow response of the path planner.

The capability of generation of collision-free paths in dynamic environments is critical, specially when the aerial robot is part of a system formed by several moving robotics agents that share the same environment. The path planner is therefore required to work efficiently in dynamic environments to avoid collisions.

There exist multiple cases where the environment might be simplified to a 2D one. For example, the robot might be commanded to navigate maintaining a certain altitude, being, therefore, the environment considered as 2D for this flying altitude. This simplification, reduces the complexity of the planning problem, decreasing the computational requirements. The proposed path planner is limited to 2D environments, but, as pointed out in Section 10.3, this simplification can be easily removed.

The remainder of the chapter is organized as follows: in Section 10.1, the proposed path planner algorithm is presented. In Section 10.2 the performance of the proposed path planner is evaluated both with simulations and with real experiments. Section 10.3 discusses the main weaknesses of the proposed path planner, analyzing the way to overcome them and pointing out some lines of future work. Finally, Section 10.4 sums up the contributions of the chapter.

10.1. Path planning algorithm description

This section deeply describes the main details of the proposed algorithm, which has the following features:

- Objects of the environment described with geometric primitives (Section 10.1.1).
- Usage of a probabilistic roadmap to sample the arena (Section 10.1.2).
- Usage of an A* algorithm as a graph search algorithm (Section 10.1.5) to find the collision-free path in the probabilistic roadmap, guided by a potential field map (defined in Section 10.1.3) used as a cost function (as described in Section 10.1.4).
- Path shortening of the collision-free path (Section 10.1.6).
- Velocity and acceleration planning to achieve a faster traversal time[1] (Section 10.1.7).
- Collision-free path check after environment changes (Section 10.1.8).

10.1.1. Environment description

The first step required to create a collision-free path is the adequate definition of the environment. The environment is described by means of its boundaries (Section 10.1.1), the objects it has (Section 10.1.1), and the moving agents that are navigating there (Section 10.1.1).

The generation of this environment description is carried out by the Situation Awareness System, but the presented planning component imposes the kind of descriptor required.

Whenever a path planner query is requested, the proposed path planner generates an internal obstacle map based on the environment description, as the junction of the static objects map and the moving agents map.

Environment boundaries

The environment boundaries might have any shape and therefore the arena is not required to be rectangular.

Objects of the environment described with geometric primitives

The objects of the environment are described as a set of uniquely labeled geometric primitives. The advantages of using geometric primitives instead of, for example, a grid square cell map, point clouds, or surface representations, is that the environment is more compactly described but without loss of resolution.

Reducing the information used to describe the environment limits the number of queries to the cost function that evaluates the distance to an obstacle, speeding up the planning algorithm.

For simplicity, the proposed approach only uses two kinds of geometric primitives for the description of the objects of the environment: rectangles, and ellipses. The parameters that describe the geometric primitives are:

- Position of the center of the reference frame attached to the object in world coordinates, $\boldsymbol{t}^W_{O\star}$.
- Angle of the reference frame attached to the object in world coordinates, $\theta^W_{O\star}$.
- Dimensions of the object, $\boldsymbol{r}_{O\star} = [r_x,\ r_y]^T$. For ellipses, their radii are used; while for rectangles, the half of their sides are used.

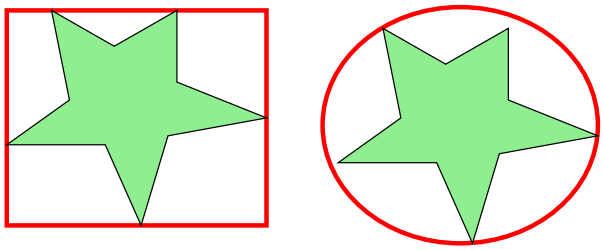
In case that the environment has objects that do not match any of the previously described geometric primitives shapes, the objects of the environment might be described, with a loss of resolution in the representation, using the following alternatives:

- Usage of a circumscribed rectangle or ellipse of the generic object (Fig. 10.1a). This is the simplest conservative way to describe the environment. This alternative is valid enough in the case of a nonpopulated environment. In the case of a very populated environment, this approach might occupy all the free space, being unable to calculate any collision-free path.
- Usage of an overlapped combination of rectangles and ellipses (Fig. 10.1b). This approach increments, in a conservative way, the resolution of the description of the environment, while using a reduced number of geometric primitives.
- Usage of a combination of rectangles, following a quadtree representation (Fig. 10.1c). This approach is less efficient than the previous one because it uses a higher number of geometric primitives, but it allows to use quadtree-based maps or grid square cell maps.

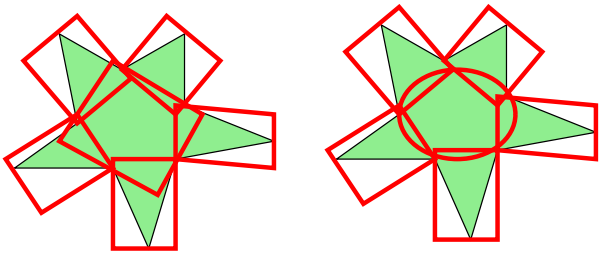
Moving agents

Similarly to the static objects map, the moving agents (like other aerial robot agents) are added to a moving agents map using the previously described geometric primitives but considering them as temporary objects. The

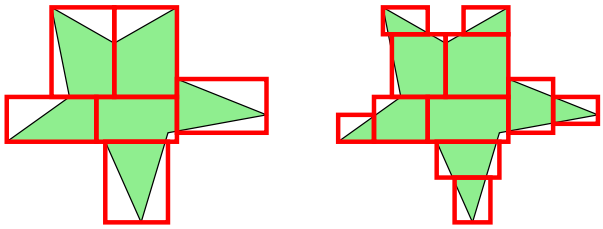
[1] This feature has been developed jointly with other researchers.



(a) Generic object described with a circumscribed rectangle and ellipse



(b) Generic object described as an overlapped combination of rectangles and / or ellipses



(c) Generic object described as a combination of rectangles, following a quadtree representation

Figure 10.1: Generic object described as a combination of basic geometric primitives (rectangles and ellipses).

size of the geometric primitives of the moving agents includes not only the actual size but also a fattening that depends on the movement (e.g. the velocity in module and direction) of these agents. In addition, if the moving agents are far enough (distance is greater than a threshold) from the current position of the agent that is planning the path, they are not added to the moving agents map, since they will most likely leave their current position before the agent reaches that point.

10.1.2. Arena sampling using a probabilistic roadmap

To reduce the time that the planner requires to plan a collision-free path, the arena is probabilistically sampled using a probabilistic roadmap (PRM). This probabilistic roadmap (also called graph) samples the arena (considering only its previously known boundaries), creating n random points (also called nodes or vertices) following a uniform distribution in the arena boundaries. The n sampled points are connected by means of edges to their nearest m neighbors (called m-neighborhood). The connection between any of two points of the probabilistic roadmap has to fall completely inside the arena.

The probabilistic roadmap is built at launching time and therefore no knowledge of the obstacles of the environment or the moving agents is incorporated to this probabilistic roadmap (see an example on Fig. 10.2), unlike (Kavraki et al., 1996). Creating a probabilistic roadmap that does not incorporate the knowledge of the fixed obstacles of the environment slows down the planning time because it will check the points that are in collision, but it allows to handle previously unknown obstacles with a dynamic mapping.

The previously known knowledge of the static objects of the environment can be incorporated to the probabilistic roadmap by modifying the uniform sampling function of the node generation into a custom probability density function (as depicted in Fig. 10.3). Moreover, the custom probability density function might include the knowledge of the existing corridors, as shown in Fig. 10.3.

The number of nodes n of the PRM must be representative of the environment. If the environment does not have a large number of obstacles, and the obstacles do not create corridors where the aerial robot must enter, the number of nodes n might be reduced. In the case of an environment populated with many obstacles, the number of nodes must be high. The higher the number of nodes, the higher the path planning time.

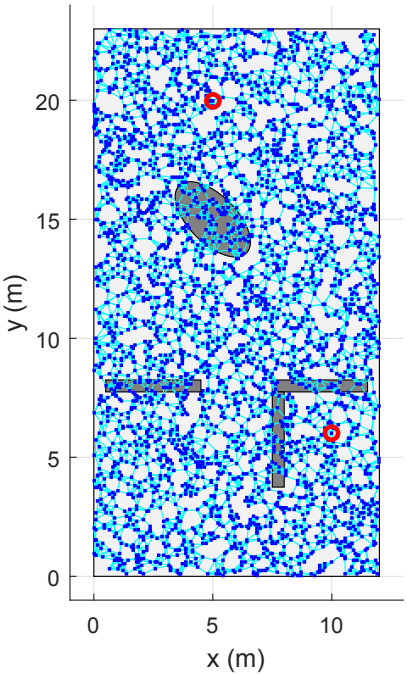


Figure 10.2: Probabilistic roadmap without any previuous knowledge about the obstacles. The red circles represent the initial and final points. The number of nodes used is $n = 3500$, with a 6-neighborhood.

Whenever a planning query is requested, the initial point and the final point of the query are temporarily added to the probabilistic roadmap, removing them after the creation of a collision-free path.

10.1.3. Potential field map as a cost function

The potential field map (PFM), see (Hwang and Ahuja, 1992), is adopted as the cost function (see Section 10.1.4) used by the planner to guide the search of the collision-free path in the probabilistic roadmap (as described in Section 10.1.5).

The total potential field map, $p_T = p_T(P)$, in a point P of the arena, is given by (see Fig. 10.4c):

$$p_T = p_q + p_O = p_q + \sum_{\forall i} p_{O_i} \tag{10.1}$$

where p_q is the contribution of the planning query, p_O is the contribution of all the obstacles; and p_{O_i} is the contribution of the obstacle O_i.

Potential field map of the planning query

To guide the search in the direction from the initial point P_0 to the target point P_f, a potential field map cost function is created, p_q. Both initial and target points are given in world coordinates, $\boldsymbol{t}^W_{P_0}$ and $\boldsymbol{t}^W_{P_f}$, respectively.

The contribution of the query to the potential field map, $p_q = p_q\left(P, P_0, P_f\right)$, in a point P of the environment, is given by a parabolic expression (see Fig. 10.4a):

$$p_q = \frac{\left(x^W_P - x^W_{P_f}\right)^2}{c_a} + \frac{\left(y^W_P - y^W_{P_f}\right)^2}{c_b} + k_f \tag{10.2}$$

being

$$c_a = \frac{k_r \cdot \left(x^W_{P_0} - x^W_{P_f}\right)^2 + \left(y^W_{P_0} - y^W_{P_f}\right)^2}{k_r \cdot \left(k_0 - k_f\right)} \tag{10.3}$$

$$c_b = k_r \cdot c_a \tag{10.4}$$

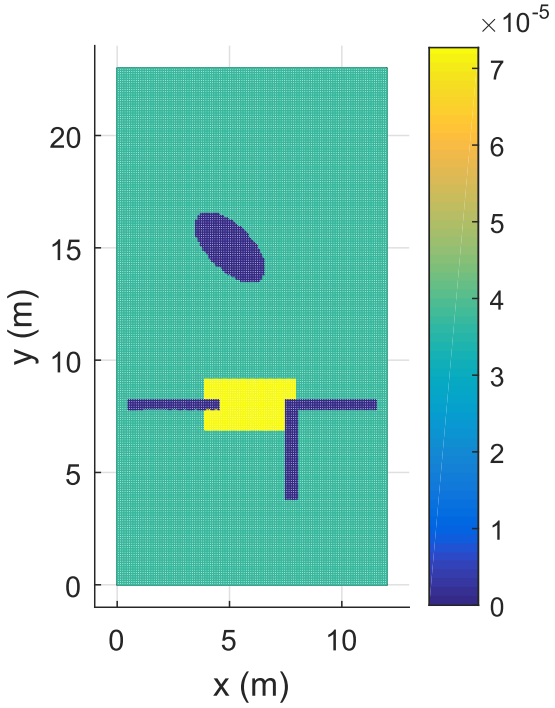


Figure 10.3: Example of a probability density function for the PRM generation incorporating previous knowledge of the environment.

where the coefficient k_r determines the priority of the planned movement in a specific direction; the coefficient k_0 determines the altitude (cost) of the potential field map in the initial point; and the coefficient k_f determines the altitude (cost) of the potential field map in the final point. All these coefficients determine the behavior of the planner.

Potential field map of the obstacles

The contribution of the obstacle $O\star$ to the potential field map, $p_{O\star} = p_{O\star}(P, O\star)$, in a point P of the environment, is calculated using the implicit equation v of the obstacle (see Fig. 10.4b), and it is given by:

$$p_{O\star} = \begin{cases} \infty, & \text{if } v \leq 0 \\ 0, & \text{if } v > v_{ig} = \frac{1}{k_2} \cdot \log\left(\frac{1}{c_{ig}} - 1\right) \\ \frac{k_1}{1+e^{k_2 \cdot v}}, & \text{otherwise} \end{cases} \tag{10.5}$$

being k_1 and k_2 two coefficients that determine the behavior of the path planner (in getting close to the obstacles); and c_{ig} a percentage value that indicates the contribution of the object $O\star$ in the potential field map, being ignored if its contribution to the cost is lower than $c_{ig} \cdot k_1$, as it's considered to be far enough.

The implicit equation $v = v(P, O\star)$, in a point P of the environment described in world coordinates by $\boldsymbol{t}_P^W = [x_P^W,\ y_P^W]^T$, generated by the obstacle $O\star$, is given by:

- Implicit equation of an ellipse:

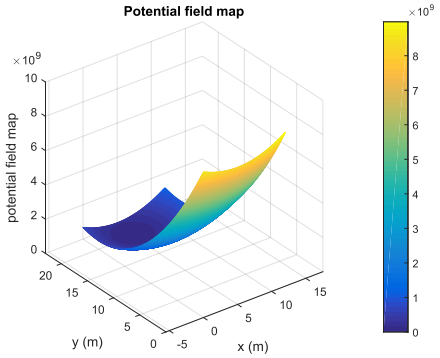
$$v = \left(\frac{x_P^{O\star}}{\hat{r}_x}\right)^2 + \left(\frac{y_P^{O\star}}{\hat{r}_y}\right)^2 - 1 \tag{10.6}$$

- Implicit equation of a rectangle:

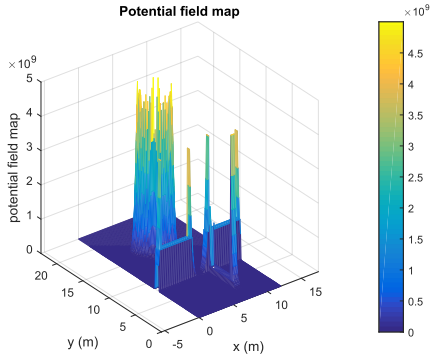
$$v = \left|\frac{x_P^{O\star}}{\hat{r}_x} + \frac{y_P^{O\star}}{\hat{r}_y}\right| + \left|\frac{x_P^{O\star}}{\hat{r}_x} - \frac{y_P^{O\star}}{\hat{r}_y}\right| - 2 \tag{10.7}$$

where Eq. (10.6) and Eq. (10.7) are transformed from object-relative coordinates to world coordinates by means of:

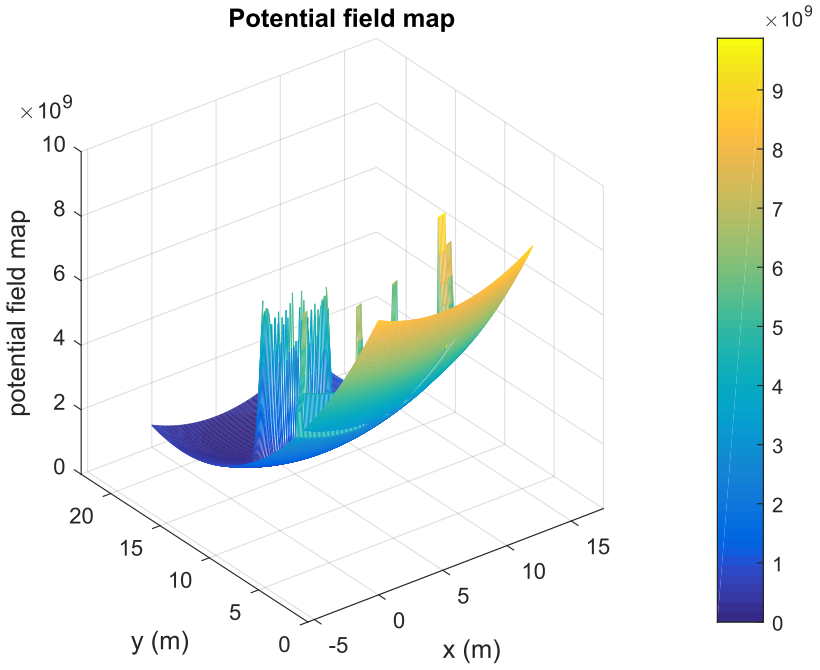
$$\boldsymbol{t}_P^{O\star} = \left(\boldsymbol{R}_{O\star}^W\right)^T \cdot \left(\boldsymbol{t}_P^W - \boldsymbol{t}_{O\star}^W\right) \tag{10.8}$$



(a) Contribution of the planning query to the potential field map.



(b) Contribution of the obstacles to the potential field map.



(c) Total potential field map.

Figure 10.4: Potential field map for a query from $t_{P_0} = [10,6]^T$ and $t_{P_f} = [5,20]^T$, with two rectangular obstacles, and an elliptical obstacle.

and

$$\boldsymbol{R}_{O\star}^{W} = \begin{bmatrix} \cos\theta_{O\star}^{W} & -\sin\theta_{O\star}^{W} \\ \sin\theta_{O\star}^{W} & \cos\theta_{O\star}^{W} \end{bmatrix} \tag{10.9}$$

The value $v = v(P, O\star)$ of the implicit equations in the point P has the following meaning: if $v < 0$, the point P is inside the geometric primitive $O\star$; if $v = 0$, the point P is in the boundaries of the geometric primitive $O\star$; and if $v > 0$, the point P is outside the geometric primitive $O\star$.

To consider the dimensions of the robot in the planned path, the dimensions of the geometric primitives are augmented following the equation $\hat{\boldsymbol{r}}_{O\star} = \boldsymbol{r}_{O\star} + \max(\boldsymbol{r}_R)$, being $\boldsymbol{r}_R$ the dimensions of the robot, and $\max(\boldsymbol{r}_R)$ a vectorial representation of the maximum dimension of the robot.

It is important to note that the use of the implicit equation, is a simplification that allows having an intuition of the distance to the objects in a very fast way, but since its value differs to their actual distance, the shape of the potential field map will be different than if it would be calculated using the distance value.

10.1.4. Cost of moving between two points

Given a potential field map p (as the one mentioned in Section 10.1.3), it generates a surface S given by $S = \left[\boldsymbol{t}_P^W,\ p\left(\boldsymbol{t}_P^W\right)\right]^T$, for all points P of the arena.

The cost of moving between points A and B following the curve $C_{A\to B}$, is given by the line integral for the unit scalar field along the curve $C_{A'\to B'}$, that is the projection of the curve $C_{A\to B}$ over the surface S.

$$c(A,B) = \int_{C_{A'\to B'}} 1 \cdot \|\mathrm{d}\boldsymbol{l}\| \approxeq \sum_{C_{A'\to B'}} \|\Delta \boldsymbol{l}\| \tag{10.10}$$

where points A and B are given in world coordinates as $\boldsymbol{t}_{P_A}^W = [x_A,\ y_A]^T$ and $\boldsymbol{t}_{P_B}^W = [x_B,\ y_B]^T$ respectively.

If the curve $C_{A\to B}$ is parametrized by $s \in [s_{min}, s_{max}]$, then, any point P of this curve is given by $\boldsymbol{t}_P^W(s)$, and for a $\Delta s = s_i - s_{i-1}$:

$$\Delta \boldsymbol{l} = \begin{bmatrix} \Delta \boldsymbol{t} \\ \Delta p \end{bmatrix} = \begin{bmatrix} \boldsymbol{t}_P^W(s_i) - \boldsymbol{t}_P^W(s_{i-1}) \\ p\left(\boldsymbol{t}_P^W(s_i)\right) - p\left(\boldsymbol{t}_P^W(s_{i-1})\right) \end{bmatrix} \tag{10.11}$$

and sampling the s interval, the cost of moving between points A and B, is calculated combining equations 10.10 and 10.11, getting:

$$c(A,B) \approxeq \sum_{s_i=s_{min}}^{s_{max}} \sqrt{\|\Delta \boldsymbol{t}\|^2 + \|\Delta p\|^2} \tag{10.12}$$

In the case that the curve, $C_{A\to B}$ is a straight line connecting points A and B, the following parametrization can be used:

$$s \in [s_{min}, s_{max}] = \left[0,\ \left\|\boldsymbol{t}_{P_B}^W - \boldsymbol{t}_{P_A}^W\right\|\right] \tag{10.13}$$

being a generic point P of the world over the curve $C_{A\to B}$ given by:

$$\boldsymbol{t}_P^W(s) = \boldsymbol{t}_{P_A}^W + s \cdot \boldsymbol{n}_C \tag{10.14}$$

where

$$\boldsymbol{n}_C = \frac{\boldsymbol{t}_{P_B}^W - \boldsymbol{t}_{P_A}^W}{\left\|\boldsymbol{t}_{P_B}^W - \boldsymbol{t}_{P_A}^W\right\|} \tag{10.15}$$

and for a $\Delta s = s_i - s_{i-1}$:

$$\Delta \boldsymbol{t} = \Delta s \cdot \boldsymbol{n}_C \tag{10.16}$$

and therefore, sampling the s interval, the cost of moving between points A and B, is calculated by:

$$c(A,B) \approxeq \sum_{s_i=0}^{s_{max}} \sqrt{\Delta s^2 + \left(p\left(\boldsymbol{t}_P^W(s_i)\right) - p\left(\boldsymbol{t}_P^W(s_{i-1})\right)\right)^2} \tag{10.17}$$

The number of sampling points of the s interval, n_{s_i}, is calculated taken into account that the Δs has to be smaller than the smallest dimension of all the obstacles, and hence, at least a sampling point will fall inside the obstacle if it is crossed. Therefore, the minimum number of sampling points of the s interval, $n_{s_i} \in \mathbb{Z}$ is given by:

$$n_{s_i} \geq \frac{s_{max}}{\min_{\forall O\star}\left(\hat{\boldsymbol{r}}_{O\star}\right)} \tag{10.18}$$

10.1.5. A* algorithm to find a collision-free path in the probabilistic roadmap graph

The proposed approach uses a simple but efficient widely used A* algorithm, (Hart et al., 1968), as a graph search algorithm, to find the collision-free path in the probabilistic roadmap (calculated in Section 10.1.2), guided by the potential field map as the cost function (as explained in Section 10.1.4).

This informed search algorithm searches among all possible paths to the solution for the one that incurs in the smallest cost, and among these paths, it first considers the ones that appear to lead most quickly to the solution. At each iteration of its main loop, it needs to determine which of its partial paths to expand into one or more longer paths. It selects the path that minimizes:

$$f(n_i) = g(n_0, n_i) + h(n_i, n_f) \tag{10.19}$$

where n_i is the working node on the current iteration, $g(n_0, n_i)$ is the smallest cost of the path from node n_0 to node n_i, and $h(n_i, n_f)$ is an heuristic that estimates the cheapest path from node n_i to node n_f.

In the presented planner, the cost $g(n_0, n_i)$ is given by the sum of the cost needed to move between node n_0 and node n_i in the probabilistic roadmap (see Section 10.1.4):

$$g(n_0, n_i) = \sum_{n_j=n_0}^{n_{j+1}=n_i} c\left(n_j, n_{j+1}\right) \tag{10.20}$$

where the cost function $c\left(n_j, n_{j+1}\right)$ corresponds to the total potential field map, incorporating the contribution of the query and the obstacles.

The heuristic cost $h(n_i, n_f)$ is given by the cost needed to move from node n_i to node n_f in a straight line in the environment (not in the graph):

$$h(n_i, n_f) = c\left(n_i, n_f\right) \tag{10.21}$$

where the cost function $c\left(n_i, n_f\right)$ corresponds to the potential field map with only to the contribution of the query.

After the exploration of the graph, the path t_p (also called plan) has to be created by revisiting the explored path. An example of the collision-free path computed might be seen in Fig. 10.5.

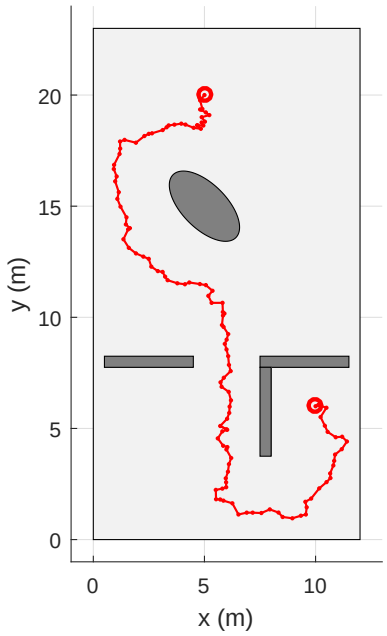


Figure 10.5: Raw collision-free path generated with the A* algorithm from point $t_{P_0} = [10, 6]^T$ to point $t_{P_f} = [5, 20]^T$.

10.1.6. Path shortening

Once the collision-free path has been found, a post-processing is applied to shorten it. An example is shown in Fig. 10.6.

In the presented algorithm, the path is shortened, t_s, by taking into account the obstacles of the environment. The initial node n_0 and the final node n_f are always the beginning and ending points of the shortened path. To obtain the middle points of the shortened path, Algorithm 2 is used.

10.1.7. Velocity and acceleration planning

In order to achieve a faster traversal time, the sharp corners of the trajectory are substituted, when possible, by circumference arcs and a velocity and acceleration plans are calculated over the resulting path. Therefore, the trajectory executed by the aerial robot is a sequence of straight segments and circumference arcs. The velocity

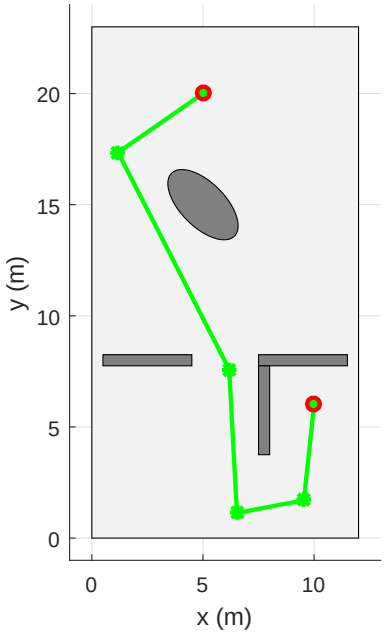


Figure 10.6: Shortened collision-free path from point $t_{P_0} = [10,6]^T$ to point $t_{P_f} = [5,20]^T$.

Algorithm 2 Path shortening algorithm.

Initialization: $t_s = \{\}$, $n_i = n_0$, $n_j = n_{i+1}$
Loop:
while $n_j \neq n_f$ **do**
  if $c(n_i,n_j) \geq \sum_{n_k=n_i}^{n_{k+1}=n_j} c(n_k,n_{k+1})$ **then**
    $t_s = \{t_s,n_j\}$, $n_i \leftarrow n_j$, $n_j \leftarrow n_{j+1}$
  else
    $n_j \leftarrow n_{j+1}$
  end if
end while
Note: The cost function $c(n_i,n_j)$ corresponds to the potential field map, incorporating only the contribution of the obstacles.

and acceleration plans, inspired in (Hoffmann et al., 2008), are calculated over the trajectory as described in (Pestana et al., 2013a; Pestana et al., 2014b).

In the circumference arcs calculation, the radii are limited to ensure that the trajectory does not deviate from the shortened path, further than a parametrized confidence distance. The velocity plan is calculated considering and fulfilling the following constraints: the equations of the uniformly accelerated motion and the parametrized maximum velocity, v_{max}, and acceleration constraints, a_{max}. The maximum velocity for each turn is calculated as $v_{turn} = \sqrt{a_{max} \cdot R_{turn}}$, where R_{turn} is the arc radius. When a turn is too sharp, it stays as a concatenation of two straight segments. Such turns are almost stall-turn maneuvers, and therefore, the velocity plan in such sharp turns is preconfigured with a parameter named $v_{stall\,turn}$.

The velocity and acceleration plans resemble a bang-bang type control on the commanded acceleration, constrained by the maximum velocity and the turn velocities, which are themselves constrained by the maximum acceleration.

10.1.8. Collision-free path check

The planned collision-free path t_s has to be checked whenever the environment changes (a new static object has been mapped, or a moving agent has moved) to determine if it is still collision-free.

If the cost of the planned collision-free path t_s is lower than a threshold, the path is considered to be

collision-free, and it is calculated by:

$$c(n_0, n_f) = \sum_{n_j=n_0}^{n_{j+1}=n_f} c\left(n_j, n_{j+1}\right) \tag{10.22}$$

where the cost function $c\left(n_j, n_{j+1}\right)$ corresponds to the potential field map, incorporating only the contribution of the obstacles.

10.2. Evaluation and results

10.2.1. Methodology

The proposed path planner, incorporated as an open-source component in Aerostack, has been intensively used in multiple research projects, including two international competitions, IMAV 2013 (see Section 12.2.2), and IARC 2014 (see Section 12.2.3); in several experiments (see Section 12.3.1 and Section 12.3.4), and multiple public demonstrations (see Section 12.4).

Its performance has been demonstrated when used in a system formed by a single aerial robot, and in a system with multiple aerial agents. The environments tested included both previously known and previously unknown objects, with different shapes, as a house with a door and windows, column areas, and narrow passages. The tested obstacles were both fixed and moving. In all these challenges, the planner was able to calculate collision-free paths in real time, re-planning them fast enough when needed.

An intensive isolated evaluation of the performance of the presented path planner is done in Section 10.2.2, by means of two different simulations with very challenging environments. These simulations have been run several times, showing that the random nature of the PRM leads to different results.

To complement the simulations, an experiment with a real flight is presented in Section 10.2.3. In this experiment, the performance of the presented path planner is evaluated within Aerostack through an emulated search and rescue mission similar than the one presented in Section 12.3.4.

10.2.2. Simulation in complex static environments

In this section, the performance of the presented path planner is evaluated by means of two simulations. The environments of these simulations, formed only by static objects, are very challenging and unrealistic compared with the environments that an aerial robot normally faces.

System setup

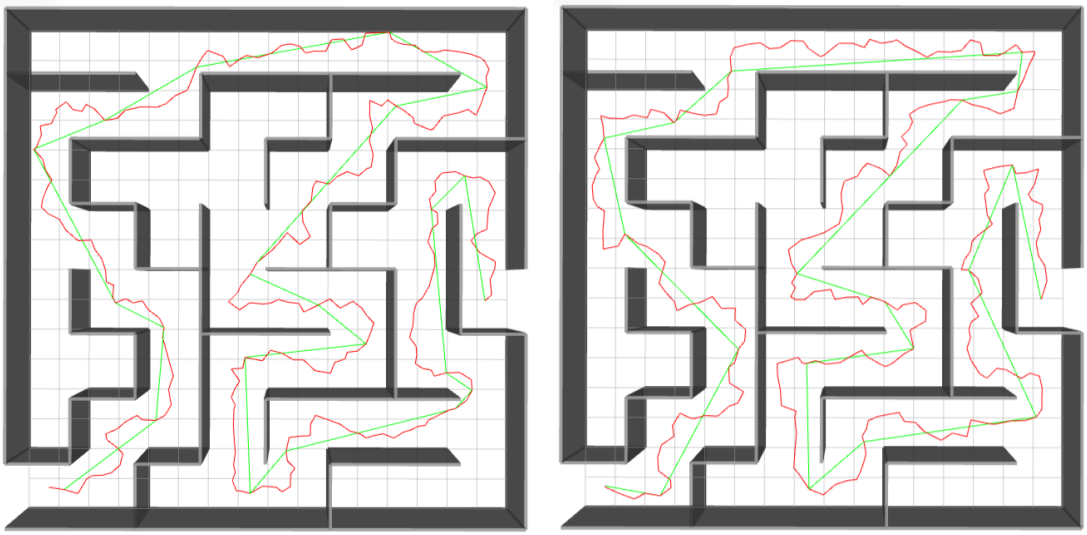
In these tests, no previous knowledge about the objects of the environment is incorporated to the planner. The reader must note that in these environments, the heuristic given by the potential field map is not contributing to speed up the search process and therefore the graph is needed to be visited almost completely by means of the A* algorithm to be able to find the solution. The aerial robot is configured as an ellipse of 0.5×0.5 m^2.

Results

In both tests, shown in Figs. 10.7 and 10.8, the raw collision-free path over the PRM is displayed in red, while the green line is the shortened collision-free planned path.

Fig. 10.7 shows the collision-free paths calculated for an environment that is a labyrinth of dimensions 16×16 m^2, to move from point $t_{P_0} = [1,\ 1]^T$ (bottom left) to point $t_{P_f} = [15,\ 7]^T$ (middle right). Two different runs of the algorithm are shown. As the obstacles density is very high, the PRM requires a large number of nodes, set to 3000 in the presented case (approx. 11.72 nodes/m^2, same number of elements in the graph than an equivalent 2D cell map with a resolution of 29 cm/cell). The neighborhood of the nodes of the PRM has been set to 6 (higher than the 4-neighborhood of a 2D cell map).

Fig. 10.8 shows the collision-free path calculated for an environment of dimensions 32×12 m^2, that include five dead end areas (creating local minimum in the potential field map) and a narrow passage, to move from point $t_{P_0} = [3,\ 6]^T$ (left) to point $t_{P_f} = [28,\ 6]^T$ (right). Four different runs are shown. The configured number of nodes of the PRM is set to 3000 (approx. 7.81 $nodes/m^2$, the same number of elements in the graph than an equivalent 2D cell map with a resolution of 36 cm/cell), with a 6-neighborhood.



(a) Run 1. (b) Run 2.

Figure 10.7: Collision-free path calculated for a labyrinth of dimensions 16×16 m^2, to move from point $t_{P_0} = [1,\ 1]^T$ (bottom left) to point $t_{P_f} = [15,\ 7]^T$ (middle right).

Despite the high complexity of the simulated environments, the proposed path planner is able to find a solution without falling in any local minima, validating its robust behavior. Moreover, as compared, it is more efficient than the 2D cell map based equivalent planner, as it uses a smaller number of nodes for a particular resolution.

10.2.3. Real experiments with a multi-robot system

This section presents a real experiment with multiple (three) aerial agents performing a fully autonomous indoors mission. An emulated search and rescue mission, similar to the one presented in Section 12.3.4 is executed. In this mission, an emergency situation is emulated inside a house (right side of Fig. 10.9). Two aerial robots (labeled as red in Fig. 10.9) take-off from the emergency team base (left side of Fig. 10.9) and enter the building searching for a subject (labeled as green in Fig. 10.9). Once detected, the target is found, the two search robots return to the rescue team base while another aerial robot (labeled as blue in Fig. 10.9) perform a rescue task (i.e. a visual servoing interaction Section 12.3.2) to the subject. After the rescue, the aerial robot returns to the rescue team base.

System setup

The same system setup than in the experiments presented in Section 12.3.4 is used.

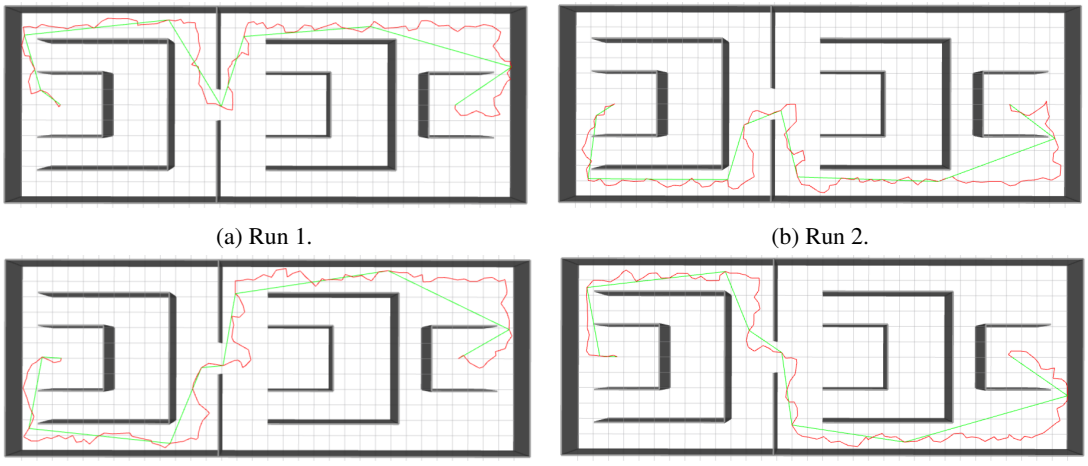
Three Parrot AR.Drone 2.0. are used as aerial platforms, individually connected to three average laptops using a Wi-Fi interface. The three laptops are connected to a LAN by means of a switch. Every laptop is running all the components of Aerostack required for every robotic agent to perform the fully autonomous mission, including a separate instance of the presented path planner per aerial robot.

The dimensions of the arena are 9×11 m^2. The configured number of nodes used in the PRM is inflated to 3000 (approx. 30.30 nodes/m^2, same number of elements in the graph than an equivalent 2D cell map with a resolution of 18 cm/cell) The neighborhood of the nodes of the PRM has been set to 5 (higher than the 4-neighborhood of a 2D cell map). The aerial robot is configured as an ellipse of 0.7×0.7 m^2.

Whenever the proposed planner is unable to find a collision-free path, an empty path is commanded to the path following controller, meaning that it has to hover in the current position until a new collision-free path is found.

Results

Fig. 10.10 shows the trajectories followed by the aerial robots. The two search aerial robots, hereinafter designated as *drone1* and *drone2*, are labeled with the red and green trajectories respectively. The rescue robot,



(a) Run 1. (b) Run 2.

(c) Run 3. (d) Run 4.

Figure 10.8: Collision-free paths calculated for an environment of dimensions 32×12 m^2, to move from point $t_{P_0} = [3,\ 6]^T$ (left) to point $t_{P_f} = [28,\ 6]^T$ (right).

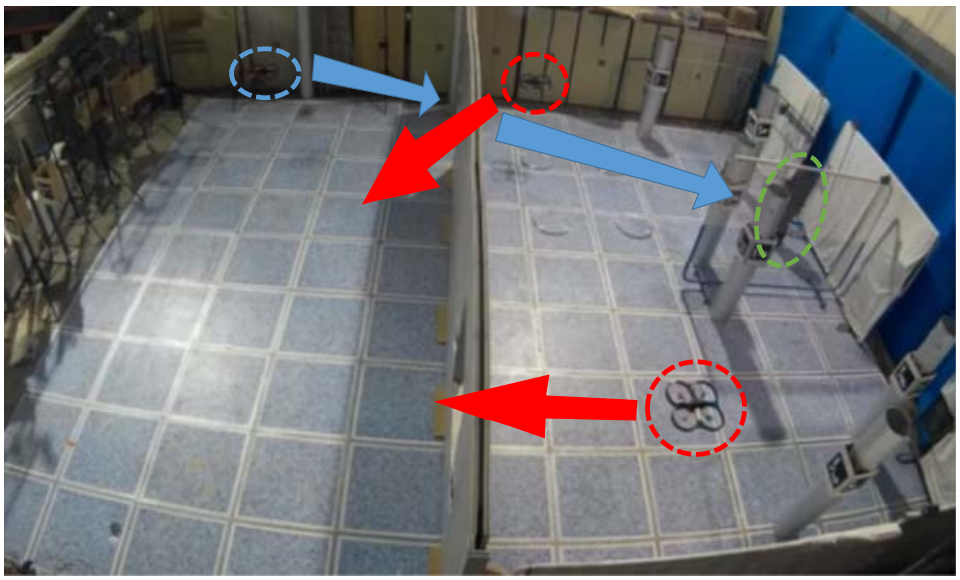


Figure 10.9: A frame of the search and rescue mission, after the detection of the target.

designated as *drone3*, is labeled with the blue trajectory.

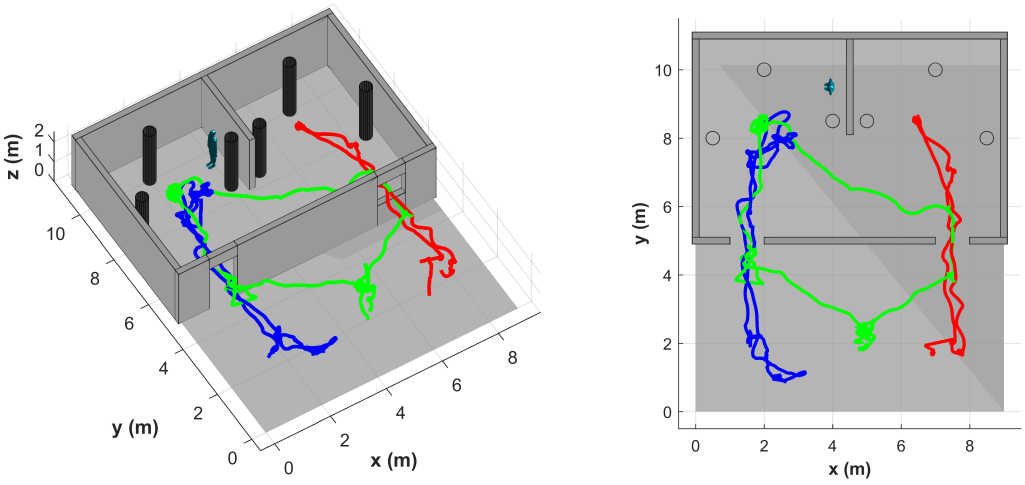
Fig. 10.11 shows the trajectories followed by the aerial robots (solid lines), together with their planned path (dashed lines), in different instant times. As can be extracted, the proposed path planner is able to generate collision-free paths fast enough to ensure an adequate deliberative behavior.

In Fig. 10.11a, the mission planner of *drone1* commands it to cross the window (point $[7.5,\ 6]^T$) after take-off, *drone2* is commanded to face the window (point $[7.5,\ 4]^T$) before crossing it, to guarantee that it enters the house through the window, generating a conflict between the agents on purpose to show the performance of the proposed planner.

In Fig. 10.11b can be seen that *drone2* needed to plan a new path after *drone1* moved forward to avoid it. Nevertheless, as *drone1* moves forward, *drone2* is unable to plan a collision-free path to reach the commanded point, and it waits until *drone1* has cleared the commanded point (as seen in Fig. 10.11c).

After crossing the window, *drone1* is commanded to explore the left room (point $[6.5,\ 8.5]^T$). Once the path is clear, *drone2* is able to plan a collision-free path to face the window, and after that, it is commanded to explore the right room (point $[2,\ 8.5]^T$), as represented in Fig. 10.11d.

Once the target is detected by *drone2*, both exploration robots are commanded to return to their take-off positions (*drone1* to point $[7.5,\ 2]^T$; and *drone2* to point $[5,\ 2]^T$). In the meantime, *drone3* takes off and it is



(a) 3D view

(b) 2D view

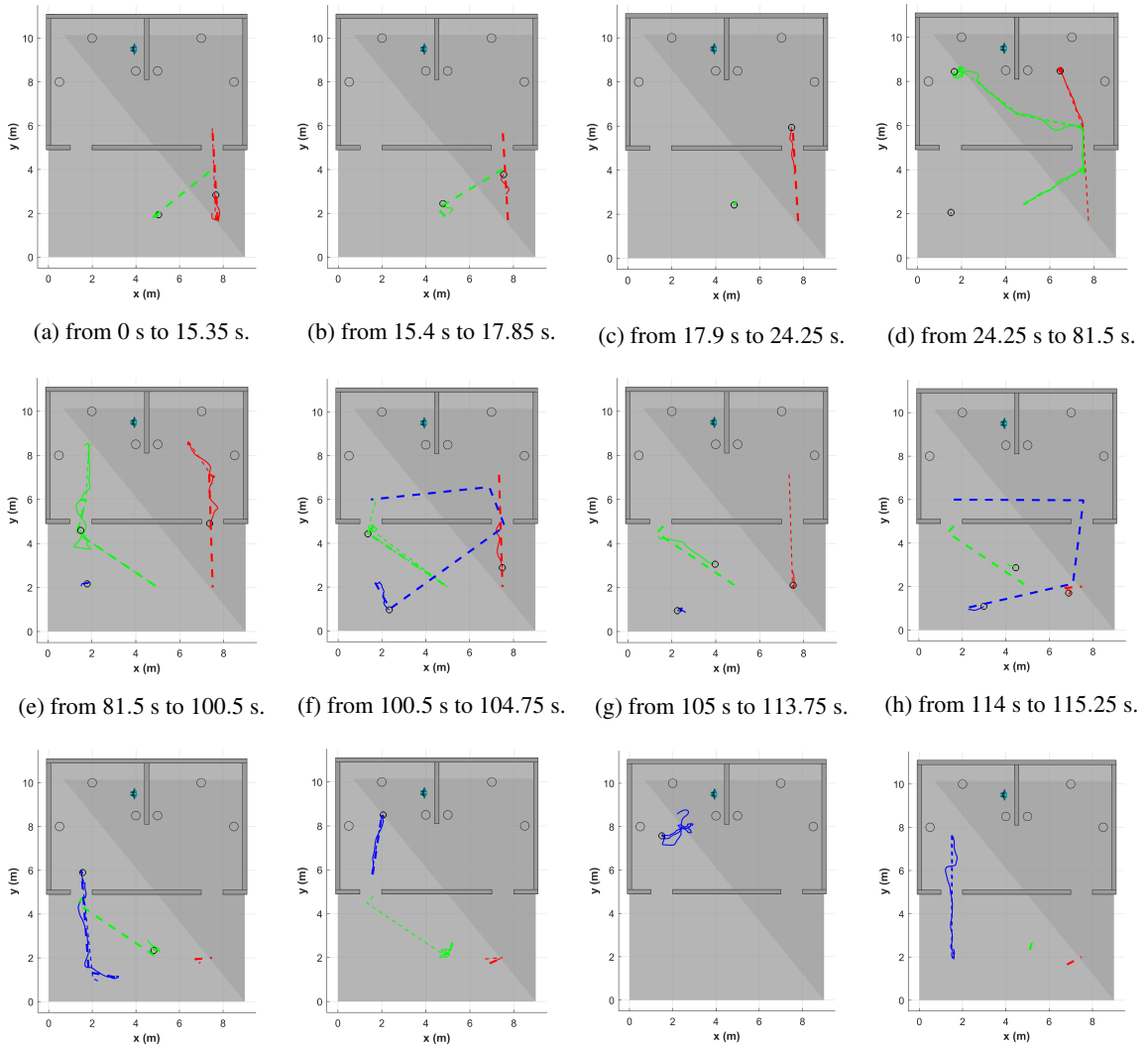
Figure 10.10: Trajectories followed by the aerial robots.

commanded to cross the doorway (point $[1.75,\ 6]^T$). Nevertheless, as can be seen in Fig. 10.11e, while the doorway is occupied, *drone3* is unable to find a collision-free path.

In Figs. 10.11f to 10.11i an interesting situation where *drone3* is trying to find a collision-free path with the disturbances of *drone1* and *drone2* is shown. In Fig. 10.11f, as the doorway is occupied by *drone2*, *drone3* plans a collision-free path through the window, since *drone1* has already cleared it. As *drone2* moves forward, the previously planned path by *drone3* is not collision-free anymore, and as *drone2* is still very close to the doorway, *drone3* is unable to find a collision-free path, as represented in Fig. 10.11g. As soon as *drone1* lands, it is not considered as moving agent anymore, and *drone3* is able to find again a collision-free path through the window (Fig. 10.11h). Finally, in Fig. 10.11i, as *drone2* is moving forward, *drone3* is able to find a shorter collision-free path crossing the doorway.

In Fig. 10.11j *drone3* is commanded to face the target moving to point $[2,\ 8.5]^T$. Once it reaches this point, *drone3* performs the visual servoing task (Fig. 10.11k). Whenever the subject is labeled as rescued, *drone3* is commanded to return to its take-off position (point $[1.5,\ 2]^T$), as shown in Fig. 10.11l.

This experiment demonstrates that the proposed path planner is able to operate in real-time, generating collision-free paths robustly without falling in any local minima in a complex environment formed by both static obstacles and moving agents. As explained, the proposed trajectory planner was the only component responsible for the collision avoidance.



(a) from 0 s to 15.35 s. (b) from 15.4 s to 17.85 s. (c) from 17.9 s to 24.25 s. (d) from 24.25 s to 81.5 s.

(e) from 81.5 s to 100.5 s. (f) from 100.5 s to 104.75 s. (g) from 105 s to 113.75 s. (h) from 114 s to 115.25 s.

(i) from 115.5 s to 129.75 s. (j) from 130 s to 149 s. (k) from 149.5 s to 186.25 s. (l) from 186.5 s to 205.35 s.

Figure 10.11: Trajectories followed by the aerial robots in several time frames. Solid lines represent the followed paths, while dashed lines are the planned paths.

10.3. Considerations and further improvements

This section explores some of the limitations of the presented path planner and points out how to overcome them.

Generalization to other environment descriptions

The proposed position path planner imposes a description of the environment based on two kinds of geometric features, rectangles, and ellipses. This requires an adequate Situation Awareness System that transforms the available environment description into the geometric primitive based one.

On the one hand, more kinds of geometric features like triangles should be added for a richer and more accurate description of the environment.

On the other hand, other environment descriptions such as quadtrees or cell maps might be directly used by considering each cell as a rectangular obstacle. Nevertheless, its direct usage is highly discouraged as the efficiency of the presented algorithm would be drastically reduced.

Generalization to 3D environments

The generalization to 3D environments is an immediate future work. For a 3D environment, the geometric primitives that might be used for environment description are prisms, ellipsoids, and cylinders, among others.

Therefore, new implicit equations v would need to be defined for such geometric primitives. The rest of the algorithm presented in Section 10.1 would remain equivalent.

Dynamic arena sampling

An important limitation of the proposed path planner is the need of sampling the environment with a probabilistic roadmap, and therefore, selecting the number of nodes used, their neighborhood, and their sampling probability density function. As discussed before, a large number of nodes lead to inefficient searches, but a small number of nodes yield an inaccurate coverage of the free environment, what is specially critical if the environment is populated with a large number of obstacles or the obstacles create small corridors.

To overcome this limitation an improvement over the PRM might be done, such as a Lazy-PRM, (Bohlin and Kavraki, 2000); or a C-PRM, (Song et al., 2001). In this approach, whenever a plan is not found, more nodes might be dynamically added to the probabilistic roadmap to have a better coverage of the environment. A possible criteria to add these nodes might use the potential field map, adding random nodes with higher probability where the potential field map has higher values (excluding the value ∞ of equation 10.5, so excluding the obstacles), that is, near the obstacles and their surroundings.

Efficient management of moving agents

The time that the presented path planner requires to create a collision-free path mainly depends on the number of nodes of the probabilistic roadmap, the number of obstacles, the position and shape of the obstacles, and the distance between the initial point and the final point.

As explained in Section 10.1.8, the planned collision-free path is checked whenever the environment changes. Whenever this planned path is not collision-free anymore, a new collision-free path calculation needs to be done. This would happen often in an environment with a large number of moving agents or if the moving agents are moving in the neighborhood of the robot. Every time a planning request is executed, the A* algorithm is executed from the beginning. A more efficient approach would reuse the information of a previous query, updating only the information that differs from the previous query, speeding up the search. This reuse of information might be done using a better graph search algorithm like a D* algorithm, (Stentz, 1994).

10.4. Summary

In this chapter, a robust real-time collision-free path planning algorithm used for the horizontal 2D navigation of multirotor aerial robots in dynamic environments has been presented.

The objects of the environment are represented as geometric primitives, simplifying the information needed to describe the environment and therefore reducing the computational cost to determine the distance to the obstacles.

The arena is sampled using a probabilistic roadmap approach, generated at launching time and therefore without adding the obstacles to it, unlike the state of the art PRMs. This has two advantages: on the one hand, the space of the arena is covered more compactly than if using other approaches such as a grid map, but without loss of resolution. On the other hand, it allows handling moving obstacles efficiently since they are not included in the probabilistic roadmap graph.

A discrete search algorithm, concretely an A* algorithm is employed to calculate the collision-free path on the probabilistic roadmap graph. To speed up this search, and to be able to avoid the obstacles, a potential field map is used as the cost function to guide the search, that never falls in a local minimum.

Finally, a velocity and acceleration planning along the collision-free planned path is carried out, permitting the fast collision-free path following.

The performance of the proposed path planner has been widely demonstrated thanks to its previous intensive usage in two international aerial robotics competitions, several different self-proposed challenges, and various public demonstrations. Additionally, in this chapter, its performance has been evaluated with two simulations with complex environments including a labyrinth and dead ends, and with a real flight experiment where three fully autonomous aerial robots were executing an emulated search and rescue mission.

An analysis of the main limitations of the proposed path planner, together with the proposed future work to overcome them has been carried out.

The proposed path planner is completely available to the scientific community as an open-source software integrated into Aerostack.

The proposed robust real-time path planning for the collision free navigation of multirotor aerial robots in dynamic environments, presented in this chapter, can be found on (Sanchez-Lopez et al., 2017c).

Part IV

RESULTS AND CONCLUSIONS

Chapter 11

Additional Components Used in the Validation

It is presented in this chapter, with a low level of detail and only descriptively, four sets of components that have been implemented in Aerostack and that are used in Chapter 12, for the validation of the proposed system architecture and software framework. These components can also be used as illustrative examples of the abstract components defined in Chapter 4.

First of all, a set of controllers for multirotors are described in Section 11.1. Second, the set of task-based mission planners that have been explored are presented in Section 11.2. Third, Section 11.3 shows a proposal of a constraint-based action specialist. Finally, in Section 11.4, the multi-modal user interfaces implemented in Aerostack are presented.

11.1. Multirotor controllers

The three different kinds of controllers that have been explored and are used on the current version of Aerostack are the following:

- Low-level embedded controllers, analyzed in Section 11.1.1.
- Navigation controllers, described in Section 11.1.2.
- Visual servoing controller, presented in Section 11.1.3.

11.1.1. Low-level embedded controllers

These controllers have a very tight dependency on the used hardware and they typically have hard real-time constraints, working at very high rates (normally more than 1000 Hz). Because of these requirements, they are normally embedded in specific hardware components. An example of this kind of controllers are the motor speed controllers.

In the current version of Aerostack, it is assumed that the used aerial platform has these controllers available and working with an adequate performance in independent hardware components and thus, these controllers are not provided.

Motor speed controller

The motor speed (spinning velocity) controller, also called Electronic Speed Controller (ESC), controls the rotation speed of an electric motor given a rotation speed reference. The propeller, fixed to the motor, converts this motor rotation into a thrust and a torque applied to the robot body.

Normally, in commercial multirotors, the ESC are presented as independent hardware components (examples of commercial ESC are shown in Fig. 11.1), associating every motor of the multirotor with an ESC board. Therefore, quadrotors have four motors with four ESC, while octorotors have eight motors together with its eight ESC.

(a) Turnigy® Plush ESC. (b) Turnigy® AE ESC. (c) Turnigy® MultiStar ESC.

Figure 11.1: Different Turnigy® ESC models.

The typical hardware setup of an ESC (for example, those in Fig. 11.1) is the following: They have two sets of inputs: the energy coming from the battery (two cables); and the signal that encodes the rotation speed command, normally, set with a PWM (Pulse Width Modulation) protocol or a PPM (Pulse Position Modulation) protocol (three cables). Some other ESC manufacturers use a different hardware setup for the ESC inputs, and, for example, the motion command is given by an i2c (Inter-Integrated Circuit) protocol instead of PWM or PPM. Its output is the voltage command given to the motor (three cables in the case of using brushless motors).

11.1.2. Navigation controllers

In this section are included all the controllers that allow the navigation of the aerial robot, that is, the movement of the aerial robot on an obstacle-free space following different possible motion references such as a path, a position, or a velocity.

Due to the under-actuated nature of the conventional multirotors, only four out of its six degrees of freedom are controllable. Normally, for common navigation tasks, position and heading are controlled, being the other two degrees of freedom imposed.

In the current version of Aerostack, a cascaded set of different controllers is implemented, although the Motor System of the general system architecture does not limit to it.

Depending on how the controllers are configured, forming a cascaded configuration, different control modes are allowed, and therefore the controllers accept different motion references as inputs, ranging from paths in the most complex case, to forces and torques in the simplest. The output of this set of controllers is the input of the low-level controllers, previously described in Section 11.1.1.

The available control components can be grouped at the same time in five sets, and can be configured as represented in Fig. 11.2, depending on the type of the motion reference input:

- Wrench mapper.
- Attitude control.
- Position control.
- Point to look control.
- Path following control.

The reference frames involved in these controllers are shown in Fig. 11.3, and follows the standard robotics convention to define the reference frames[1][2]. The robot reference frame is rigidly attached to the robot, being the x-axis pointing forwards, the z-axis pointing upwards, and the y-axis forming a right-handed reference frame. The world reference frame is fixed to the earth, being the z-axis parallel to the gravity vector (in the displayed case, with the opposite direction).

[1]Online: `http://www.ros.org/reps/rep-0103.html`
[2]Online: `http://www.ros.org/reps/rep-0105.html`

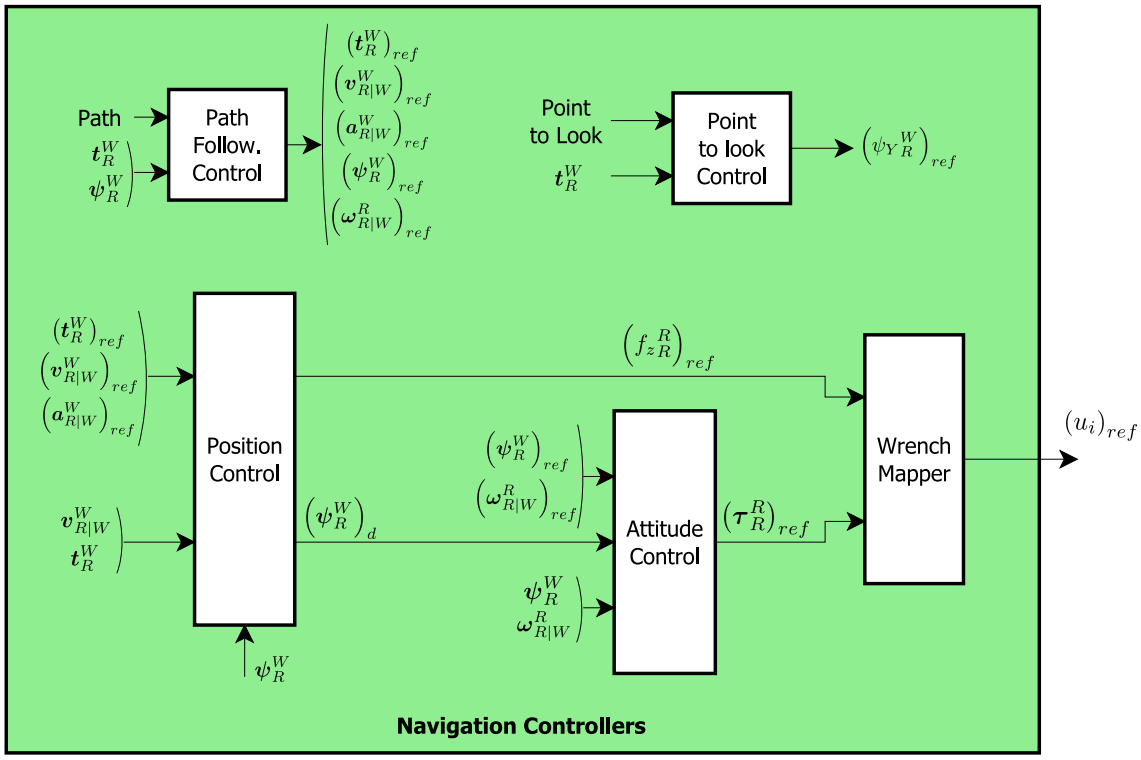


Figure 11.2: Cascade of control components that allows the aerial robot to move on an obstacle-free space.

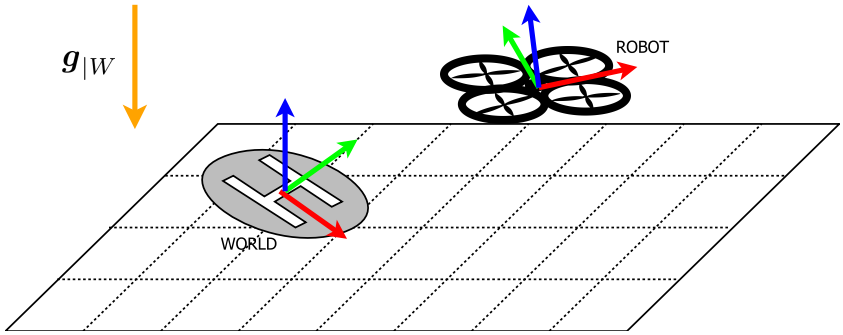


Figure 11.3: Reference frames involved on the navigation controllers.

Further details, as well as multiple results are out of the scope of this thesis, and can be found in the Master thesis (Pestana, 2012), and in (Mellado-Bataller et al., 2013b), (Mellado-Bataller et al., 2013a), (Pestana et al., 2013a), (Pestana et al., 2014b), and (Pestana et al., 2014a).

Wrench mapper

It transforms the desired force and torque of the center of mass of the aerial robot, $\left(\boldsymbol{f}_R^R\right)_{ref}$ and $\left(\boldsymbol{\tau}_R^R\right)_{ref}$, to the desired spinning velocity of the motors, $(u_i)_{ref}$.

This component has to take into account the number, type and configuration of motors together with the type of propellers that the aerial robot has. Therefore, it has a tight dependency on the hardware of the aerial robot.

Attitude control

It generates references of the torque of the center of mass of the aerial robot, $\left(\boldsymbol{\tau}_R^R\right)_{ref}$, required to follow the given references of attitude and angular velocity of the aerial robot, taking into account the current attitude and angular velocity of the aerial robot. Its working frequencies normally range between 100 and 1000 Hz, having hard real-time requirements.

Although it depends on the particular implementation of the attitude control component, it is sometimes divided in a cascaded controller formed by an angular velocity controller and an attitude controller.

Most of the commercial multirotor platforms include a hardware component, usually called autopilot, that, at least, has embedded the attitude control component and the wrench mapper component. This hardware component is able to directly send commands to the motor speed controllers without any extra integration effort by the user. Normally, this embedded attitude control component can be used with two main different control modes:

- Angular velocity control mode (also called rate control, manual control, or acrobatic control): the input motion references are the three-axis angular velocity, $\left(\omega^R_{R|W}\right)_{ref}$; and the total thrust, $\left(f^R_{zR}\right)_{ref}$.
- Horizontal attitude control mode (also called horizontal stabilization): the input motion references are the horizontal angles, $\left(\psi^W_{PR}\right)_{ref}$ and $\left(\psi^W_{RR}\right)_{ref}$; the vertical angular velocity, $\left(\omega^R_{zR|W}\right)_{ref}$; and the total thrust, $\left(f^R_{zR}\right)_{ref}$.

The current version of Aerostack does not provide any attitude control component or wrench mapper component, assuming that the used platform has a hardware autopilot with these embedded components working with an adequate performance. Thus, the autopilot component is, therefore, dealing with the hard real-time constraints.

Position control

It generates references of the thrust force of the center of mass of the aerial robot and the desired attitude of the aerial robot, $\left(\tau^R_R\right)_{ref}$ and $\left(\psi^W_R\right)_d$, required to follow the given references of position and linear velocity and acceleration of the aerial robot, taking into account the current position and linear velocity and acceleration of the aerial robot, as well as its attitude. Its working frequencies normally range between 10 and 200 Hz, and therefore it does not have hard real-time requirements, as the previously explained control components.

Although it depends on the particular implementation of the position control component, it is sometimes divided in a cascaded controller formed by a position controller and a linear velocity controller.

Some of the commercial autopilots have embedded a position control component that can be used with different control modes, depending on the input motion references, for example, linear velocity mode, or position mode. Nevertheless, as this is not a general fact, and it was even less common in the moment that Aerostack was firstly designed, Aerostack provides its own position control component. However, it is important to highlight that the general definition of the Motor System of the proposed system architecture does not enforce to use the existing controllers on Aerostack, being possible to use the embedded position control component of the used autopilot.

Point to look control

It allows to command the aerial robot to head a particular horizontal 2D point P of the world, given in world coordinates by $\boldsymbol{t}^W_L = \left[t_{xL}{}^W,\, t_{yL}{}^W\right]^T$, knowing the position of the aerial robot in world coordinates, $\boldsymbol{t}^W_R$. As the common multirotor platforms are underactuated, the aerial robot can only be commanded to head a horizontal 2D point, but never a 3D point.

The reference frames involved in this point to look control component are shown in Fig. 11.4.

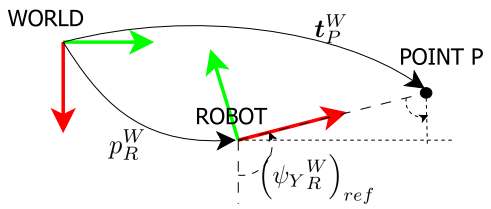


Figure 11.4: The reference frames involved in the point to look control component.

The heading references are given by:

$$\left(\psi_{Y\,R}{}^W\right)_{ref} = atan2\left(t_{xL}{}^W - t_{xR}{}^W,\, t_{yL}{}^W - t_{yR}{}^W\right) \tag{11.1}$$

Note that the commanded vertical attitude angle, $\left(\psi_{Y\,R}{}^W\right)_{ref}$, as $atan2$, is represented in the interval $[-\pi, \pi]$.

Path following control

It monitors the position of the robot with respect to a reference path, calculating the position reference of the robot that must be given to the position control, enabling the possibility to perform an accurate path following. Additionally, if the reference path includes references of velocity or acceleration of the robot, it forwards these references for the position of the path where the aerial robot is located.

11.1.3. Visual servoing controller

The visual servoing controller allows the aerial robot to follow an object in the 3D space, using as the motion reference, the information of the object in the image plane coordinates, thanks to an RGB camera, $\boldsymbol{f} = [f_u,\ f_v, f_\Delta]^T$, being f_u, and f_v the position of the center of a bounding box in the image plane; and f_Δ its size.

This controller accepts as references, the position of the center of a bounding box in the image plane, $(f_u)_{ref}$, and $(f_v)_{ref}$; and its size, $(f_\Delta)_{ref}$, that is, $(\boldsymbol{f})_{ref} = \left[(f_u)_{ref},\ (f_v)_{ref},\ (f_\Delta)_{ref}\right]^T$. The output of this controller is the desired position of the robot, $\left(\boldsymbol{t}_R^W\right)_{ref}$, and the desired heading of the robot, $\left(\psi_{Y\,R}^{\ W}\right)_{ref}$.

To perceive the object to follow, a calibrated RGB camera is mounted on the aerial robot. Specific components of the Feature Extraction System and the Situation Awareness System process the images given by the camera, detecting and tracking the object in the image plane.

This controller assumes that the object can be represented appropriately with a bounding box and that its movement can be adequately represented by a translation with scale movement in the image plane. In addition, the dimensions of the object are assuming to be known.

Fig. 11.5 represents the reference frames involved in the presented controller.

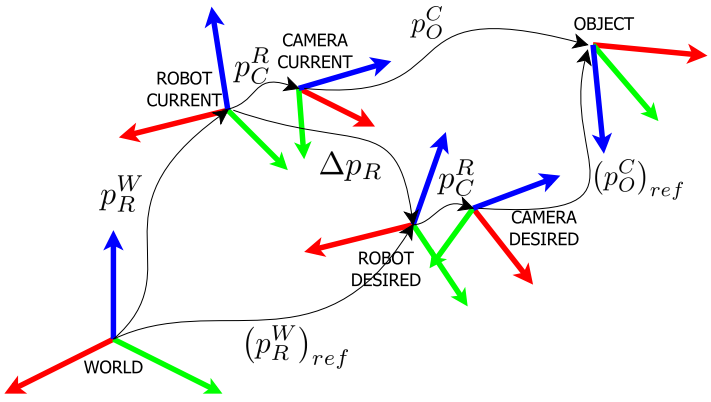


Figure 11.5: Reference frames involved on the visual servoing controller.

The structure of this visual servoing controller is represented in Fig. 11.6 and it has the following stages:

1. Computation of the error between the motion reference, and the feedback used by the controller: $\Delta\boldsymbol{f} = (\boldsymbol{f})_{ref} - \boldsymbol{f}$.
2. Decoupling of the desired robot movement using a heuristic and calculation of the increment in the position and heading of the robot. This step uses a proposed heuristic that depends on some parameters like the camera calibration parameters, and its pose in the robot, as well as the object dimensions, and the desired following distance. Some components of the state of the robot are as well used in this heuristic, such as the attitude of the robot, $\psi_{Y\,R}^{\ W}$. This step gives as outputs the desired increment on the position of the robot, $\Delta\boldsymbol{t}_R$, and the desired increment on the heading of the robot, $\psi_{Y\,R}$.
3. Calculation of the position and heading commands that will be sent to the navigation controllers (described in Section 11.1.2), knowing the current position and heading of the robot.

Further information about this controller and its results are out of the scope of this thesis and can be found on (Pestana et al., 2013b), and (Pestana et al., 2014c).

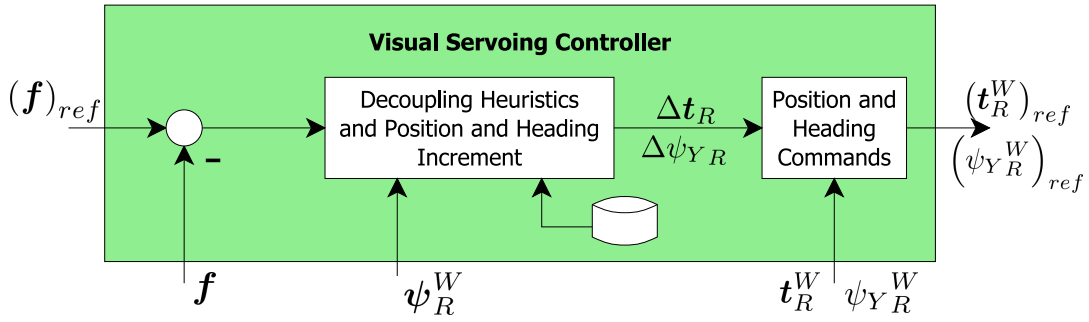


Figure 11.6: Visual servoing controller.

11.2. Mission planners

Different task-based mission planners, that decompose a complete mission in a task tree, have been explored, and are available in the current version of Aerostack. Tasks are defined as a basic component to structure a mission with a modular organization. At the same time, every task might be decomposed in another simpler task tree. An important requirement is that every task tree always ends with actions (either executive or deliberative).

Task-based sequential mission planner

The first mission planner available in Aerostack (presented in (Sanchez-Lopez et al., 2013a), (Sanchez-Lopez et al., 2014a), (Pestana et al., 2014e), (Sanchez-Lopez et al., 2016b), and (Pestana et al., 2016)) proposes to use a basic task-based mission planner, whose missions have a predefined sequential behavior. Missions are defined as a task tree by the operator using a particular language using XML syntax to be readable by both humans and machines.

The mission definition includes the possibility to use while-loop, for-loops, and conditional clauses allowing to change the mission flow and reacting therefore to some monitored events. An example of a while-loop is the following: while battery level is greater than a specific value, keep doing the current task. An example of a for-loop is the following: perform a particular task a specific number of times. An example of a conditional clause is: if a specific visual marker is detected, perform a particular task.

The capability to interact with dynamic environments that might change the execution of a mission is limited to the mission definition given by the operator.

This proposed mission planner has demonstrated to be very useful for the execution of simple missions with static (or pseudo-static) environments. Most of the commercial mission planners used in aerial robotics are based on simpler versions of the proposed mission planner.

A second mission planner available in Aerostack, presented in (Molina et al., 2016), is an extension of the task-based approach together with reactive planning (using event handlers formulated with rules) to facilitate a more flexible specification of plans to be adaptive to a dynamic environment. This representation is supported by a formal language, the TML language (Task-based Mission specification Language), that incorporates deliberative terms about actions and skills that were not considered in previous planners of Aerostack This language formalization creates a new simpler but complete grammar and vocabulary that is verified after the mission definition by the operator, allowing to correct errors, giving robustness to the mission planner.

Another improvement that simplifies the mission generation to the human operator is the usage a graphical user interface that automatically generates the XML file from easier commands given to the interface (for example by means of clicks on buttons).

Task-based dynamic mission planner

The third mission planner available in Aerostack (see (Sampedro et al., 2016)) proposes a task-based mission planner able to work in dynamic environments.

The operator only has to define the high-level mission without specifying the complete mission tree, but only high-level tasks. These high-level tasks (called behaviors) are internally represented as a task tree that might be called by the operator and have been included in the mission planner. Examples of these behaviors are: explore an area (that requires to sample the area to explore and concatenate several navigation actions to a point *P* actions); or find an object (that requires to explore an area until the object is found, including specific stops to look for the object with the onboard camera).

Together with the behaviors, a set of restrictions and conditions that have to be satisfied for every task, have been included in the mission planner. Examples of the restrictions and conditions are: before moving to point P, a take-off is needed; or, when trying to navigate to a point P_A, if it is occupied, navigate to a point $P_{A'}$ in the neighborhood of the point P_A.

The complete mission tree can be generated from a very simple set of tasks and behaviors. The proposed restrictions and conditions allow simplifying the mission definition since the task restrictions are automatically added to the task tree. The missions and tasks are therefore automatically nested, creating complex missions without the operator intervention. Additionally, it allows interacting with dynamic and changing environments.

Moreover, the proposed mission planner is suitable for its usage as a centralized multi-agent mission planner (as described in Section 3.3), as it is able to automatically decompose a mission in several tasks, having the capability to assign them to different agents.

11.3. Action specialist

A constraint-based action specialist, deeply presented in (Molina et al., 2016), has been explored, being available in Aerostack.

For some feasibility checks, the proposed action specialist incorporates approximate models with a constraint-based representation. This model uses variables, parameters, functions and constraints.

Variables x_i represent the dynamic values of physical references and magnitudes (e.g., destination point, current charge of the battery, etc.).

Parameters k_i represent constant values for physical magnitudes related to the performance of the robot such as maximum speed, battery consumption rate of the vehicle, etc. Parameters can be divided into vehicle-independent parameters (general for any kind of vehicle) or vehicle-specific for each category of vehicle.

Functions $f_i(x_i,k_i)$ represent spatio-temporal and motion functions (see Table 11.1) such as the length of a path, distance to the closest obstacle, maximum distance covered with certain battery charge, required speed to reach a point at certain time, etc.

Function	**Description**
$distance(x_1,x_2)$	Distance from point x_1 to point x_2
$distanceBattery(x_1,k_1)$	Maximum distance covered with battery charge x_1 and consumption rate k_1
$distanceObstacle(x_1)$	Distance from point x_1 to the closest obstacle
$length(x_1)$	Length of path x_1
$path(x_1,x_2)$	Path from point x_1 to point x_2 (generated by the path planner)

Table 11.1: Example functions used in the verification model.

Constraints c_i are conditions about the physical world that must be satisfied. Examples of these conditions are: c_1, the destination point must be safe from obstacles; and c_2, there must be enough battery for the movement. The previous sample conditions are represented with the following two constraints (using functions, variables and parameters):

$$c_1: \quad distanceObstacle(x_2) > k_2$$
$$c_2: \quad length(path(x_1,x_2)) < distanceBattery(x_3,k_1)$$

where the variable x_1 is the current point, x_2 is the destination point, and x_3 is the current battery charge; and the parameter k_1 is the battery consumption rate, and k_2 is the minimum free acceptable space between obstacle and vehicle.

This type of model is generic to be reusable for different physical platforms. Only the vehicle-specific parameter values must be configured.

11.4. Multi-modal user interfaces

The current version of Aerostack implements three kinds of user interfaces: (1) Command Line Interfaces (CLI), (2) Graphical User Interfaces (GUI), and (3) Natural User Interfaces (NUI). The first ones are omitted due

to their simplicity, while the GUIs are briefly described in Section 11.4.1, getting afterward into deeper details with the NUIs in Section 11.4.2.

A deeper review of these multi-modal user interfaces can be found in (Suárez Fernández et al., 2016).

11.4.1. Graphical User Interfaces (GUI)

In general, a GUI provides some functions to help operators in certain tasks that are difficult to be supported by a NUI. For example, they correspond to tasks where the operator requires detailed information such as vehicle set up or mission monitoring at a software level (e.g., during software maintenance).

The GUI allows the interaction with the agents, observing its states and dynamics and presents graphical views and images to help the user to understand both the external and internal behavior of the vehicle.

In general, the operator can use a GUI to perform the following types of tasks:

- Specify the robot behavior in advance (vehicle set up)
- Monitor the robot behavior during a mission
- Operate manually with simple movements
- Collect data for later use

Fig. 11.7 shows a sample screen of the user interface with the windows layout divided into the following main parts:

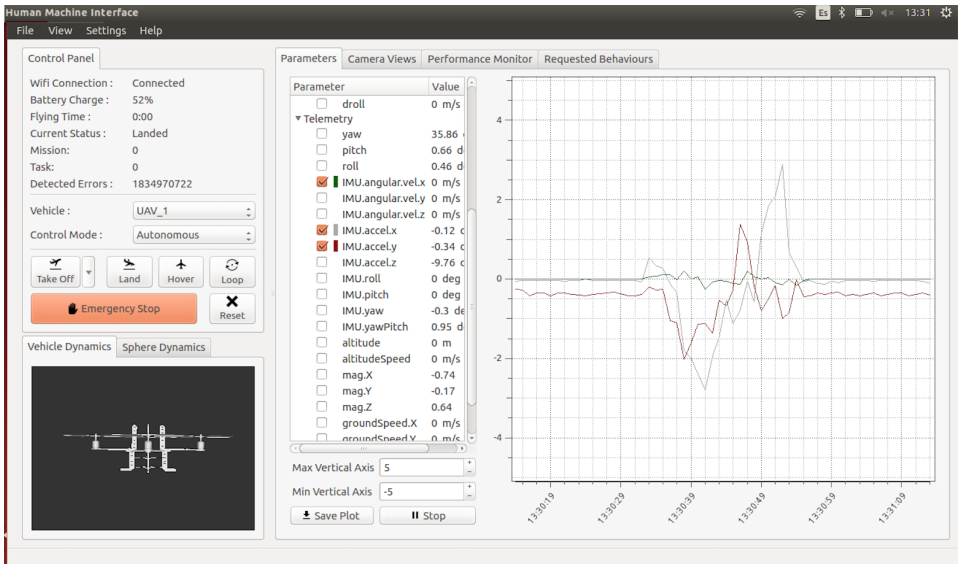
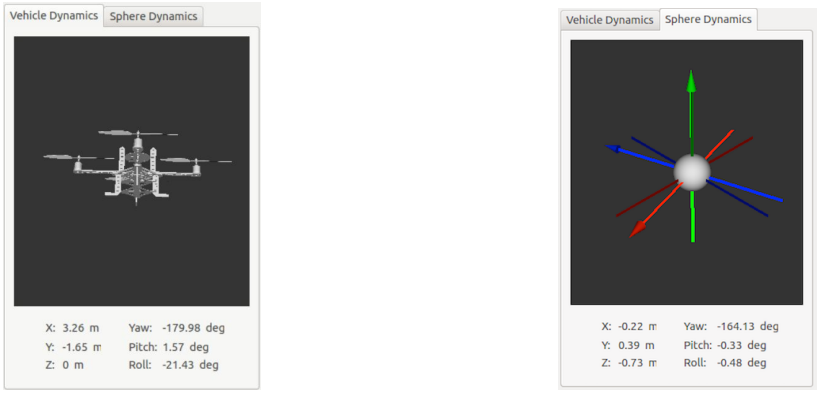


Figure 11.7: Sample screen of the proposed graphical user interface.

- *Control panel*, upper left side: The control panel shows the general state of the system and the main control commands. This panel provides a summary of information considered critical that needs to be permanently present on the screen.
- *Detailed contents panel*, right-hand side: It allows the user to get further details of the components and processes of the aerial robot. It includes different tabs, for example:
  - *Parameter viewer*: In order to monitor the vehicle's behavior in detail during a mission, the operator can observe the values of numerical parameters and display multiple plots of the parameters selected in real time to analyze and compare their values.
  - *Camera viewer*: The camera viewer shows pictures and/or video images captured by the aerial vehicle during flight.
  - *Requested behaviors viewer*: The operator can use the behavior viewer to consult and request the activation of specific behaviors. The GUI shows a list of available behaviors for the vehicle and indicates for each one its state.
- *The dynamics viewer*, lower left side: The GUI includes a viewer that shows in animated 3D views the dynamics of the vehicle. This can be used by the operator, for example, to better guide the vehicle during manual operations of the vehicle with the keyboard.
  This viewer presents two animated images as seen in Fig. 11.8: (1) the vehicle dynamics, a 3D representation of the vehicle (Fig. 11.8a) and (2) the sphere dynamics, a sphere with orientation axis



(a) The Vehicle Dynamics.

(b) The sphere Dynamics.

Figure 11.8: The dynamics viewer.

(Fig. 11.8b). In the sphere representation, there are three fixed axis that represent the reference system, and three variable axis that represent the orientation changes.

- *Drop-down menus*, top: the GUI includes menus with options such as file, view, settings, etc. that allow performing additional tasks.

11.4.2. Natural User Interfaces (NUI)

Despite the functionality of the GUI, there are operator tasks where a NUI can provide a more efficient communication compared to a conventional GUI. Natural communication can include using gestures to guide the vehicle during manual operation, which are easier to learn, or voice commands that can be used more efficiently in combination with other communication modes.

In general, a communication based on NUI can be especially useful in human-robot cooperative work for certain complex missions in dynamic environments where the partial information used by the vehicle can be complemented with human decisions.

In the following sections, a description of the types of NUIs implemented for each interaction will be given. First, the vision based NUIs will be explained, starting with *markers* and followed by *body* interaction. Afterwards, *hand* interaction is described, finishing with *speech* interaction.

Visual marker interaction

Visual cues of color, depth, and motion are in many species, specially in humans, a large source of information in how the world is perceived (Posner et al., 1976). Using visual cues or markers is not an uncommon practice in robotics since these take use of arguably the most important sensor in robotics, the camera.

In this type of interaction, represented in Fig. 11.9, the user manipulates visual markers to *command* the aerial robot what to do, and therefore a non-expert user could pick up a predefined set of visual command markers and interact in a safe and entertaining way. Since no additional device apart from the onboard camera is needed, the interaction is effortless and the user feels integrated into the decision-making process of commanding the aerial robot. The visual markers allow their robust and accurate detection and precisely pose estimation, as (Dudek et al., 2007) suggest. These markers rely on a specific pixel pattern that uniquely encodes information which is needed for the detection algorithms to operate.

The command process is as follows: when the markers appear in the onboard cameras' field of view, they are detected and identified using the vision processing algorithms of the Feature Extraction System. Depending on the identifiers of the detected markers, different events are sent via the Situation Awareness System for the aerial robot to perform actions such as *take-off*, *hover*, *land*, etc.

Visual body interaction

As presented in Fig. 11.10, the aerial robot, equipped with an onboard camera, uses computer vision algorithms from the Feature Extraction System to detect a person and track it in the image plane. A visual

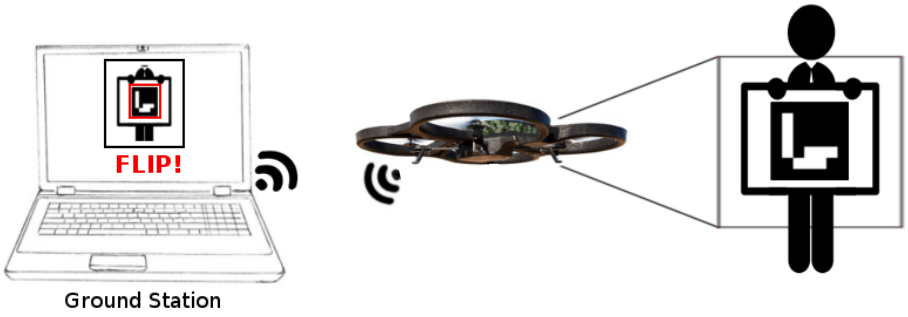


Figure 11.9: High-level description of the visual marker NUI.

Figure 11.10: High-level description of the visual body NUI.

servoing control algorithm of the Motor System (described in Section 11.1.3) sends commands to the aerial robot ensuring it maintains the distance and point of view (heading angle) to the person.

This NUI demonstrates one of the most primitive behaviors of animals. Baby animals, by instinct, follow their parents everywhere, this interaction is similar to how the presented aerial robot is behaving.

Hand gesture interaction

Many solutions have used depth cameras like the Kinect for the hand gesture recognition, as discussed in Section 2.8, nevertheless, sensors like the Leap Motion Controller which is explicitly targeted for this, remain unexplored. This sensor directly computes the position of fingertips and hand orientation used for control, giving a more intuitive flight experience to the user with the palm of their hand.

The Leap Motion controller is an 8 cm long USB connected sensor designed to track hand and finger motions in a small working space. The device is intended for consumer use and requires minimal setup on the host computer. This provides accessible means of controlling practically any interface with the palm of your hand. The sensor works by projecting infrared light upward and detecting reflections using monochromatic infrared cameras. Its field-of-view (FOV) extends from 25 mm to 600 mm with a 150° spread from the device, with a frame-rate of roughly 200 fps and a precision of $1/100$ mm per finger (Han and Gold, 2014).

The proposed hand gesture NUI is represented in Fig. 11.11. The Leap Motion is used to compute the orientation and position of the user's hand relative to its own axes.

After conducting several experiments, the results show that a direct transformation between the movement of the hand and the movement of the aerial robot felt more instinctive for users, as represented in Fig. 11.12.

Speech interaction

Interaction by means of speech offers hands/eyes-free interfaces, what can enhance the user's experience by being able to devote all their visual attention at commanding tasks to the aerial robot. The development of an effective speech command interaction requires prior in-depth knowledge of the tasks that can be performed and who the end-user will be. These interfaces need to respond to input reliably or they may be rejected by the user. As stated in Section 2.8, few examples of speech interaction with aerial robots can be found in the literature.

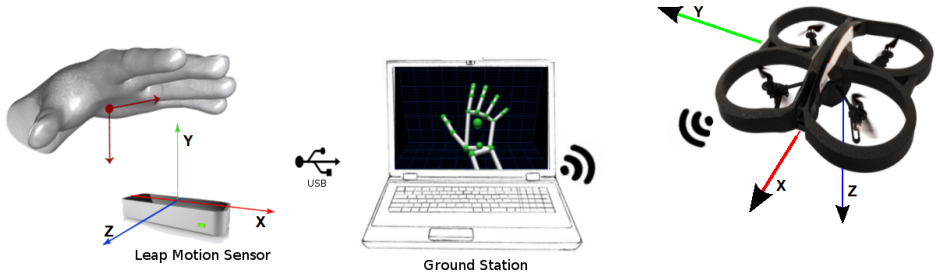


Figure 11.11: High-level description of the hand gesture NUI.

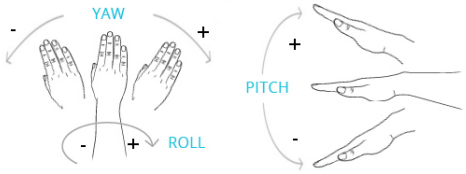


Figure 11.12: Hand gesture commands used for aerial robot flight. In this setup positive pitch, roll and yaw rate commands move the quadrotor backwards, right and clockwise respectively.

Fig. 11.13 shows the high-level setup of the proposed speech command NUI. No devices other than a microphone and speakers are needed to command tasks to the aerial robot. Voice commands are sent to the Ground Station to be converted to actions that the aerial robot is able to perform. Additionally, the aerial robot provides a speech feedback of the action that is executing.

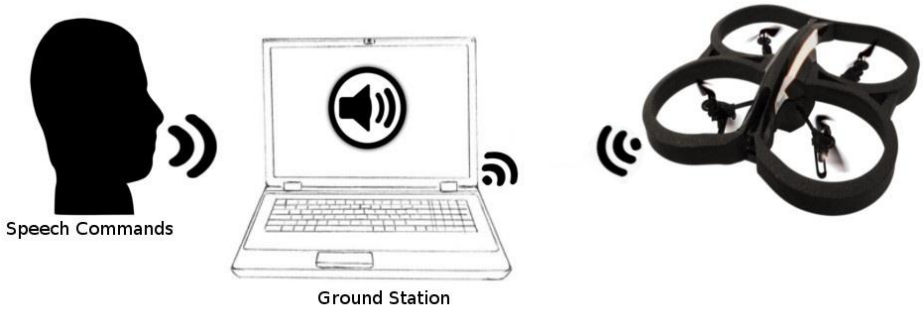


Figure 11.13: High-level description of the speech NUI.

Voice processing is done using the ROS package implementation of the Pocketsphinx library. The CMU Pocket Sphinx speech recognizer is the general term to describe a group of speech recognition systems based on hidden Markov models (HMM's) developed at Carnegie Mellon University (Lee et al., 1990). This package automatically splits the incoming audio into utterances to be recognized. Currently, the recognizer requires a language model and dictionary file that can be automatically built from a corpus of sentences using the Online Sphinx Knowledge Base Tool[3]. This software allows to easily integrate new voice commands in order to expand the tasks or behaviors required for the aerial robot.

A grammar of approximately fifteen commands has been developed that include, but are not limited to, *move forward*, *move backward*, *rotate right*, and the like. The software package listens for these simple one to three-word tasks, and when a positive detection is made, a voice synthesizer was implemented to offer acknowledgment of the action in the present continuous tense: *taking off*, *moving forward*, etc.

[3]Online: `http://www.speech.cs.cmu.edu/tools/lmtool-new.html`

Chapter 12

Evaluation, Applications and Results

12.1. Evaluation methodology

The determination of the degree of achievements of the proposed objectives in Section 1.4 is a complex task since the performance on the fully autonomous intelligent navigation of a heterogeneous aerial multi-robot system is highly dependent on the kind of the mission executed, the environment where the system has to move, the hardware setup, the algorithms employed, and their software implementation.

To provide different metrics of the quality and performance of the work presented in this thesis, the evaluation results are gathered into three different groups:

- local results on the proposed algorithms performing individually.
- global results of the presented system architecture, software framework, and algorithms, performing as a complete system executing the missions requested by international competitions, self-proposed challenges, and public demonstrations.
- metrics related to the software framework.

Local evaluation of the proposed algorithms

The proposed algorithms, presented along Part III, require an individual evaluation to be able to determine their performance, minimizing the interferences on their execution of the rest of the components and systems. The local evaluation of these proposed algorithms is done along Part III.

These proposed algorithms aim to solve a particular problem, and therefore, they are highly dependent on the complete application, including the kind of mission and its environment, and the proposed hardware setup. Thus, they are only useful if similar conditions than the designed one are met. Their performance is as well conditioned to their software implementation (e.g. a bad coding of the algorithm might make it running slowly and inefficiently).

Despite being evaluated in an isolated way, the importance of each component appears when integrated into the complete system. That means that individually, some of the proposed algorithms are simple and they present limited results when compared with the most advanced state of the art counterparts, but when integrated as a part of the complete system they allow an outstanding performance of the complete system.

Global evaluation of the complete system

The presented system architecture, software framework, and proposed algorithms are combined into a

complete fully functional system capable of solving different challenges.

The global evaluation of the complete system is done by means of the participation in international aerial robotics competitions (Section 12.2), performing several self-proposed challenges (Section 12.3), and exhibiting it in multiple public demonstrations (Section 12.4).

The advantages of the participation in international competitions are multiple. In the one hand, the fact that the mission definition and how it is evaluated, the mission environment, the general rules, and the competition date are imposed externally by the competition organizers, provides a rigid challenge that has to be completed with no possible modification. As the complete system has to efficiently work the competition day, the system has to be, no only fully functional, but also extremely reliable. In the other hand, it allows a comparison of the complete proposed system with the rest of the teams that are using a different approach to complete the proposed challenge. Finally, an external evaluation of the proposed system is done by the referees with qualitative and quantitative measures in the form of awards.

The self-proposed challenges different than international competitions allow pushing the system to its functionality limits. As they are not externally given, their objectives might be adapted to the particular research line that is wanted to be investigated, with an unlimited level of difficulty. Normally, these challenges target complex applications that require being solved with a high level of autonomy. They do not present a hard deadline, and therefore, the reliability of the system is a minor requirement. Moreover, as they are carried out in lab experiments, high requirements on safety are not needed.

Finally, public demonstrations, are a very demanding evaluation. They require a fully functional, reliable and safe system since it has to work for the demo without failures, with special attention on safety because hundreds of people could be injured. The mission to be completed and its environment can be controlled as they are not externally given, nevertheless, they have to be challenging enough to cause surprise and satisfaction to the public. The audience could be very variate, ranging from kids and teens, who are easily surprised but they are not aware of risks of an aerial robot and therefore they impose a challenging requirement on safety; non-specialized adults, who are easily surprised and they are very conservative with their safety; and specialized adults, who are aware of the possible risks of an aerial robot, but expect a very challenging demonstration. Additionally, the system is required to be demonstrated to mentally disabled people who act as kids and teens, but with the added difficulty of a very unpredictable behavior and a hard difficulty to control them.

Software metrics

The last group of results shown are the software metrics presented in Section 12.5. This collection includes several quantitative pieces of evidence that demonstrate some performance and quality features of the software framework.

Simulations and real experiments

Both local and global evaluation are done by means of two kinds of tests: simulations and real experiments.

Simulations are carried out in the first steps of the evaluation stage and provide a general overview of the performance of the proposed solution under controlled conditions and being capable of isolated some elements. This kind of tests allows ignoring some technological and economical problems, to concentrate on the difficulties of the adopted solution.

Real experiments are required on every system that is supposed to work in the real world. This kind of tests evaluates the proposed solution in the presence of technological and economical limitations.

Media

Multiple experiments are shown in the following sections of the present chapter and in along Part III, mainly as images, tables, and plots. Nevertheless, the reader is encouraged to watch the complementary videos associated to this thesis and gathered in Appendix E.4 to have a complete overview of the performance of the system when performing a particular experiment.

Comparison with the state of the art

Before finishing the statement of the evaluation methodology, it is important to mention the comparison of the complete system and their components with their state of the art counterparts.

In the one hand, the comparison of the performance of a specific proposed algorithm with its state of the art counterpart is done along Part III, when evaluating every component individually. However, as stated before, the importance of every proposed algorithm appears when integrated into the complete system and therefore, sometimes the comparison with the state of the art counterparts is omitted.

In the other hand, the comparison of the performance of the complete system against its state of the art counterpart is a more difficult task. A qualitative comparison, highlighting the main weaknesses of the state

of the art complete systems, has been done when reviewing the state of the art in Chapter 2. A higher level quantitative comparison is as well done by means of the participation in international competitions, being the achieved final results a great evidence of the performance of the complete system compared with the state of the art ones. Nevertheless, a deeper quantitative comparison of the performance of the presented complete system with its state of the art counterparts is not feasible. The reasons for this are mainly the following:

- different hardware setup proposed by the state of the art systems for the execution of a particular mission, that might be unavailable in the research group.
- the absence of an open-source software implementation of the state of the art complete systems.
- the different missions or environments that might be targeted by a particular state of the art system might not be reproducible in the research group.

12.2. International competitions challenges

This section describes the participation of the research group on international competitions, being the following:

- IMAV 2012, Section 12.2.1.
- IMAV 2013, Section 12.2.2.
- IARC 2014 (Mission 7), Section 12.2.3.
- IMAV 2016, Section 12.2.4.

The structure of all these sections is always the same: firstly, the competition is briefly described; then, the proposed system setup (hardware and system architecture) is explained; after that, the achieved results are shown; and finally the related publications are listed.

12.2.1. IMAV 2012

Competition description

The 2012 International Micro Aerial Vehicle Competition (IMAV 2012), that took place in July 2012 in Braunschweig (Germany), had several categories. The research group participated in the category “Indoor Flight Dynamics - Rotary Wing MAV”.

In this challenge, a rotary wing micro aerial vehicle had to perform as many loops as possible in the arena represented in Fig. 12.1 during 3 minutes.

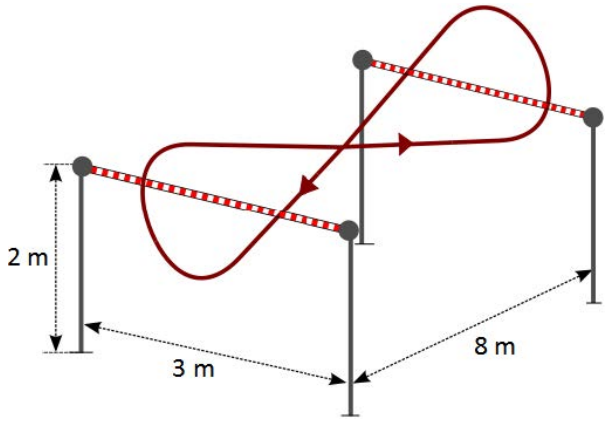


Figure 12.1: IMAV 2012 - Indoor dynamics competition arena. Figure extracted from the IMAV 2012 competition rules.

The total score achieved was given by the following equation:

$$T = S \cdot A \cdot D \cdot M \tag{12.1}$$

being:

- S, the size factor calculated with respect to the maximum dimension D_b (in cm) between two points on the vehicle with: $S = (2 - D_b/100)^2$.

- A, the level of autonomy, where $A = 1$ if video-based, $A = 6$ if automated flight control, and $A = 12$ if automated mission control. If any external aid was used, $A = A - 2$.
- D, the design factor, being $D = 1$ if a commercial aircraft is used, and $D = 4$ if a self-made aircraft is used.
- M, the mission score, adding 2 points to the mission score M for each fully completed figure eight in between the barrier. An extra 2 score points were rewarded once for showing dynamic maneuvers such as a looping or a 360° roll.

System setup

Hardware setup

An AscTec Pelican quadrotor was used. This quadrotor was equipped with four ESC and with the AscTec autopilot, an autopilot board that stabilizes the vehicle using the information from IMU, pressure altimeter and magnetometer fused by means of an internal Kalman filter, providing as well an estimate of the aerial robot attitude, its angular velocities, and its linear accelerations. This stabilization controller and the state estimator are embedded, closed, and unmodifiable but gains are tunable. The autopilot allows to command the vehicle in attitude mode for the horizontal angles (pitch and roll), in velocity mode for the vertical angle (yaw), and the average thrust command given by its motors.

This quadrotor was also equipped with a series of complementary sensors: a Hokuyo Scanning Laser Range Finder for horizontal depth mapping, a sonar altitude sensor for flight altitude estimation and a downward looking camera for horizontal velocity estimation based on optical flow.

The quadrotor was, as well, equipped with an onboard computer, an AscTec Atomboard, which has a dual-core Atom 1.6 GHz processor with 1 GB of RAM, where all the state estimation and control processing was executed, and that was connected to the AscTec autopilot by means of a serial cable. The Atomboard computer was connected by means of a Wi-Fi network to a ground computer for operator monitoring and supervision.

System architecture setup

The system architecture employed in this competition was relatively simple (see Fig. 3.3), compared with the complete functionality provided by the proposed system architecture of Chapter 3, being formed by the following pseudo components:

- *Aerial platform and autopilot*: As mentioned, the AscTec Pelican incorporates the low-level embedded controllers described in Section 11.1.1. Additionally, the attitude control of the navigation controllers presented in Section 11.1.2 was provided by the AscTec autopilot. Moreover, as stated, the autopilot estimates the aerial robot attitude, its angular velocities, and its linear accelerations.
- *Mid-level controller*. The position and path following control of the navigation controllers presented in Section 11.1.2 were used.
- *High-level controller*.
  - *Trajectory planning*. The velocity and acceleration along the path were calculated by means of a component similar than the one presented in Section 10.1.7.
  - *Mission planning and execution*. A component with some of the responsibilities of the current Supervision System and the Planning System was included to supervise the autonomous operation of the aerial robot and to schedule the actions that needed to be commanded depending on the mission execution. It was implemented as a finite state machine.
  - *Action sequencing*. Some basic actions like take-off, land, hover, move in velocity, move in position, and move in path following were included in an incipient Executive System.
- *Perception components*.
  - *Horizontal velocity*. A component to compute the horizontal velocity of the aerial robot based on optical flow was included. The optical flow was calculated using the OpenCV implementation of the pyramidal Lucas-Kanade algorithm for some image features selected by the Shi-Tomasi method. Currently, several sensors, like the px4flow directly provide an estimation of the horizontal velocity and flight altitude, but by the time of this competition, these sensors were not available.
  - *Odometry based state estimation*. The odometric state of the aerial robot was estimated by means of a similar component than the one proposed in Section 7.2.
  - *Global localization*. The global pose of the aerial robot in the arena was calculated using a particle filter and the measurements given by the LIDAR.
- *Communication*. A relatively complex Communication System, described in (Mellado-Bataller et al., 2013b), (Mellado-Bataller et al., 2013a), was implemented.

The reader must note that by the time of this competition, the existing system architecture was not related to the system architecture presented in this thesis, but it has been included here because it served as the inspiration

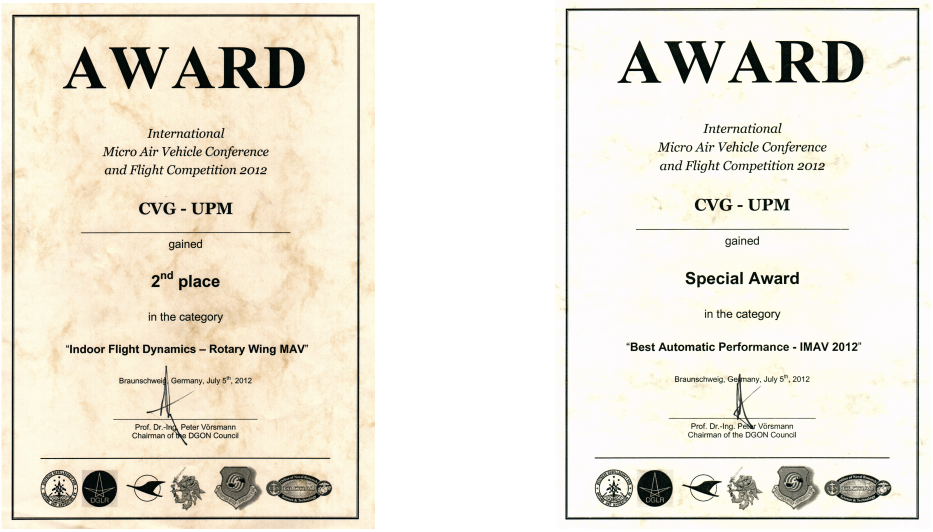
for the development of the presented system architecture, and some of the components built for this competition were reused.

Achieved results

Competition results

The competition results can be checked online in `http://www.imavs.org/imav2012results/`.
Two awards were obtained in the competition:

- The second place in the category Indoor Flight Dynamics - Rotary Wing MAV (Fig. 12.2a).
- A special award in the category Best Automatic Performance - IMAV 2012 (Fig. 12.2b), since the research group was the only team performing the complete mission with the highest level of autonomy.



(a) 2nd place in the category Indoor Flight Dynamics - Rotary Wing MAV.

(b) Special Award in the category Best Automatic Performance - IMAV 2012.

Figure 12.2: The two awards obtained in the IMAV 2012 Competition.

As the participation of the author of the thesis in this competition was minor, the results related to this competition has been only briefly presented.

Related publications

More information about the components used in this competition in addition to some quantitative results verifying the performance of all the components working together as a complete system can be consulted in (Pestana, 2012), (Pestana et al., 2013a), (Pestana et al., 2014a), and (Pestana et al., 2014b).

12.2.2. IMAV 2013

Competition description

The 2013 International Micro Aerial Vehicle Competition (IMAV 2013), that took place in September 2013 in Toulouse (France), had several categories (more information about the competition can be consulted online: `http://www.imav2013.org/`). The research group participated in the category “Indoor Autonomy”.

In this challenge a fleet of aerial vehicles had to navigate in the environment represented in Fig. 12.3 during 10 minutes.

The mission consisted of multiple mission elements that could be performed in any order by one or more MAV. Not all the mission elements were targeted to be executed in the research group competition strategy. The ones affected were the following:

- Takeoff and hover (1)
  - a takeoff was performed from the starting zone.

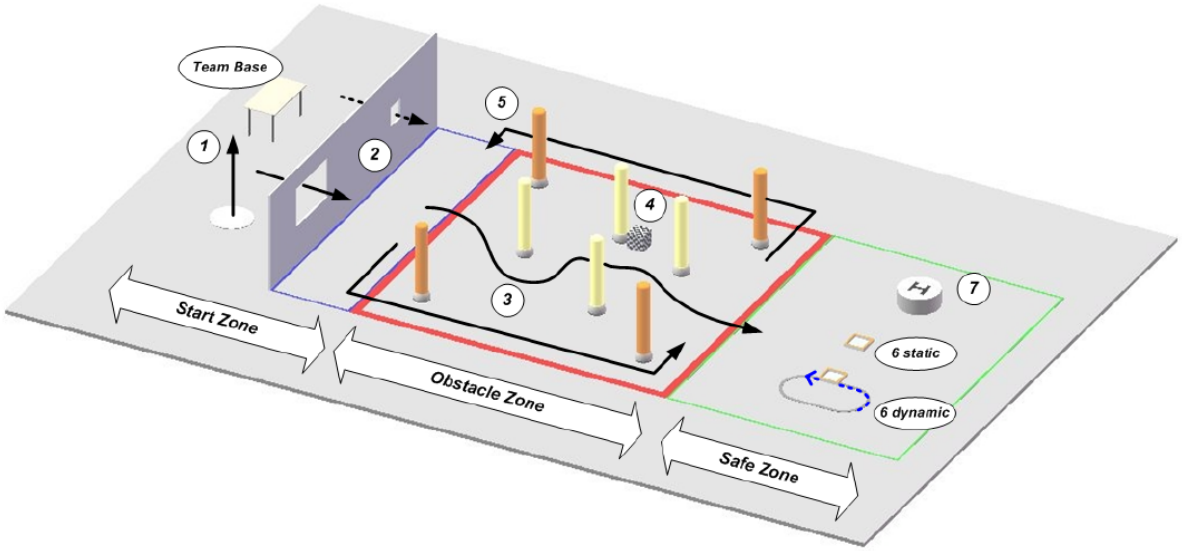


Figure 12.3: IMAV 2013 - Indoor autonomy competition arena. Figure extracted from the IMAV 2013 competition rules.

- after taking off, the MAV had to hover for at least 5 seconds within the takeoff area (circle of 2 meters diameter).

- Flying through the window (2)
  - the MAV had to pass through one of the two different size windows existing on the wallÍž higher scores were awarded for flying through the smallest window.
- Flying through the obstacle zone (3)
  - several poles were placed in the flying area and the MAV had to cross the area to reach a safe zone at the other side.
  - four fixed poles marked the corner of the obstacle zone, four others had unknown positions.
- Follow a path (5)
  - The designated path was laid out around the four fixed poles which marked the obstacle zone.
  - the score depended on the number of laps completed flying over the designated path.
- Precision landing (7)
  - the MAV has to perform a precision landing on a small platform.
  - extra points were awarded if the MAV was able to take off again after staying still 10 seconds on the ground and without any operator intervention.

The total score achieved was given by the following equation:

$$T = \sum_{\forall j} \left(\sum_{\forall i} (M_{ij} \cdot A_{ij}) \cdot S_j \right) \cdot P \tag{12.2}$$

being:

- M_{ij}, the score of the i-th mission element, executed by the j-th MAV.
- A_{ij}, the level of autonomy during the execution of the i-th mission element by the j-th MAV.
- S_j, the size factor of the j-th MAV, given by $S_j = 1/D_j$, where D_j is the maximum horizontal distance of a rotary wing (including blades) and it was limited to 1 m. Additionally, the MTOW was limited to 2 kg.
- $P \in [1.0,\ 1.1]$, the presentation factor.

The level of autonomy A_{ij} was given by: (1) video based control $= 1$; (2) assisted flight control $= 4$; (3) autonomous flight control, and autonomous target detection $= 6$; (4) fully autonomous mission control 12. Additionally, if any external aid was used, $A_{ij} = A_{ij} - 2$.

The main mission elements M_{ij} that affected to the scoring of the research group strategy were given by:

- Takeoff (1): 1 per MAV.
- Fly through window (small, big, none) (2): 2, 1, 0 per MAV
- Fly through obstacle zone (3): 1 per MAV
- Path following (5): number of laps (per MAV)
- Landing (precision) (7): 1 per MAV. Extra points: $+1$ if taking off after 10 seconds staying still without any operator assistance

System setup

The competition strategy followed was to use a fully autonomous fleet of relatively simple aerial robots. The selected aerial platform was an out of the box Parrot AR.Drone 2.0. As this platform had a limited payload, some mission elements (like dropping) were discarded. Additionally, all the processing (apart from the out of the box onboard autopilot) was required to be done on a ground computer. Moreover, due to the limited number of sensors available on the aerial robot, to self-localize it and to perceive the environment, external aids were used. These external aids were ArUco visual markers that were placed on the poles and on the wall.

Hardware setup

As stated before, a fleet of out of the box Parrot AR.Drone 2.0. was used (see Fig. 12.4).

This aerial platform has two cameras (a front camera and a bottom camera), together with an IMU, a magnetometer, a barometer, and a down-looking ultrasound sensor. It includes an autopilot that estimates the aerial robot attitude (by means of the IMU and the magnetometer), its flight altitude (thanks to the ultrasound sensor and the barometer), its horizontal linear velocity (calculating the optical flow with the bottom camera), and its angular velocities and linear accelerations (filtering the IMU). The autopilot allows to command the aerial platform in attitude mode for the horizontal angles (pitch and roll), and in velocity mode for the vertical angle (yaw) and for the vertical movement (z). The autopilot includes as well some actions (and the required controllers to execute them) like take-off, hover and land.

Figure 12.4: Parrot AR.Drone 2.0. during the IMAV 2013 competition.

Every Parrot AR.Drone was connected using an ad-hoc Wi-Fi network to a ground computer. All the ground computers were connected to a local wired network by means of a switch (see Fig. 12.5). Due to the limited payload of the Parrot AR.Drone, using an onboard computer was a challenging task and therefore discarded. The usage of a ground computer per aerial platform conferred the system the purely distributed essence. Nevertheless, as the proposed system was relying on ROS as the communications middleware, and ROS has a centralized nature, a master computer (executing the ROS core) was required. This master computer was added to the network as a network master computer.

System architecture setup

A fleet of fully autonomous similar agents was used. The system architecture used in this competition is shown in Fig. 3.4, and was formed by the following components:

- *Aerial platform and autopilot*: As mentioned, the Parrot AR.Drone 2.0. incorporates the low-level embedded controllers described in Section 11.1.1. In addition to the low-level embedded controllers, the attitude control and the vertical velocity control of the navigation controllers presented in Section 11.1.2 was provided by the Parrot autopilot. Moreover, as stated, the autopilot estimates the aerial robot attitude, its flying altitude, its horizontal linear velocity, and its angular velocities. Finally, the autopilot includes as well some actions like take-off, hover and land.

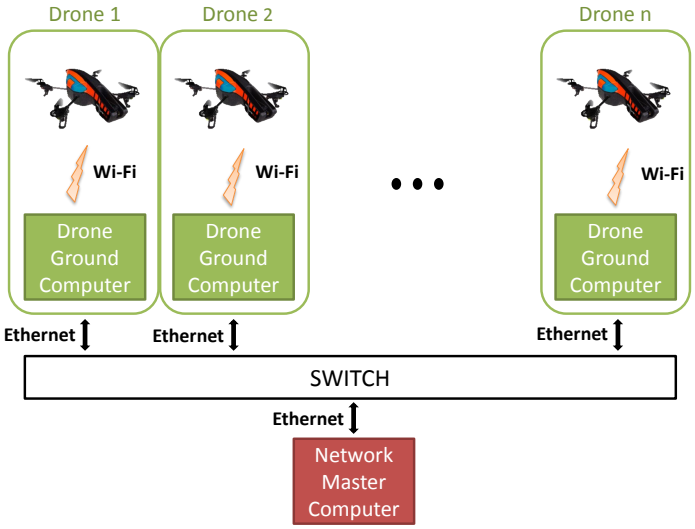


Figure 12.5: Proposed hardware setup for the IMAV 2013 competition.

- *Trajectory controller*: included several navigation controllers that accepted path references and transformed them into commands accepted by the autopilot controllers of the platform, ensuring that the reference is followed by the aerial robot (see Section 11.1.2). Additionally, it included some features of the Executive System of the current system architecture like the sequencing of the commands to enable and disable the different kinds of controllers.
- *Perception components*: The perception components developed for this competition are described in Chapter 7, and comprises the following:
  - *Aruco eye*: it was in charged of the detection of the ArUco visual markers and their 3D pose reconstruction with respect to the camera frame (it is fully described in Section 7.1).
  - *Pose estimator*: it estimated the state (pose and velocities) of the aerial robot with respect to the take-off point. For the prediction stage, a model of the aerial robot was used, while the update stage fused the information given by the autopilot estimates in terms of attitude, ground velocity, and flying altitude (see Section 7.2).
  - *Localization*: to compensate the drift on the pose estimation of the pose estimator component, a visual marker based SLAM was incorporated (see Section 7.3). This drift-free estimated pose of the aerial robot was shared with the rest of the agents of the fleet.
  - *Obstacle detector*: it had the responsibility of the perception of the obstacles in the environment (poles and wall). To do so, the mapped visual markers were used, together with a previous knowledge of the association of the visual markers and the obstacles (see Section 7.4).
- *Trajectory planner*: it generated collision free paths knowing the obstacles in the environment and the pose of the other robotics agents of the fleet (see Chapter 10). Additionally, it was in charged of monitoring that the planned paths were still collision-free, generating a new collision-free path otherwise. Moreover, if the trajectory planner was unable to calculate a collision-free path, an empty path was commanded, meaning to the controller to stay hovering, until a collision-free path was found. Note that this responsibility has been delegated to the mission planner of Planning System in the proposed system architecture.
- *Mission planner*: it was in charged of sequencing the complete mission and commanding the actions that were required to be executed (see Section 11.2).
- *Brain*: this component had several responsibilities, nowadays distributed between some components of the system architecture. For example, it was responsible for some of the tasks of the Executive System, as well as some Supervision System responsibilities. Additionally, an incipient action specialist of the Planning System was included in this component. Also, it was acting as a basic Communication System.

An emergent behavior appeared in the system, thanks to the trajectory planner component. This component allowed the collision-free navigation of all the agents of the fleet without having designed any particular collision avoidance behavior. Additionally, as all the aerial robots were executing the same mission, but with a certain time lag (around 15 seconds), the trajectory planner contributed to creating another emergent behavior enabling the synchronization of all the agents of the fleet when the number of them was high.

Achieved results

Competition results

The competition results can be checked online in `http://www.imav2013.org/index.php/final-results.html`.

The research group obtained 195.1 points in the Indoor Autonomy challenge, ranking the 1st place, and therefore obtaining the award shown in Fig. 12.6.

Figure 12.6: Award obtained in the IMAV 2013 Competition. 1st place in the category Indoor Autonomy.

Simulation results

The following paragraphs show the results of a series of simulations that were run to benchmark the designed system in a simulated IMAV 2013 competition environment.

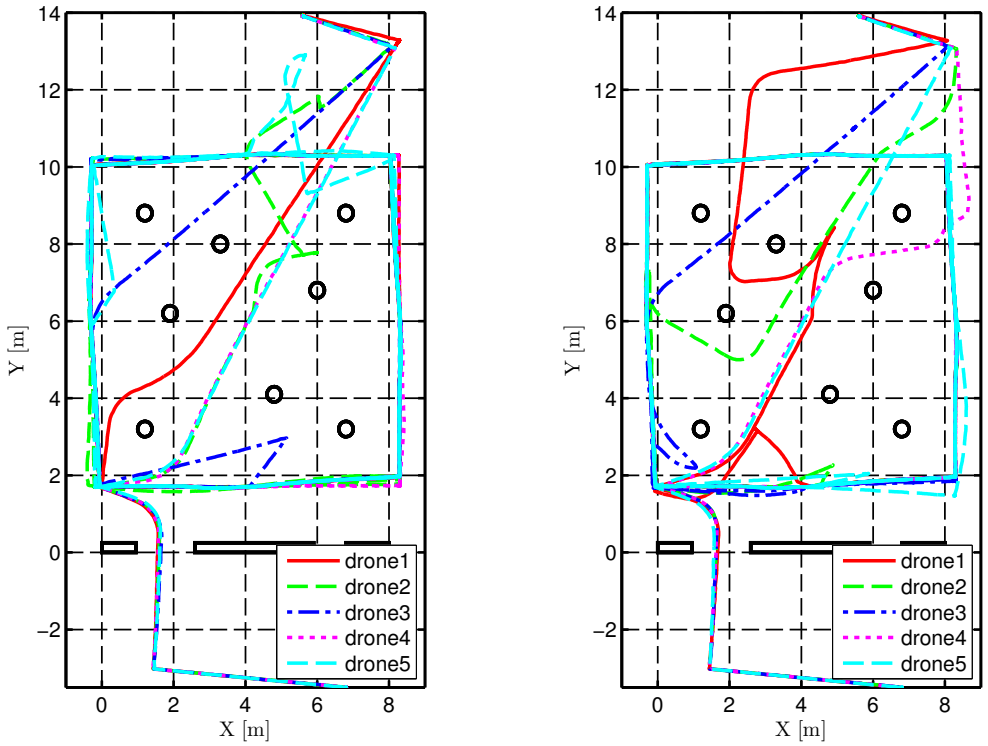
Two examples of these simulations, where five aerial robots fly in a simulated replica of the IMAV 2013 environment are shown in Fig. 12.7a and Fig. 12.7b. In both simulations, all the agents (ordered by their launching time) were able to accomplish the mission successfully. The launch of the aerial robots was timed and performed at about every 15 seconds. The poles are plotted as black circles, and the wall is plotted by three black rectangles, with the two free collision passages represented as windows. The size of the simulated aerial platforms (AR.Drone 2.0) is too big for the small window, so all the aerial robots have to cross through the big window. The path followed by each agent is shown as a distinct line, whose color and line style are specified in the figures' legend. In Fig. 12.7a, the conflicts during the navigation when an aerial robot had to avoid another one are easily perceived because they stand out from the normal execution of the mission. Only two such conflicts occurred during this simulation: the first one resulted in *drone3* taking a detour during the laps execution when another aerial robot was traversing the window; and the second one occurred while *drone2* was finishing the crossing of the unknown poles area. In Fig. 12.7b, a higher number of navigation conflicts occurred during the mission execution.

Experimental results

In the following paragraphs, one experimental test with three aerial robots is shown. The aerial robots performed the IMAV 2013 competition mission in a replica of the IMAV 2013 competition environment.

A 2D overview plot of the flight is represented in Fig. 12.8, with the same meaning than Fig. 12.7a and Fig. 12.7b. The inner unknown pole locations shown are the estimates of *drone1*. The aerial robots are commanded to look at specific known poles to control the position error by perceiving known visual markers. At the middle of each side, the agents are commanded to look at the next corner pole. At this moment the position estimate might change. These events increase the localization error in specific points of the mission execution and trigger the planning of new paths. In this flight, they occur repeatedly in the points $[-0.5,\ 7.75]$ and $[5.5,\ 10.5]$.

Fig. 12.9 shows the same experimental flight than Fig. 12.8, but now displaying the current status of the fleet at different time instants. These figures, ordered in time, from top left to bottom right, show each agent's plot



(a) First run.

(b) Second run.

Figure 12.7: Simulated flights where five aerial robots flew simultaneously to perform navigation tasks in a replica of the IMAV 2013 environment.

with different color and line styles. The aerial robots ordered by their launch times are plotted with: (*drone1*) green solid lines, (*drone2*) blue-dash dotted lines and (*drone3*) red dashed lines. The planned path is shown with a thicker width than the actual executed path. The dotted lines are the current path references at the end of the plotted time period. The arrows indicate the direction at which the aerial robot is estimated to be looking in the plotted position. As shown, the aerial robots are commanded to look at one of the four known corner columns of the map. The unknown poles mapped by *drone1* are shown in these figures.

Related publications

The complete details about the system setup of this competition together with a large number of results can be consulted in (Sanchez-Lopez et al., 2013a), (Pestana et al., 2014e), and (Pestana et al., 2016).

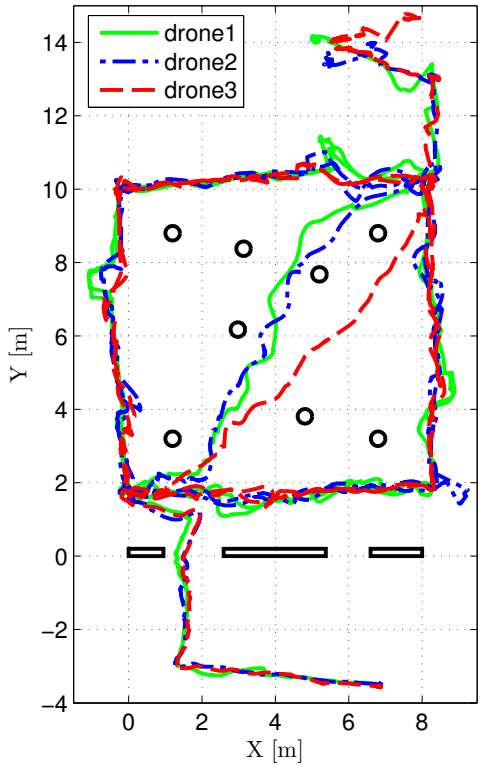


Figure 12.8: Experimental flight, three aerial robots fly simultaneously performing navigation tasks. The environment is a replica of the IMAV 2013 Indoor Challenge environment.

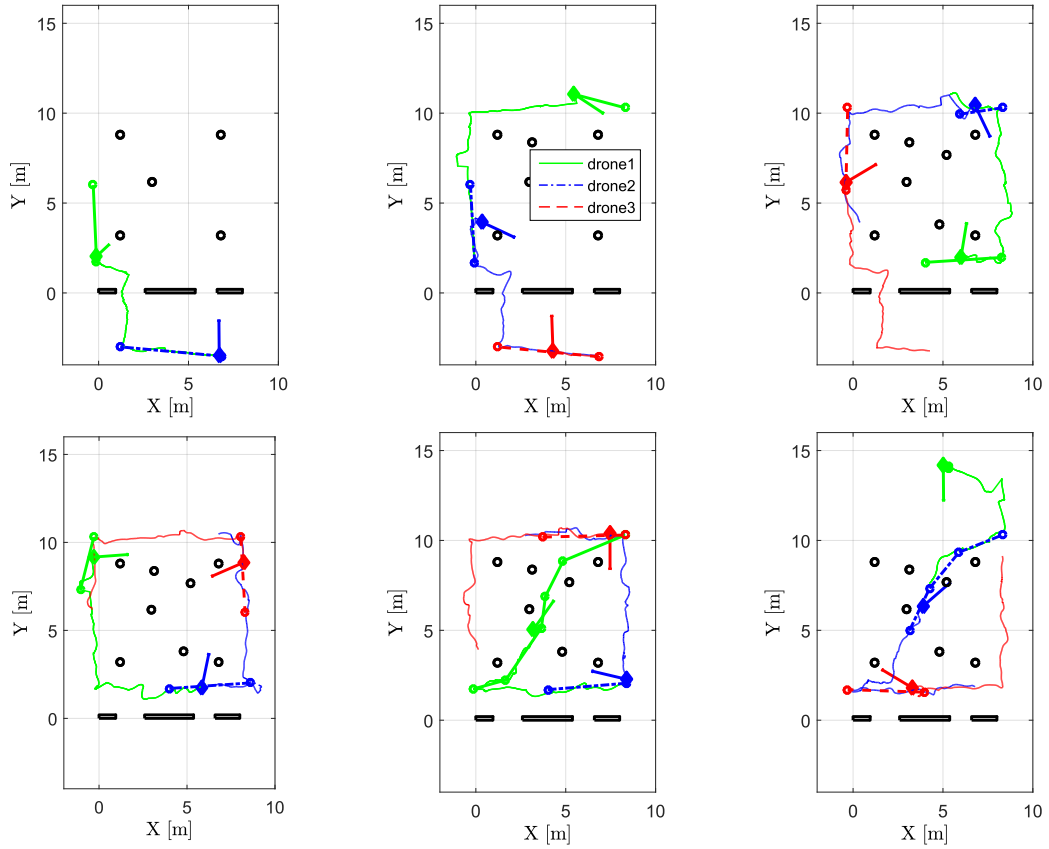


Figure 12.9: Different instant times of the same experimental flight as shown in Fig. 12.8.

12.2.3. IARC 2014 (Mission 7)

Competition description

The 7th mission of the International Aerial Robotics Competition (IARC) (see `http://www.aerialroboticscompetition.org/`) proposes an 'impossible' (sic) challenge. The mission is divided in two, mission 7a and mission 7b. The later will only be carried out if at least two teams are able to complete mission 7a.

Mission 7a challenge requires the fully autonomous navigation during 10 minutes of an aerial robot in a GPS-denied, physical clues (e.g. walls) free, large flat indoors area as depicted in Fig. 12.10. The arena contains some visual clues: a white wide line marks two sides of the arena, while a red and a green wide line mark the other sides. Additionally, a 1×1 m^2 white grid gives the only guaranteed visual features inside the arena.

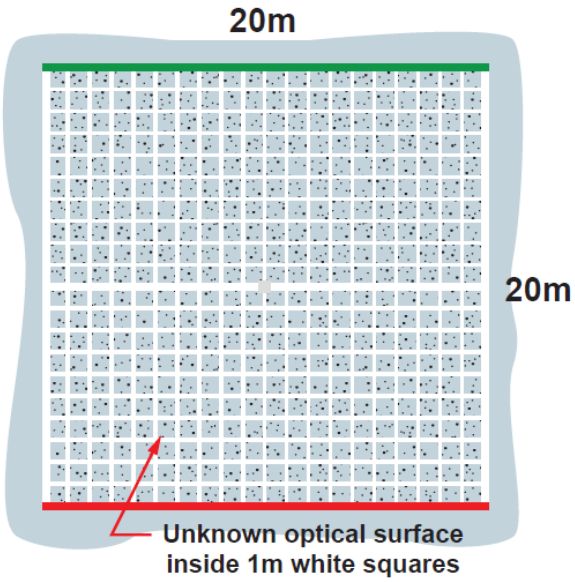


Figure 12.10: Arena of the 7th mission of the International Aerial Robotics Competition. Figure extracted from the rules.

Ten ground robots (iRobot create), also called target robots, move inside the arena with a predefined movement that includes random turns. Whenever a ground robot collides with something, it does a 180° turn. Additionally, the ground robots have a sensor on the top that, when excited, forces the robot to turn 90°. The robots are white colored with a red or green colored plate on the top.

In addition to these ground robots, another extra four ground robots, moving in a circle, and with a white colored plate on the top, are carrying random height poles. These robots are considered obstacles.

The main objective of the competition is to autonomously guide with the aerial robot more than seven ground robots to one side of the arena (red or green) by interacting with them, without colliding with the obstacle robots.

This mission has been repeated (in August, simultaneously in the two venues, China and USA) yearly since its first run in 2014 until a team is able to complete the mission and win the challenge. At the moment of writing these lines, no team has managed to even guide a single robot to one of the sides of the arena.

The research group participated in this competition in August 2014, in the China venue.

System setup

Being aware of the 'impossibility' of the challenge, the research group decided to plan the participation in the competition as a two years project with two milestones in the two competition dates. The first year project (2014) aimed to develop the basic components for the autonomous collision-free navigation in the arena and the detection and tracking of the ground robots. The second year was going to focus on the ground robot interaction and their guidance. The experience acquired during the participation in the first year competition, despite knowing that winning the challenge was impossible, was important to test the developed components on the real arena and to be able to improve them for the second year competition. Unfortunately, for several reasons, the second year

could not be carried out the year after the first participation, although currently, the research group has plans of participating in the 2017 or 2018 editions.

The proposed system was a purely vision-based aerial robot for the arena localization and the ground robots detection. After the take-off, a local localization was providing the pose of the robot with respect to the take-off point. A first exploratory task was required to be executed after the take-off to be able to localize the robot with respect to the arena (e.g. knowing where the green and red lines are). Once the aerial robot was localized in the arena, the mission to guide the ground robot started. As the detection range of the ground robots was very limited, only a reduced number of robots were able to be tracked and therefore, the confidence on the pose estimation of the other ground robots quickly decreased. Depending on the pose estimation of the ground robots, an interaction or an exploration (to find ground robots) task might be executed.

Hardware setup

A modified version of the AscTec Pelican quadrotor was used (see Fig. 12.11).

This quadrotor was equipped with four ESC and with the AscTec autopilot, an autopilot board that stabilizes the vehicle using the information from IMU, pressure altimeter and magnetometer fused by means of an internal Kalman filter, providing as well an estimate of the aerial robot attitude, its angular velocities, and its linear accelerations. This stabilization controller and the state estimator are embedded, closed, and unmodifiable but gains are tunable. The autopilot allows to command the vehicle in attitude mode for the horizontal angles (pitch and roll), in velocity mode for the vertical angle (yaw), and the average thrust command given by its motors.

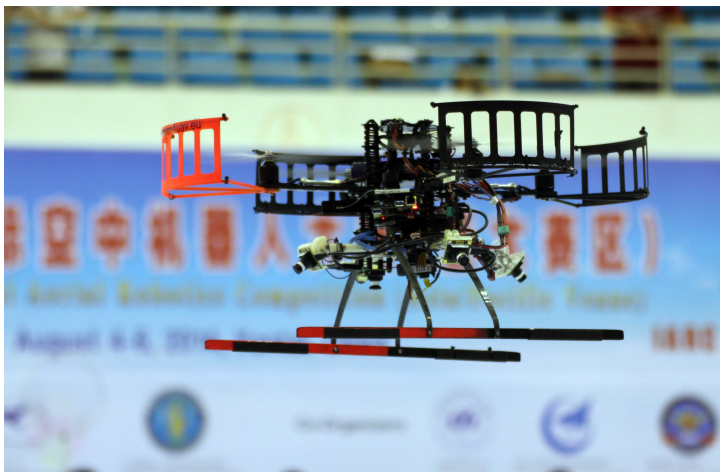


Figure 12.11: Modified version of the AscTec Pelican for the participation on the 7th mission of the IARC. Picture from the competition.

Four cameras were mounted on its perimeter (front, back, right and left) and another extra camera was mounted on its bottom part, looking downwards. All these cameras were used for the detection of the arena visual features and the ground robots. Additionally, a px4flow sensor was mounted to measure its horizontal velocity and the distance to the ground.

The quadrotor was equipped with an onboard computer, an AscTec Mastermind, which has an Intel i7 processor with 4 GB of RAM. The Mastermind was connected to the AscTec autopilot by means of a serial cable, and it was connected by means of a Wi-Fi network to a ground computer for operator monitoring and supervision. All the algorithms were run on the Mastermind computer.

System architecture setup

The system architecture used in this competition is shown in Fig. 3.6, and was formed by the following components:

- *Aerial platform and autopilot*: As mentioned, the AscTec Pelican incorporates the low-level embedded controllers described in Section 11.1.1. Additionally to the low-level embedded controllers, the attitude control of the navigation controllers presented in Section 11.1.2 was provided by the AscTec autopilot. Moreover, as stated, the autopilot estimates the aerial robot attitude, its angular velocities, and its linear accelerations.
- *Localization and mapping*: this system includes all the components with the functionalities of the Feature Extraction and Situation Awareness System. It is divided in:

  - Computer vision algorithms: It groups the algorithms that would be included in the Feature Extraction System (as for example, the one described in Section 8.1).
  - Odometry-based state estimator: It represents the component to estimate the state of the aerial robot based on the measurements of the odometric sensors (as the component described in Section 8.2).
  - Localization and mapping: It gathers all the components of the Situation Awareness System used to estimate the drift-free estimate of the state of the aerial robot in the arena (as the components described in Section 8.3, Section 8.4, and Section 8.5), and the components required to estimate the state of the ground robots (both obstacles and targets).
- *Flight controller* and *Mid-level controller*: These two systems were handling symbolic actions of different complexity.
  - The *mid-level controller* accepted four symbolic actions (take-off, land, hover and move), counting with a small Executive System to sequence them and a basic Motor System.
  - The *flight controller* accepted more complex symbolic actions (navigate in path following, position or velocity; perform fast altitude movements; or interact with the ground robots by means of touching them or landing in front of them). This component had internally a basic Executive System and a complete Motor System. Additionally, as it included some deliberative actions (like navigate in path following avoiding obstacles), part of the Planning System was included here, concretely a trajectory planner (like the one described in Chapter 10).

  These two systems included some of the controllers presented in Section 11.1.
- *System supervisor*: this complex component included features of the modern Executive System, Supervision System, and the action specialist of the Planning System.
- *Mission planner*: it was responsible for sequencing the mission commanding the required actions. Its functionalities were equivalent to the modern mission planner of the Planning System.
- *Human machine interface*: it was responsible to some of the functionalities of the modern Communication System.

The reader must note that the ground robot interaction components were considered on the proposed architecture as a proof of concept, but these components were never developed as they were scheduled for the second year project of the competition.

Achieved results

Competition results

As stated before, at the moment of writing these lines, no team has managed to complete the mission or even guide a single robot to one of the sides of the arena.

Due to several problems faced before the competition day, such as the insufficient payload of the aerial platform, the limited bandwidth of the onboard computer when acquiring the images from the five cameras, or the limited performance of some of the computer vision algorithms, the complete system architecture designed for this competition could not be tested in the competition and only partial tests were carried out.

Despite the difficulties, the performance of two components was awarded after the competition:

- Best System Control Award (Fig. 12.12a)
- Best Target Detection Award (Fig. 12.12b)

Related publications

The complete details about the system setup of this competition together with a large number of results can be consulted in (Pestana et al., 2014d), and (Sanchez-Lopez et al., 2015).

(a) Best System Control Award. (b) Best Target Detection Award.

Figure 12.12: The two awards obtained in the IARC 2014 Competition.

12.2.4. IMAV 2016

Competition description

The 2016 International Micro Aerial Vehicle Competition (IMAV 2016), that took place in Beijing (China), had several challenges. The research group participated in the "Indoor Competition".

In this challenge, up to two aerial vehicles had to navigate in the environment represented in Fig. 12.13 during 30 minutes.

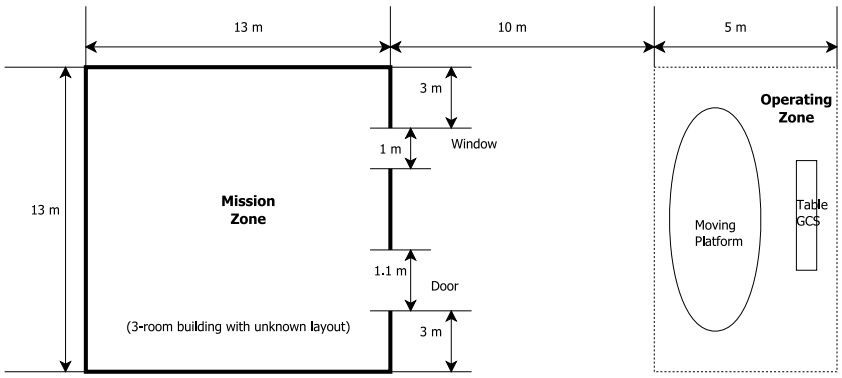


Figure 12.13: IMAV 2016 - Indoor competition arena.

The mission consisted of multiple mission elements that could be performed in any order by a single aerial vehicle. Not all the mission elements were targeted to be executed in the research group strategy. The ones affected were the following:

- Take-off at Operating Zone from a moving platform
- Enter building through the window or the doorway
- Search Bucket A and drop in Item A.
- Search Bucket B and drop in Item B.
- Exit building through the window or the doorway
- Landing at Operating Zone on a moving platform
- Provide a 3-Dimensional map of the interior of the building

The total score achieved was given by the following equation:

$$T = \sum_{\forall j} (I_j \cdot M_j \cdot A_j \cdot V_j) \tag{12.3}$$

being:

- I_j, the in-a-row factor for the j-th mission element.
- M_j, score of the j-th mission element (see IMAV 2016 rules for more details).
- A_j, the level of autonomy during the execution of the j-th mission element. Being $A_j = 1$ if first person view, $A_j = 6$ if semi-autonomous, and $A_j = 12$ if fully-autonomous. If any external aid was used, $A_j = A_j - 2$.
- V_j, the size factor of the MAV performing the j-th mission element, given by $V_j = (2 - L/100)^2$, where L is the maximum horizontal distance of a rotary wing (including blades) in *cm*.

System setup

The competition strategy was to use a single fully autonomous aerial robot equipped with a large number of sensors to carry out all the mission elements in-a-row.

Hardware setup

A custom quadrotor was built for the participation of this competition (see Fig. 12.14). This quadrotor was using a Pixhawk as the autopilot for the attitude and thrust control. This autopilot counts as well with a state estimator that provides an estimate of the attitude, angular velocity and flying altitude of the aerial vehicle, using its onboard sensors (IMU, magnetometer, and barometer).

Figure 12.14: Custom aerial platform for the participation on the IMAV 2016 competition. Picture from the competition.

The aerial vehicle was equipped as well with a Hokuyo 2D LIDAR to detect the walls of the building; an Intel RealSense RGB-D camera, used for the creation of a 3D map of the building and for the detection of the buckets; a bottom camera for the detection and tracking of the buckets before the item release operation; and a down-looking LightWare 1D LIDAR for altitude estimation and for detecting the moving landing platform.

System architecture setup

The proposed system architecture used in this competition is shown in Fig. 3.9, being its main components the following:

- *Feature Extraction System*: a bucket color-based detection was developed.

- *Motor System*: the existing controllers available on Aerostack, and described in Section 4.2, were used.
- *Situation Awareness System*: several new components were included, such as:
  - Flight altitude aerial robot state estimation.
  - LIDAR-based aerial robot localization and mapping using grid maps.
  - Multi-sensor fusion for aerial robot state estimation.
  - State estimation and mapping of the items.
  - State estimation of the moving landing platform.
- *Executive System*: the existing Executive System components available on Aerostack were used, incorporating the drop item action, and the visual servoing using the bottom camera action.
- *Planning System*: the existing Planning System components available on Aerostack were used, as the task-based dynamic mission planner presented in Section 11.2. A new path planner based on grid maps was developed.
- *Supervision System*: the Supervision System components available on Aerostack were used.
- *Communication System*: the Communication System components available on Aerostack were used, as the GUI presented in Section 11.4.

As the reader might note, most of the components used were available on Aerostack before the competition, having developed specifically for this competition only a few of them, mainly those related to the perception.

Achieved results

Competition results

The competition results can be checked online in `http://www.imavs.org/2016/documents/Result.pdf`.

The research group obtained 268.1 points in the Indoor challenge, ranking the 4th place. All the targeted mission elements described above were successfully carried out in a row with the highest level of autonomy. The reduced number of points collected in comparison with other teams was due to the fact that the aerial vehicle used by the research group was noticeably bigger than the rest of the competitors what reduced the V_j factor (being $V_j \approx 1$). The difference in size was due to two main reasons: on the one hand, the used aerial robot was the only one with protections and the only team without having any crashes during the whole competition week; on the other hand, a single aerial vehicle was used for the complete mission execution and therefore a variate number of sensors needed to be carried out.

As the participation of the author of the thesis in this competition was minor and mainly related to giving advice, higher details on the achieved results related to this competition are out of the scope of this thesis, having presented it to support the versatility and usability of the proposed system architecture and software framework.

Related publications

The complete details about the system setup of this competition together with a large number of results can be consulted in (Sampedro et al., 2017).

12.3. Self-proposed challenges

This section describes four kinds of self-proposed challenges that have been carried out:

- Multi-robot navigation, exploration and inspection, Section 12.3.1.
- Visual object following, Section 12.3.2.
- Multi-modal user interaction, Section 12.3.3.
- Search and rescue, Section 12.3.4.

The structure of all these sections is always the same: firstly, the challenge (including the environment) is briefly described; then, the proposed system setup (hardware and system architecture) is explained; after that, the experimental results are shown; later, in few lines, the open future works are stated; and finally the related publications are listed.

12.3.1. Multi-robot navigation, exploration and inspection

Challenge description

This set of challenges focuses on the fully autonomy of a fleet of aerial robots, exploring their different levels of interaction. These challenges began as a consequence of the successful participation in the IMAV 2013 competition (see Section 12.2.2), defining in this set of challenges more demanding objectives.

Three different kinds of missions are tested: a pure navigation mission, an exploration mission, and a search and inspection mission. The difference between them is the increasing level of interaction between the robotic agents of the fleet.

Challenge: Navigation

The first challenge focuses on collision-free fully autonomous navigation.

In the IMAV 2013 competition, all the robots had the same mission, performing all the competition mission elements in a predefined order, sharing the environment. As stated in (Pestana et al., 2016), the observed emergent behavior is a synchronization in the execution of the tasks, where a robot waited while another robot was performing a particular task.

In this challenge, every robotic agent has a different mission, although they might be similar (e.g. go from point A to point B, being points A and B different for every robotic agent). All the robotic agents have to share the same environment, and to make the challenge more demanding, the aerial robots are forced to share limited parts of the environment (e.g. all the robots are forced to fly through a small window as a task of the complete mission). The only interaction between the aerial robots is, therefore, sharing the environment, and the solution to this problem comes in an emergent way, without any master or coordinator.

Challenge: Exploration

The second challenge tackles the fully autonomous exploration of an environment (with previously a known size and shape) by a fleet of aerial robots.

The main difference between the navigation and the exploration is that in the exploration, all the robots of the fleet share the same global mission (the exploration of the environment), and therefore a cooperation between the agents is required to efficiently fulfill the objective of the mission.

For this challenge, the centralized approach presented in Section 3.3 is used, although, as stated in that section, a decentralized approach might be used as well.

Challenge: Search and inspection

The fully autonomous search and inspection of an element of the environment correspond to the third challenge.

The main difference between the exploration and the search and inspection missions is the level of interactions between the robotic agents. Whereas in the exploration mission all the agents share the global goal and the environment, behaving all them similarly, in the search and inspection mission, some robotic agents might have different particular goals within the same global mission. Additionally, these particular goals depend on the environment and the state of the system, and they are dynamically assigned.

An example to illustrate this is the following: a fleet of aerial robots have the goal of finding a particular object hidden in the environment and then to inspect it. At the beginning, all the robots have the same goal: to find the object to inspect. Once the object is found by a particular robot, the goal of every robot is changed and depends on the environment and the current position of the robotic agents. For example, the robot that found the object to inspect, and the one that is closer to it might have the goal to inspect it, while the others are required to return to their takeoff positions.

Similarly to the exploration mission, the centralized approach presented in Section 3.3 is used.

Environment

Similarly as the IMAV 2013 competition, the environment is simplified to an indoor structured static one formed by simple elements (poles and walls). The environment is augmented with visual markers, to simplify the aerial robot localization, and for obstacles perception and mapping. The elements of the environment are therefore labeled with a set of unique previously known visual markers. The environment (number and position of its elements) might be previously unknown, being the responsibility of the aerial robots, its mapping. The floor is assumed to be flat and to have enough visual features (i.e. a texture).

Different environment configurations (i.e. number, kind and pose of the elements of the environment) are used in the different challenges.

System setup

The employed approach for solving this set of challenges is similar to the one used in the IMAV 2013 competition (see Section 12.2.2) where a fully autonomous fleet of relatively simple aerial robots was used. The selected aerial platform was an out of the box Parrot AR.Drone 2.0. As this platform has a limited payload, all the processing (apart from the out of the box onboard autopilot) is required to be done on a ground computer.

Moreover, due to the limited number of sensors available on the aerial robot, to self-localize it and to perceive the environment, external aids are used. These external aids are ArUco visual markers that are placed on the poles and on the wall.

Hardware setup

Likewise as for the IMAV 2013 competition (see Section 12.2.2), a fleet of out of the box Parrot AR.Drone 2.0. is used.

This aerial platform has two cameras (a front camera and a bottom camera), together with an IMU, a magnetometer, a barometer, and a down-looking ultrasound sensor. It includes an autopilot that estimates the aerial robot attitude (by means of the IMU and the magnetometer), its flight altitude (thanks to the ultrasound sensor and the barometer), its horizontal linear velocity (calculating the optical flow with the bottom camera), and its angular velocities and linear accelerations (filtering the IMU). The autopilot allows to command the aerial platform in attitude mode for the horizontal angles (pitch and roll), and in velocity mode for the vertical angle (yaw) and for the vertical movement (z). The autopilot includes as well some actions (and the required controllers to execute them) like take-off, hover and land.

As described in Section 12.2.2, every aerial platform is connected via Wi-Fi to a ground computer for all the processing tasks. All the ground computers are connected to a LAN by means of a switch.

A different number of aerial robots are used in the different experiments.

System architecture setup

The proposed system architecture described in Chapter 3 is used, being the components employed available on Aerostack, as the following:

- *Perception*: The perception solution presented in Chapter 7 was used.
- *Motor System*: The existing low-level embedded controllers and the navigation controllers available on Aerostack, and described in Section 11.1.1 and Section 11.1.2, were used.
- *Executive System*: the existing Executive System components available on Aerostack were used.
- *Planning System*: the existing Planning System components available on Aerostack were used, as the task-based sequential mission planner presented in Section 11.2, and the collision-free path planner described in Chapter 10.
- *Supervision System*: the Supervision System components available on Aerostack were used.
- *Communication System*: the Communication System components available on Aerostack were used, as the GUI presented in Section 11.4.

Experimental results

This section shows the experimental results on the three different kinds of missions tested: a pure navigation mission, an exploration mission, and a search and inspection mission.

Navigation

Bellow some simulation and experimental results related to the navigation challenge are shown (see Fig. 12.15).

Two different environment configurations are defined:

- "Pinball": Several poles are placed in the middle of the environment in a diamond configuration, emulating the pins of a pinball.
- "Hole": Several poles are placed in the middle of the environment forming a single opening (a hole) that all the aerial robots have to cross.

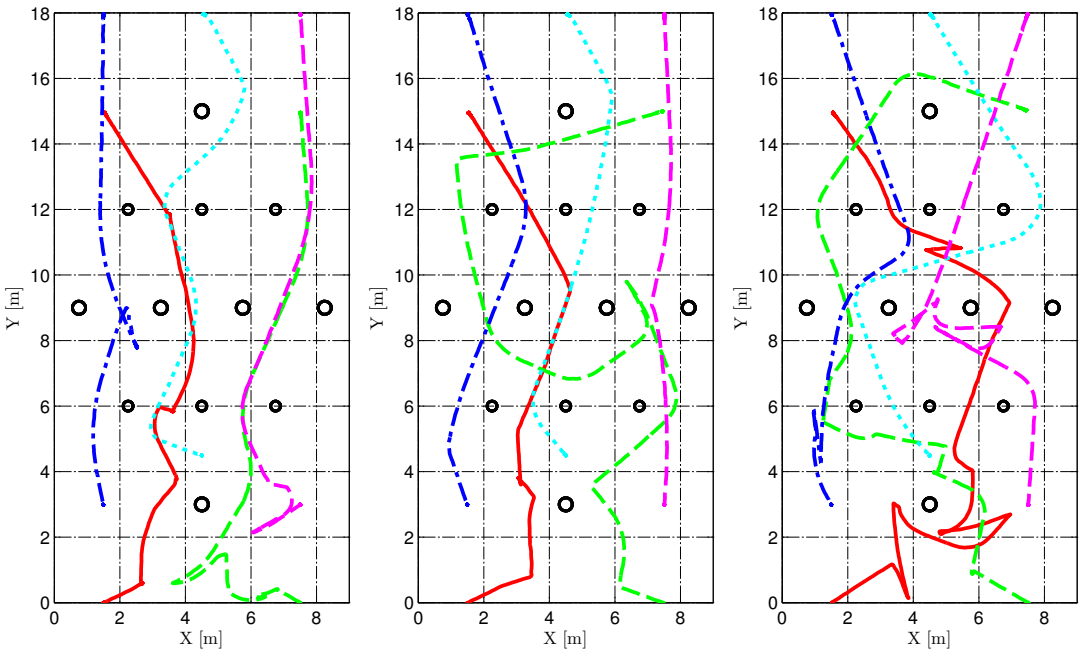
The two environments have been tested with five aerial robots flying simultaneously, first in simulation and then with real experiments. All the robots have a similar mission that roughly consists on: (1) take-off, (2) navigate from point A to point B, (3) land. Points A and B are, of course, different for every aerial robot.

Fig. 12.16 shows the trajectories performed by the aerial robots during three different simulations of the "Pinball" environment. The circles displayed in black represent the poles in the environment. As there is no coordinator or master, and the interaction between robots emerges, the trajectories followed are different in every execution of the simulation. These trajectories might be more direct, with less interferences between the aerial robots (as in Fig. 12.16a), or with a high number of interferences (as in Fig. 12.16c.

Figure 12.15: Image of a multi-robot navigation experiment with the “Hole” environment.

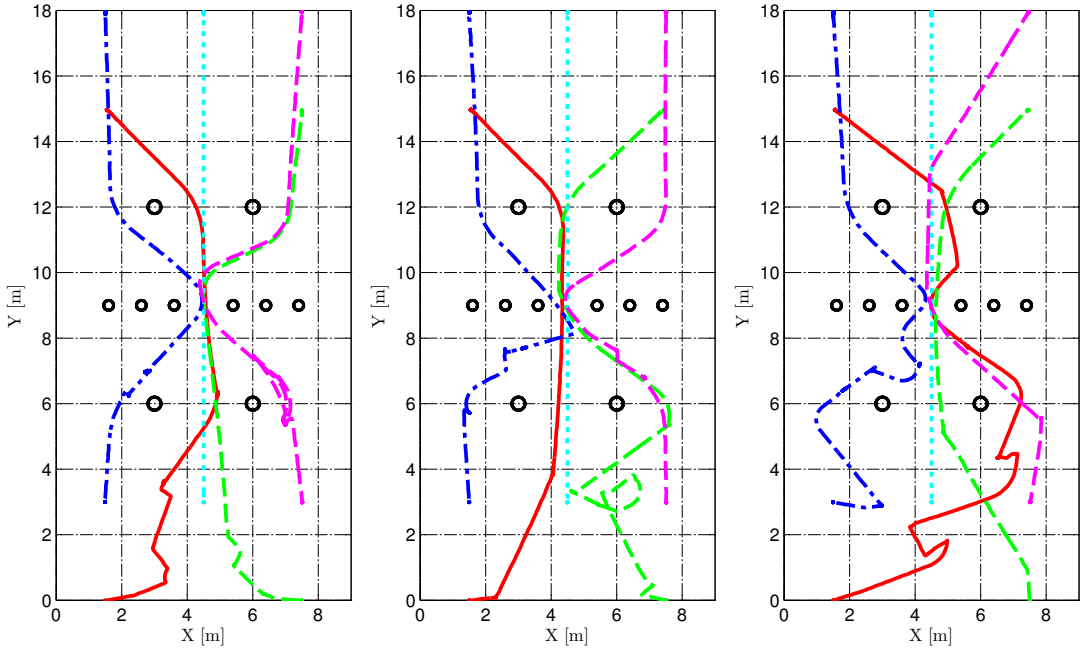
Fig. 12.17 shows the trajectories performed by the aerial robots during three different simulations of the “Hole” environment. The same considerations done for Fig. 12.16 apply to this figure.

Fig. 12.18 and Fig. 12.19 shows different executions of the “Pinball” and “Hole” environments, respectively, in real flights. The trajectories followed by the aerial robots are equivalent to the ones performed during the simulations, although they are noisier due to the localization errors and the noise in the real measurements.



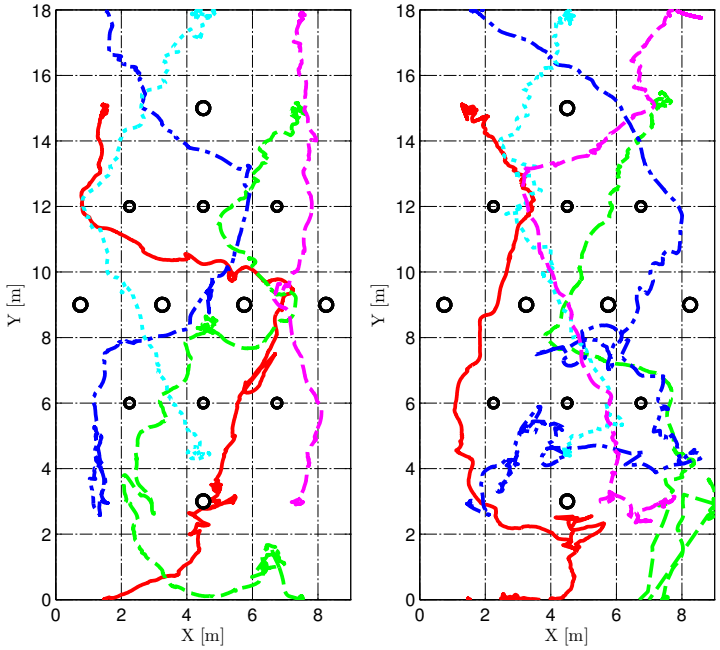
(a) Simulation 1. (b) Simulation 2. (c) Simulation 3.

Figure 12.16: Trajectories performed by the aerial robots during a simulation of the “Pinball” environment.



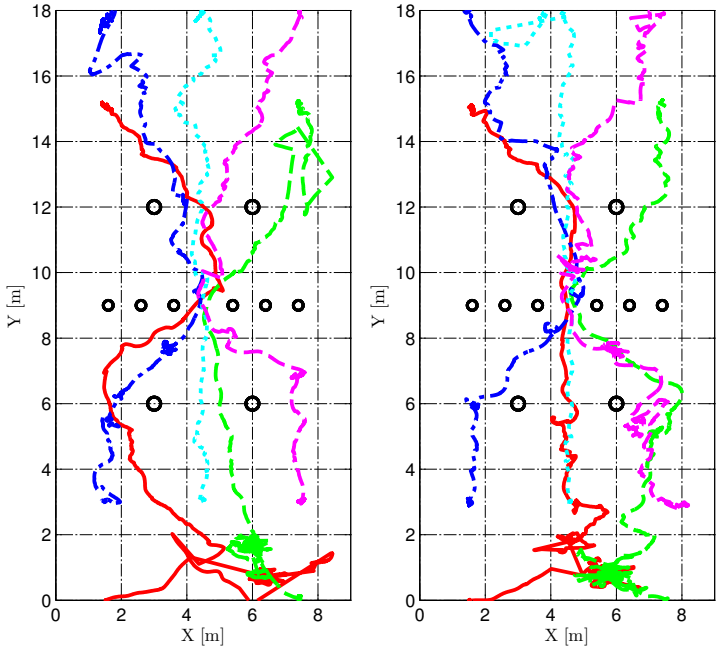
(a) Simulation 1. (b) Simulation 2. (c) Simulation 3.

Figure 12.17: Trajectories performed by the aerial robots during a simulation of the “Hole” environment.



(a) Real experiment 1. (b) Real experiment 2.

Figure 12.18: Trajectories performed by the aerial robots during a real experiment of the “Pinball” environment.



(a) Real experiment 1 (b) Real experiment 2

Figure 12.19: Trajectories performed by the aerial robots during a real experiment of the “Hole” environment.

Exploration

Bellow some simulation results related to the exploration challenge are shown

Fig. 12.20 shows the trajectories followed by the six aerial robots employed during a simulation experiment. The red squares represent the poles in the environment, while the green triangles represent the initial position of the aerial robots. In this simple exploration mission, every robotic agent has to: (1) take-off, (2) explore its corresponding area given by the master coordinator, (3) land Note that the aerial robots do not return to their takeoff positions after the exploration. The reader must appreciate in Fig. 12.20 that the complete environment is visited.

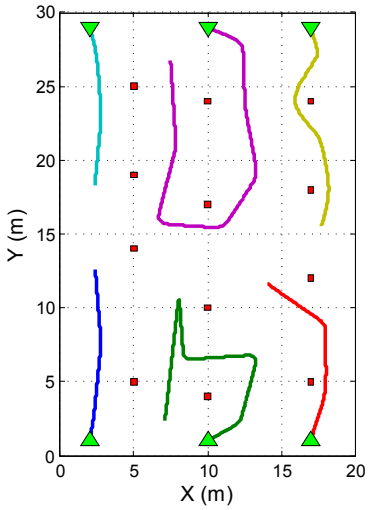
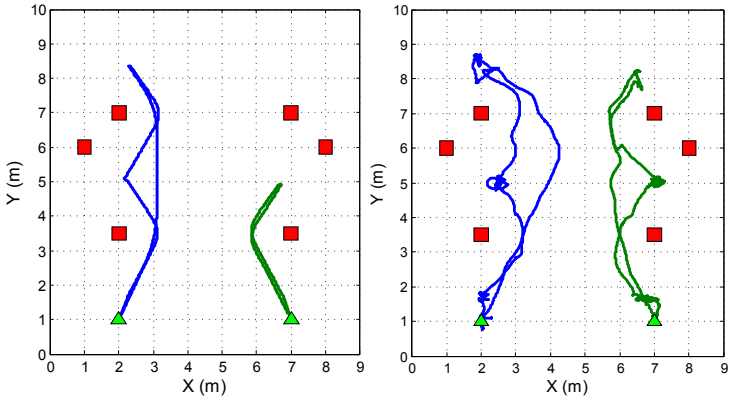


Figure 12.20: Trajectories performed by the aerial robots during a simulation experiment.

Search and inspection

Bellow some simulation and experimental results related to the search and inspection challenge are presented.

Fig. 12.21 shows the trajectories followed by the two aerial robots used during an execution of the search and inspection mission (in simulation and during a real experiment). The red squares represent the poles in the environment, while the green triangles represent the initial position of the aerial robots. The target to inspect is situated at the point $P\ [2, 7, 1.3]$ and is found by the robot with the blue trajectory. After the take-off and the assignation of the area that every agent has to explore, the exploration begins. Once the left robot finds the target, the inspection begins. The other aerial robot returns home while the left one executes the inspection tasks. After the inspection, the left aerial robot is commanded to come back home.



(a) Simulation. (b) Real experiment.

Figure 12.21: Trajectories performed by the aerial robots during an experiment.

Future work

This set of challenges focuses on the different levels of interaction between the robotics agents of the fleet, in a simplified but still realistic environment. There exist two clear possible lines of future work. In the one hand, the simplification assumed in the environment might be removed, working in a more realistic environment. In the other hand, other kinds of robotic agents interaction might be explored such as multi-robot formations.

Related publications

The complete details about the system setup of this set of experiments, together with a larger number of results can be consulted in different publications:

- Navigation: in (Sanchez-Lopez et al., 2014a), and (Sanchez-Lopez et al., 2016b).
- Exploration: in (Sampedro et al., 2016).
- Search and inspection: in (Sampedro et al., 2016).

12.3.2. Visual object following

Challenge description

This challenge tackles the problem of following an object with an aerial robot using a camera and therefore a computer vision algorithm for the detection and tracking of the object to follow.

This challenge aims to test and to push to its limits the autonomous performance of an aerial robot when using the visual servoing controller described in Section 11.1.3.

Environment

The environment is highly simplified to a GPS-denied obstacle-free one. Only one textured moving object (the object to follow) is taken into account, normally a person, since his/her movements are more challenging than the movements of an artificial object like a car. The floor is assumed to be flat and to have enough visual features (like a texture).

System setup

Hardware setup

To simplify the hardware, a single out of the box Parrot AR.Drone 2.0. is used. This aerial platform has two cameras (a front camera and a bottom camera), together with an IMU, a magnetometer, a barometer, and a down-looking ultrasound sensor. It includes an autopilot that estimates the aerial robot attitude (by means of the IMU and the magnetometer), its flight altitude (thanks to the ultrasound sensor and the barometer), its horizontal linear velocity (calculating the optical flow with the bottom camera), and its angular velocities and linear accelerations (filtering the IMU). The autopilot allows to command the aerial platform in attitude mode for the horizontal angles (pitch and roll), and in velocity mode for the vertical angle (yaw) and for the vertical movement (z). The autopilot includes as well some actions (and the required controllers to execute them) like take-off, hover and land.

This platform is connected to a ground computer via Wi-Fi where all the processing tasks are carried out.

This challenge has been tested as well with another aerial platform such as an AscTec Pelican and a Mikrokopter Oktocopter equipped with a front camera. Nevertheless, since the moving object is normally a person, the Parrot AR.Drone is the preferred platform because it is the safest one.

System architecture setup

The proposed system architecture described in Chapter 3 is used, being the components employed available on Aerostack, as the following:

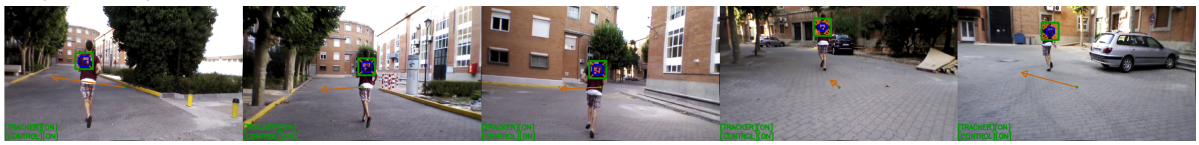
- *Perception*: An odometry based robot state estimation was used Section 7.2 for aerial robot state estimation. Additionally, a TLD based tracker (Kalal et al., 2012b) was used for the detection and estimation of the target's pose.
- *Motor System*: The existing low-level embedded controllers, the navigation controllers, and the visual servoing controller available on Aerostack, and described in Section 11.1, were used.
- *Executive System*: the existing Executive System components available on Aerostack were used.
- *Planning System*: the existing Planning System components available on Aerostack were used, as the task-based sequential mission planner presented in Section 11.2. No collision-free path planner was used as the environment was considered to be obstacle-free.

- *Supervision System*: the Supervision System components available on Aerostack were used.
- *Communication System*: the Communication System components available on Aerostack were used, as the GUI presented in Section 11.4.

Experimental results

Fig. 12.22 shows an experiment where the aerial robot is following a person that is running. The blue square on the onboard images represents the feedback given by the computer vision algorithm, while the green square represents the commanded motion reference to the visual servoing controller.

(a) Two images of the experiment. The image in the left shows a picture of the aerial robot and the person followed. The image in the right shows an onboard frame.



(b) Sequence of onboard images during the experiment.

Figure 12.22: Visual person following experiment.

Future work

Several research lines can be considered as future work. First of all, better computer vision algorithms can be developed to improve the object following in more demanding situations (e.g. when the object is partially or completely occluded). Secondly, the performance of the controller might be improved with a more precise execution, or by adding extra degrees of freedom (e.g. a gimbal). Finally, as the environment was simplified to focus on the visual object following task, the robot is unable to perceive obstacles and avoid them. Working in a more realistic environment would add more value to this challenge.

Related publications

This challenge is deeply described in (Pestana et al., 2013b), and (Pestana et al., 2014c), presenting multiple results and image sequences.

12.3.3. Multi-modal natural user interaction

Challenge description

This challenge focuses on the multi-modal natural interaction of the human user with the aerial robots, pushing the state of the art in this particular field. These experiments serve as the testbed of the components presented in Section 11.4.2.

The four kinds of natural user interfaces described in Section 11.4.2 were evaluated by means of a series of real-time experiments:

- Visual marker interaction.
- Visual body interaction.
- Hand gesture interaction.

- Speech command interaction.

These flight tests consisted of individually testing the every interaction method, thus evaluating whether or not these interfaces fit the needs and expectations of seamless natural interaction between the user and the aerial robot.

Apart from individually testing each interface method, a multi-modal interaction scenario is also tested by combining the interfaces and permitting the user to decide which interaction to use at any given time.

The reader must note that the previously described visual object following challenge (presented in Section 12.3.2) is a kind of natural user interaction when the object to follow is a person, and therefore it might be rather included in this section instead of as a separate challenge. Nevertheless, the visual object following is object independent, and its main objective is not the person interaction by itself but the fully autonomous performance of the aerial robot when following an object. Therefore, it has been considered as a separate challenge.

Environment

Similarly to the previously described visual object following challenge (presented in Section 12.3.2), the environment is simplified to a GPS-denied obstacle-free one with a flat textured floor.

System setup

Hardware setup

For the same reasons than the ones in the visual object following challenge (presented in Section 12.3.2), the preferred aerial platform is the small harmless low-cost platform Parrot AR.Drone 2.0. (described in Section 2.2).

This aerial platform has two cameras (a front camera and a bottom camera), together with an IMU, a magnetometer, a barometer, and a down-looking ultrasound sensor. It includes an autopilot that estimates the aerial robot attitude (by means of the IMU and the magnetometer), its flight altitude (thanks to the ultrasound sensor and the barometer), its horizontal linear velocity (calculating the optical flow with the bottom camera), and its angular velocities and linear accelerations (filtering the IMU). The autopilot allows to command the aerial platform in attitude mode for the horizontal angles (pitch and roll), and in velocity mode for the vertical angle (yaw) and for the vertical movement (z). The autopilot includes as well some actions (and the required controllers to execute them) like take-off, hover and land.

This platform is connected to a ground computer via Wi-Fi where all the processing tasks are carried out.

System architecture setup

The proposed system architecture described in Chapter 3 is used, being the components employed available on Aerostack, as the following:

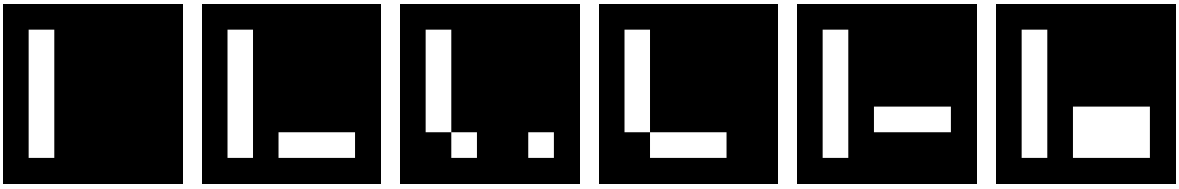
- *Perception*: An odometry based robot state estimation was used Section 7.2 for aerial robot state estimation. Additionally, a TLD based tracker (Kalal et al., 2012b) was used for the detection and estimation of the target's pose. Also, an ArUco visual marker detector, as the presented in Section 7.1 was employed.
- *Motor System*: The existing low-level embedded controllers, the navigation controllers, and the visual servoing controller available on Aerostack, and described in Section 11.1, were used.
- *Executive System*: the existing Executive System components available on Aerostack were used.
- *Planning System*: the existing Planning System components available on Aerostack were used, as the task-based sequential mission planner presented in Section 11.2. No collision-free path planner was used as the environment was considered to be obstacle-free.
- *Supervision System*: the Supervision System components available on Aerostack were used.
- *Communication System*: the Communication System components available on Aerostack were used, as the multi-modal user interfaces presented in Section 11.4.

Experimental results

This section shows the experimental results on the four different kinds of natural user interaction tested, in addition to the complete multi-modal natural user interaction test.

Visual marker interaction

This interaction was performed by using hand-held visual markers developed with the uniquely identified ArUco visual markers. Fig. 12.23 shows some examples of the markers used in flight for actions such as *take-off* (Fig. 12.23b) or *flip* (Fig. 12.23c).



(a) *hover* (b) *take-off* (c) *flip* (d) *follow me* (e) *land* (f) *follow trajectory*

Figure 12.23: Examples of visual markers used in the experiment.

Fig. 12.24 shows users interacting with the aerial robot using the visual marker NUI. For the test, users were given markers and were instructed they could use these in any way to interact with the aerial system. Since these markers included the flip action, this was the users' preferred command to send to the aerial robot. This test resulted in the interaction being described as fun and reliable. This may be due to the fact that the marker system employed is incredibly robust and users had an instant response when commanding the aerial robot since the visual markers send high-level tasks to perform.

Figure 12.24: Here the visual marker interaction is taking place. As can be seen in the bottom left image, the visual algorithm detects the ArUcos ids, and the aerial robot performs the commanded tasks.

This method of interaction extends the capabilities of the aerial robot to perform robustly in applications where verbal or physical interaction is virtually non-existent. Among such applications is aiding individuals with disabilities. People with disabilities that are non-verbal or don't normally express their needs via verbal interactions need an alternative way to do so. Tools such as picture cards have been used to provide a way for these people to express themselves which, in some cases, can serve as a bridge to verbal communication. This principle of communication for people who are not able to do it otherwise can be employed in the use of aerial robots, for example, to help in everyday tasks or as a guide.

Visual body interaction

In these experiments, different users were encouraged to interact and command the aerial robot by changing the position of their body.

As can be seen in Fig. 12.25, users interacted with the aerial robot in a number of ways including jumping in the air, turning around and even doing push-ups to test the limits of interaction.

The feedback received from the users, specially those who had no previous interaction with aerial systems was, in general, a positive one. Naturally, many users who tested this interaction felt doubtful at first,

Figure 12.25: This figure shows several users in different indoor scenarios interacting with the aerial robot using the position of their body.

understandably thinking that this machine who was following them, could lose control at any moment. At the beginning, users were cautious and a little frightened. They remained static, just looking at the aerial robot. Nevertheless, after some time interacting with the aerial robot, they began to feel very confident and forgetting that it was a robot, treating it as a living being. Users acknowledged that having such a close one-to-one interaction/relationship with the aerial robot gradually made them more and more comfortable being close to it.

After the flights had ended, many people who had experienced this interaction ended up sharing interesting applications for this type of interface ranging from personal trainers to personal cameramen for news reporting and documentaries. Reports like *I felt like it was my friend* or *I want a pet like this* were very common and by the end, people felt very upset when the aerial robot ran out of battery.

Hand gesture interaction

User interaction with this method was not easy at first. Users had to get the feeling of the relationship between the movement of their hand and the movement of the aerial robot while having to compensate the different dynamics of the hand motions relative to the aerial robots'. Nevertheless, experiencing the connection of the hand with the aerial robot made this the most natural and fun method to interact amongst users. This response from the users may be due to the fact that nowadays, basic mundane devices such as smartphones or tablets employ this type of behavior in their UIs and people are accustomed to dealing with such interactions.

Users of different ages have used the proposed NUI as seen in Fig. 12.26. Most of the people who used the interface regarded the interaction as a game which felt intuitive and entertaining. This NUI can be used in many different applications spanning from teleoperation of aerial systems in dangerous environments to quadrotor gaming systems.

Speech command interaction

This challenge was conducted to test the reliability and usefulness of the speech NUI for robot interaction. Table 12.1 details the commands that are presently recognized by the speech NUI. As was mentioned in Section 11.4.2, these commands are not fixed, they can be extended for any additional need the user might have in the future. The speech vocabulary was created to show the potential uses of the aerial robot as well as to test the response of the system in real-flight scenarios.

In general, users were drawn to this type of interaction. Most users expressed that talking to the aerial robot felt like how one would talk to a pet. Or more precisely, how one felt while training a pet. These responses to the interaction were exalted by the fact that voice feedback was implemented using a voice synthesizer to reply when commands were correctly received.

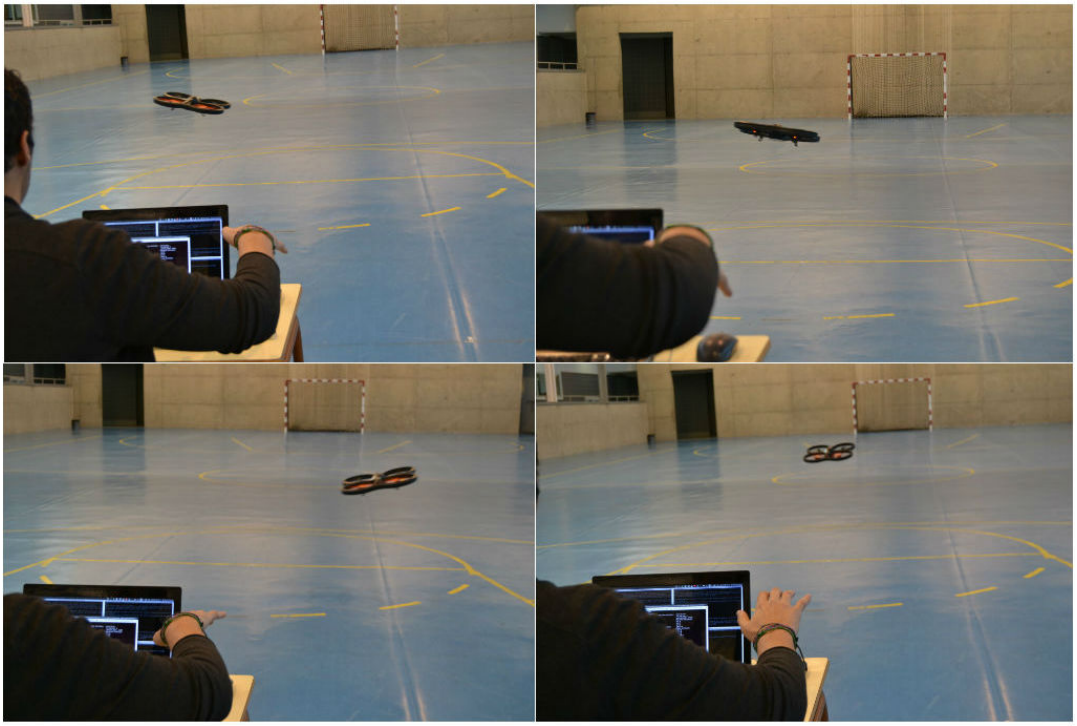


Figure 12.26: Human-robot hand gesture interaction being tested in indoor environments using the Leap Motion sensor.

Speech NUI Command List		
take-off	*land*	*sleep*
hover	*move right*	*move left*
rotate right	*rotate left*	*arm*
move forward	*move backward*	*flip*
start visual body	*start visual marker*	*start speeh*
start hand gesture	*start trajectory*	*pause speech*

Table 12.1: Speech NUI list of high-level commands.

Multi-modal interaction

In order to validate the full natural user interface system, multi-modal flight tests were performed using all of the available NUIs. In this way, multiple users can interact with the aerial robot in order to perform different tasks. In the test performed, one user was in charge of the speech interface and hand gesture interactions, while a non-skilled user was in charge of the body position interaction and the visual markers.

This scenario could easily represent rescue teams in disaster scenarios where a base operator stays with the ground station, monitoring the environment situation while interacting by voice or hand movements with the aerial robot. The base operator would then guide and/or aid the rescue operators perform their mission. After some event has occurred, the rescue operators can signal the aerial robot via visual markers to perform different tasks such as return home guiding the victims without having to exit the area and help others.

Future work

Future work in this line of research will target implementing the proposed interaction techniques on Human-Multi-Aerial-Robot Interaction. Here, users will have the ability to choose from different ways of interacting with multiple aerial robots simultaneously. This type of interaction could be useful for applications such as operator guidance of a fleet of autonomous air tankers used for aerial firefighting or massive search-and-rescue missions where one person or a team can interact with multiple aerial robots at the same time to aid in aerial imaging reconnaissance.

Related publications

This challenge is deeply described in (Suárez Fernández et al., 2016), presenting more details about the system setup of this set of experiments, together with a larger number of results.

12.3.4. Search and rescue

Challenge description

The last challenge tackles a complete search and rescue mission. This mega challenge combines all the previously described challenges and some mission elements developed for the international competitions, in a single complex mission.

The challenge described here is based on a search and rescue mission. In this mission, an emergency situation is emulated inside a house and the first response to the emergency, until the human rescue team arrives at the accident area, is demonstrated (Fig. 12.27).

Two exploratory fully autonomous aerial robots navigate from the rescue equipment base to the inside of the house, entering through an open window or door, searching for a target (e.g. an injured human subject).

As soon as a subject is detected, a rescue aerial robot is called, arriving at the position where the subject has been found. This rescue aerial robot has the responsibility of guaranteeing the survival of the subject while the human rescue team arrives at the area. The rescue aerial robot might, therefore, for example, deliver a first aid package, and track and monitor the human subject, interacting with him or her by means of natural interfaces. Thanks to this aerial robot, the rescue equipment is able to monitor the state of the injured person, being able to generate an efficient response.

If the emergency situation has been stabilized and the subject does not need assistance anymore, or if the human rescue team has arrived at the area, the rescue aerial robot might be commanded (by the subject or the rescue team) to come back to the rescue equipment base. In the meantime, the search aerial robots keep searching for more targets. As soon as the house is completely explored, or the emergency situation has been stabilized, the search aerial robot finish their exploratory mission, and they autonomously return to the rescue equipment base.

The reader must note that this mission proposal might correspond to a real search and rescue mission that is required to be executed by a fleet of aerial robots. Achieving a fully autonomously performance is a very challenging task.

Environment

Similarly than in the challenge presented in Section 12.3.1, the environment is simplified to an indoor structured static one formed by simple elements (poles and walls). The environment is augmented with visual markers, for simplifying the aerial robot localization, and for obstacles perception and mapping. The elements of the environment are therefore labeled with a set of unique previously known visual markers. The environment (number and position of the elements) might be previously unknown, being the responsibility of the robot, its mapping. The floor is assumed to be flat and to have enough visual features (like a texture). One moving textured object (the target subject) is assumed to be there statically placed until is found. The subject is allowed to move whenever the rescue robot has reached him/her until the emergency situation is labeled as stabilized, when the subject is assumed to be static again.

System setup

Hardware setup

In the same way than in the previous challenges, the preferred aerial platform is the Parrot AR.Drone 2.0. due its simplicity and safety, being the hardware setup similar. Nevertheless, other aerial vehicles such as an AscTec Pelican, a Mikrokopter Oktocopter, or a custom built one, has been used to demonstrate the good performance of this challenge.

The Parrot AR.Drone has two cameras (a front camera and a bottom camera), together with an IMU, a magnetometer, a barometer, and a down-looking ultrasound sensor. It includes an autopilot that estimates the aerial robot attitude (by means of the IMU and the magnetometer), its flight altitude (thanks to the ultrasound sensor and the barometer), its horizontal linear velocity (calculating the optical flow with the bottom camera), and its angular velocities and linear accelerations (filtering the IMU). The autopilot allows to command the aerial platform in attitude mode for the horizontal angles (pitch and roll), and in velocity mode for the vertical angle (yaw) and for the vertical movement (z). The autopilot includes as well some actions (and the required controllers to execute them) like take-off, hover and land.

As described in Section 12.2.2, every aerial platform is connected via Wi-Fi to a ground computer for all the processing tasks. All the ground computers are connected to a LAN by means of a switch.

System architecture setup

The proposed system architecture described in Chapter 3 is used, being the components employed available on Aerostack, as the following:

- *Perception*: The perception solution presented in Chapter 7 was used. Additionally, a TLD based tracker (Kalal et al., 2012b) was used for the detection and estimation of the target's pose.
- *Motor System*: The existing low-level embedded controllers, the navigation controllers, and the visual servoing controller available on Aerostack, and described in Section 11.1, were used.
- *Executive System*: the existing Executive System components available on Aerostack were used.
- *Planning System*: the existing Planning System components available on Aerostack were used, as the task-based sequential mission planner presented in Section 11.2, and the collision-free path planner described in Chapter 10.
- *Supervision System*: the Supervision System components available on Aerostack were used.
- *Communication System*: the Communication System components available on Aerostack were used, as the multi-modal user interfaces presented in Section 11.4.

Experimental results

The global goal is to search for a subject in a spatial area (Fig. 12.27) and to naturally interact with him or her. The global search goal is distributed in different local sub-goals with two different aerial robots. Each robot covers a different local search area. A third aerial robot is used for the natural interaction with the subject. The environment is an indoors area, with simple shape (walls and poles) static elements, augmented with visual markers (ArUco markers). Both the obstacles and the searched subject are uniquely labeled thanks to the visual markers.

In more detail, the mission is as follows:

- Two aerial robots take-off at the same time from one specific take-off point in the rescue equipment base.
- Each aerial robot covers a different search area defined as a set of waypoints as destination points.
- Unknown simple shape static obstacles are present and each aerial robot must detect them and avoid them. In addition, each aerial robot must avoid collisions with the other robots. The aerial robots must cross narrow areas and they must decide how to enter in the appropriate order to avoid collisions amongst themselves.
- When an aerial robot recognizes the presence of the subject, it notifies the third aerial robot with the location of the subject, and both initial aerial robots return to their respective take-off points and land.
- After receiving the notification about the location of the subject, the third aerial robot takes off from its specific take-off point at the rescue equipment base and approaches the subject.
- The third aerial robot remains near the subject and naturally interacts with him or her until receiving a command from the subject to return to its take-off point and land.

Fig. 12.27 shows different frames of the search and rescue mission. The rescue base (where the aerial robots take-off) is located on the left side of the images. The accident area is supposed to be the interior of a house, where the subject (highlighted in green) is supposed to be.

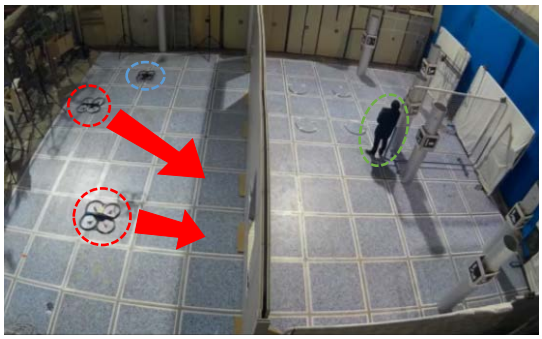
The trajectories followed by the aerial robots during the execution of the search and rescue mission are shown in Fig. 12.28. The two search aerial robot followed the trajectories plotted in red and green, while the rescue aerial robot followed the blue trajectory.

Due to the complexity of the mission and the difficulty to understand the trajectories shown in Fig. 12.28, the experiment is divided in the different instant times displayed in Fig. 12.29. These plots also represent with dashed lines, the collision-free planned path provided by the Planning System. Similarly than before, the two search aerial robots followed the trajectories plotted in red and green, while the rescue aerial robot followed the blue trajectory.

The search and rescue mission is executed as follows:

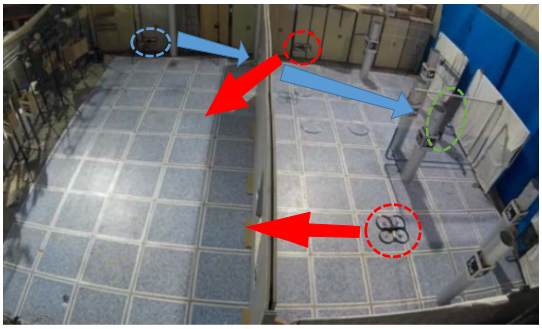
In Fig. 12.27a, the two search aerial robots (highlighted in red) take-off from the rescue base and start their exploratory mission, needing to access to the building (either through the window or the door). Fig. 12.29a shows the trajectories followed by the two search aerial robots at this time instant.

Once inside the building, every search aerial robot explores the accident area, searching for targets (see Fig. 12.27b). Every aerial robot explores a different part of the accident area as in the trajectories in Fig. 12.29b.

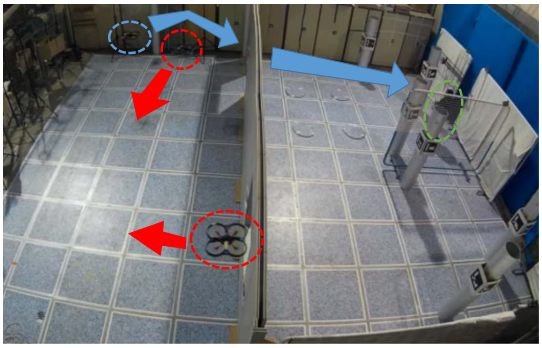


(a) The two search robots need to access to the building.

(b) The search robots explore the accident area searching for targets.

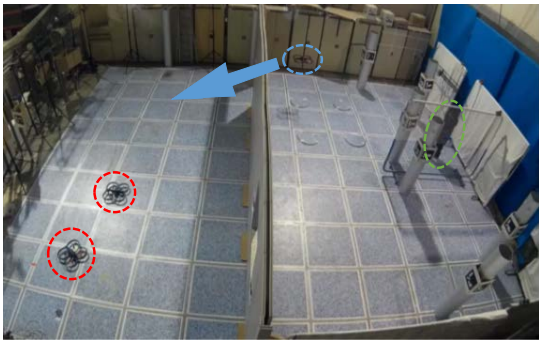


(c) The rescue robot has to reach the person, and the search robots conclude their exploratory mission.



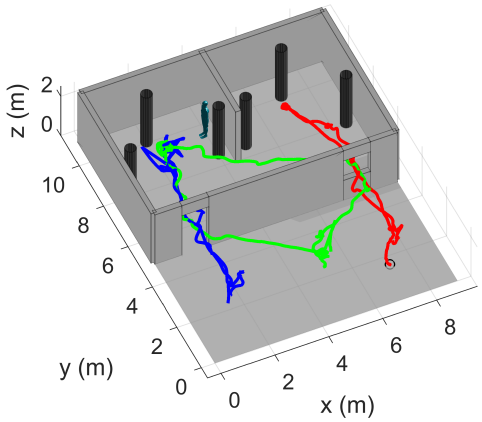
(d) The rescue robot waits until the accesses to the house are clear to avoid a collision.

(e) The rescue robot interacts with the person by means of natural interfaces.

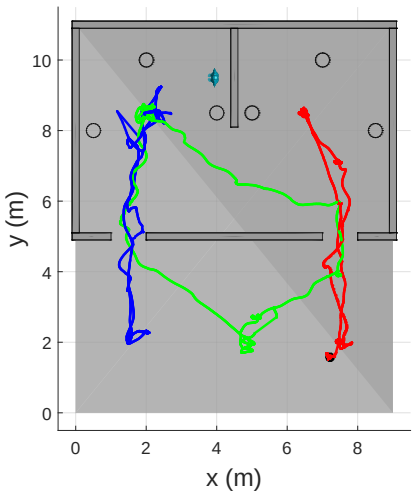


(f) The rescue robot returns to the rescue base.

Figure 12.27: Different frames of the search and rescue mission.



(a) 3D trajectory of the aerial robots in the search and rescue mission.



(b) 2D trajectory of the aerial robots in the search and rescue mission.

Figure 12.28: Trajectory followed by all the aerial robots in the entire search and rescue mission.

As soon as the subject is detected, the rescue aerial robot (highlighted in blue) takes off from the rescue base, with the objective to reach the subject (Fig. 12.27c). Fig. 12.29d shows the trajectories of the search aerial robots when the subject is detected. In parallel, the search aerial robots conclude their mission and come back to the rescue base. Although it is not needed, the subject has a visual marker to be properly identified by the search robots.

As can be seen in Fig. 12.27d and the corresponding trajectories in Fig. 12.29e, the search robot is faster leaving the house than the rescue robot entering in it, so to avoid a collision, the search robot autonomously waits until the accesses to the house are clear.

In Fig. 12.27e and Fig. 12.29g, the rescue robot interacts with the subject by means of natural interfaces.

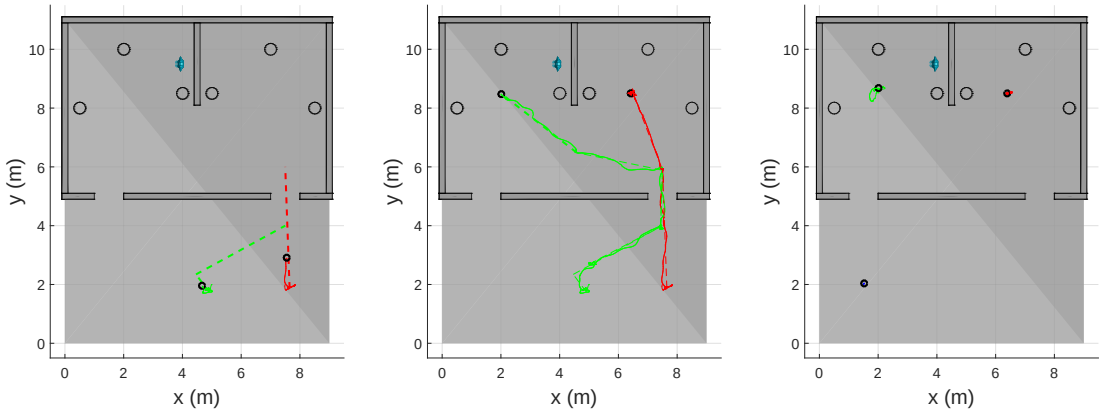
After the emergency situation is stabilized, the rescue aerial robot autonomously returns to the rescue base (Fig. 12.27f and Fig. 12.29h).

Future work

As this challenge include features from some of the previously analyzed ones, its possible future work lines are directly related to the future works of every individual challenges, ranging from the increase of the level of interactions between robotic agents and between the robotic agents and the humans; to the use of a more complex and realistic environment.

Related publications

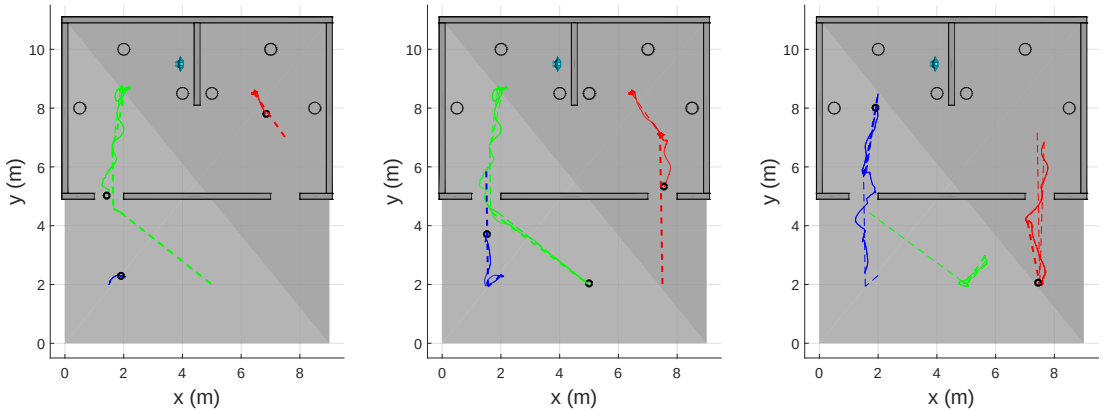
More details about the system setup of this challenge, together with a larger number of results are deeply presented in (Sanchez-Lopez et al., 2016c), and (Sanchez-Lopez et al., 2017b).



(a) 2D trajectories from time $t = 0$ s to $t = 15$ s.

(b) 2D trajectories from time $t = 0$ s to $t = 55$ s.

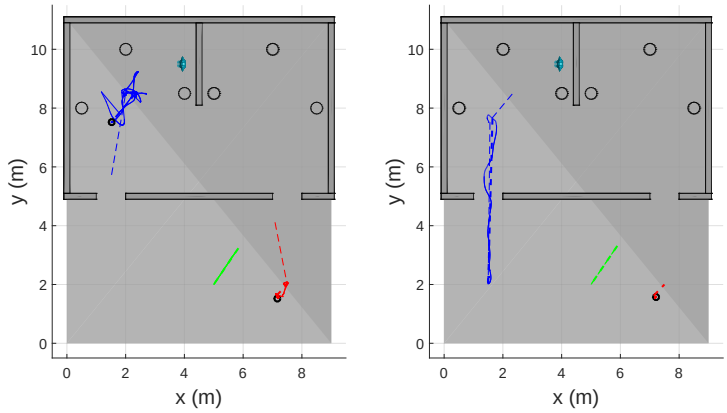
(c) 2D trajectories from time $t = 65$ s to $t = 75$ s.



(d) 2D trajectories from time $t = 65$ s to $t = 85$ s.

(e) 2D trajectories from time $t = 65$ s to $t = 98$ s.

(f) 2D trajectories from time $t = 95$ s to $t = 110$ s.



(g) 2D trajectories from time $t = 110$ s to $t = 160$ s.

(h) 2D trajectories from time $t = 160$ s to $t = 182.33$ s.

Figure 12.29: Different trajectories of the aerial robots in several time frames during the search and rescue experiment.

12.4. Public demonstrations

This section list the most distinguished public demonstrations where the proposed system architecture and Aerostack have been employed.

2014 and 2015 European Robotics Week (private events)

In the context of the 2014 and 2015 European Robotics Week[1] two private events were organized in the Centre of Automation and Robotics (CAR), where the Computer Vision Group (CVG) demonstrated their capabilities to a reduced specialized public. The demonstration consisted on a visual object following of a person, as described in Section 12.3.2.

2014 Madrid Science Week

In the context of the 2014 Madrid Science Week[2], the Computer Vision Group (CVG) demonstrated to several groups of kids and teenagers a visual object following of a person demo (Section 12.3.2) and a basic multi-robot navigation demo with a simple environment and a reduced number of aerial robots (Section 12.3.1). Not only the demonstrator interacted with the aerial robots but also a group of selected attendees was allowed to interact with them (as can be seen on Fig. 12.30).

Additionally, a similar but a little more special demonstration was carried out for a group of mentally disabled people[3].

The total number of attendees was greater than 400 people.

(a) with kids. (b) with mentally disabled people.

Figure 12.30: 2014 Madrid Science Week.

2015 European Researchers' Night in Madrid

In the context of the 2015 European Researchers' Night[4], the Technical University of Madrid (UPM) organized in Madrid a show where all the robots were exposed.

The Computer Vision Group (CVG) carried out an exhibition (see Fig. 12.31) with the aerial robots[5]. A mixed age non-specialized audience of more than 1000 people attended this demonstration where the following shows were executed:

- A basic multi-robot navigation mission (Section 12.3.1)
- A single aerial robot search and rescue mission (Section 12.3.4)

Additionally, some selected attendees were chosen to interact with the aerial robots by means of natural user interfaces (as described in Section 12.3.3):

[1]Online: http://www.eurobotics-project.eu/eurobotics-week/eurobotics-week-.html
[2]Online: http://www.madrimasd.org/semanaciencia/2014/
[3]Online: http://www.escuelaindustrialesupm.com/responsabilidad-social/ciencia-y-discapacidad-erase-una-vez-un-parque
[4]Online: http://ec.europa.eu/research/researchersnight/index_en.htm
[5]Online: http://www.madrimasd.org/lanochedelosinvestigadores2015/actividad/vuelo-de-drones-y-m%C3%A1s-robots-asombrosos?lan=en

(a) with the president of the Community of Madrid.

(b) demonstration of a search and rescue mission.

(c) a person interacting with the aerial robot.

Figure 12.31: 2015 European Researchers' Night in Madrid.

CivilDRON 2016 and 2017

The CivilDRON congress is a National Spanish congress[6], specialized in unmanned aerial systems.

In the CivilDRON 2016 congress, a multi-robot exploration and inspection mission (Section 12.3.1), and a visual object following of a person demo (Section 12.3.2) were carried out in front of a specialized public of more than 200 people (Fig. 12.32a).

In the CivilDRON 2017 congress, a simplified version of the IMAV 2016 competition mission (Section 12.2.4) was executed in front of a specialized public of more than 200 people (Fig. 12.32b).

(a) CivilDRON 2016. Multi-robot inspection mission.

(b) CivilDRON 2017. IMAV 2016 competition mission.

Figure 12.32: CivilDRON 2016 and 2017 congresses.

12.5. Software metrics

This section provides a number of quantitative evidences that demonstrate some performance and quality features of the software framework.

Modularity: number of software packages

Aerostack is a modular and specialized software with 100 software packages (defined as ROS packages), apart from multiple standard ROS packages (see Table 12.2). As stated in Section 5.2, Aerostack organizes its packages taking into account their functionality and status, in different processes and subsystems.

[6]Online: `http://www.civildron.com/`

Package group	Modified ROS Packages	Aerostack Packages
stack	4	68
stack_devel	3	17
stack_deprecated	-	8
stack_obsolete	-	23
Total	7	93+23

Table 12.2: Summary of the available number of packages of the Aerostack software framework.

Distributed processing: number of running processes

As described in Section 5.1, Aerostack uses asynchronous multitasking, where different processes run concurrently, with inter-process communication provided by ROS. Depending on the application, one can execute from 10 to 50 processes simultaneously per robotic agent in a single computer or along multiple distributed computers. The computational needs are highly dependent on the specific implementation of the used modules, but, as stated in Section 5.4, the existing components in Aerostack have run in a wide range of different computers.

Scalability, usability and evolution along its life

The examples presented in this chapter, where Aerostack has been used, contribute to demonstrate other software quality features of Aerostack such as usability and scalability. For example, the usability has been demonstrated with the fact that more than 30 known users and developers have successfully used the software framework in different practical applications. Usability in Aerostack is provided by its modularity and a uniform documentation, both in source code and text documents. Aerostack counts also with manuals and tutorials that present the main aspects of Aerostack with cases of uses and examples ranging from basic users to developers.

The experience with Aerostack has also proved its scalability. In the last four years, since the original implementation (in February 2013), Aerostack has grown gradually by including new components for more complex problems. The first public release counted with 41 software packages, while the current version (at the moment of writing these lines) has more than double this amount, 100 software packages. It is worth to remark that Aerostack is alive and evolving product supported by the Computer Vision and Aerial Robotics (CVAR) research group that keeps updating the software framework and adding new components and functionalities.

12.6. Discussion

This section briefly analyzes the results presented in this thesis, mainly in this chapter and along Part III.

Evaluation of the proposed algorithms

The proposed algorithms, presented along Part III, have been evaluated by means of local simulations and real experiments. Moreover, they have been successfully integrated into a complete system executing different missions, such as international aerial robotic competitions, self-proposed challenges, and public demonstrated.

They have demonstrated their correct performance by means of a successful local and global execution, allowing the accomplishment of different missions in different environments.

Evaluation of the system architecture

The proposed system architecture, presented in Chapters 3 to 4, has been evaluated during more than four years (since February 2013) by means of the participation in international aerial robotics competitions, performing several self-proposed challenges, and exhibiting it in multiple public demonstrations.

The different faced challenges demonstrated the validity of the proposed system architecture for the fully autonomous operation of a heterogeneous aerial multi-robot system. It outperformed in a wide range of environments, completing the objectives of different kinds of mission, and with very different hardware setups, always fulfilling the main requirements of reliability and safety.

To the knowledge of the author, no other system architecture describes a fully versatile system architecture that enables the autonomous operation of a multi-robot (i.e. several robotic agents at the same time), multi-platform (i.e. heterogeneous robotic agents) system of aerial platforms (with special focus on multirotor platforms).

Evaluation of the software framework

The proposed software framework, described in Chapter 5, has been used during more than four years (since February 2013) to implement the system architecture and the proposed algorithms.

It has demonstrated the desired features of modularity, scalability, and usability, being the perfect tool to enable the fully autonomous operation of aerial robots.

Chapter 13

Conclusions and Future Work

13.1. Conclusions

The market of the micro aerial vehicles is growing at an exponential pace. There are multiple challenges that might be solved to achieve the fully autonomous navigation of a fleet of aerial robots performing complex dynamic missions in challenging unstructured environments, to simplify their use, and to extend their utilization to a great number of applications.

Most of the research works focus on solving one of the individual problems such as an efficient hardware design; precise feature extraction, state estimation and mapping algorithms; accurate control algorithms; fast and efficient planning and task execution; complex intelligence to deal with dynamic and changing situations; multi-robot interaction; and human-robot interaction. They have succeeded with their research tasks and they have pushed the state of the art in these fields of knowledge.

Nevertheless, there is still an open challenge that has not been tackled so far: the integration of all these components in a multi-robot fully autonomous intelligent system. Open-source architecture frameworks for aerial systems have already been developed, but they still present two main weakness: (1) in most of the cases, the acquired level of autonomy is limited, focusing on semi-autonomous missions. (2) versatility is typically restricted, being limited to some applications or aerial platforms.

The main contributions of this thesis are aligned with the objective of enabling and facilitating the fully autonomous intelligent navigation of a heterogeneous aerial multi-robot system for different complexity and kinds of missions and environments. The methodology followed in this thesis was based on an iterative process that comprises the following stages: (1) challenge, (2) proposal, (3) evaluation, (4) analysis, and (5) generalization, inspired by the system engineering. This thesis has presented the three contributions described bellow: (1) a versatile system architecture for aerial robotics, (2) several algorithms to increase the autonomy level of the aerial robotic systems, and (3) a software framework for aerial robotics.

System architecture for aerial robotics

This thesis has proposed a versatile system architecture for aerial robotics that allows a fully autonomous operation for a aerial multi-robot system. This system architecture is flexible, fulfilling the requirements of being mission, platform, and environment agnostic. It is characterized in an abstract and general level, defining only to top-level functionalities, by means of:

- Definition of its main subsystems and their functionalities. Subsystems are defined along natural boundaries, minimizing the amount of information exchanged between the subsystems, achieving a high-degree of modularity. The subsystems and their functionalities are defined in a top-level way, to guarantee the versatility and flexibility of the system architecture. This system architecture gathers the proposed subsystems in seven groups: Feature Extraction System, Motor System, Situation Awareness System, Executive System, Planning System, Supervision System, and Communication System. Moreover, these subsystems are gathered with a five layered paradigm: reactive, executive, deliberative, reflective, and social, that provides an increasing level of intelligent behavior by means of an increasing symbolic level.
- Specification of the interfaces between subsystems. These interfaces are unequivocally defined, by means of an ontology, avoiding ambiguity that might lead to errors and malfunctioning. Nevertheless, they are general enough to respond to the requirements of mission, platform, and environment agnostic.

This system architecture provides system designers the initial architecture for developing their own fully autonomous intelligent aerial multi-robot systems, guiding them through this complex process and speeding up the development of aerial robotic systems.

The validation of the proposed system architecture is a complex task since the performance on the fully autonomous intelligent navigation of a heterogeneous aerial multi-robot system is highly dependent on the kind of the mission executed, the environment where the system has to move, the hardware setup, the algorithms employed, and their software implementation.

Three kinds of scenarios were used to provide a global evaluation of the complete system:

- *International aerial robotics competitions* provide a mission and an environment defined externally by the organizers that cannot be modified. Additionally, the complete system has to work in a safe, and reliable way the day of the competition. Moreover, an external evaluation given by the referees and the comparison with the other teams provide enough evidences of the performance of the system compared with the current state of the art.
- *Self-proposed challenges* allow pushing the proposed system to its functionality limits being their objectives adapted to the particular research line that is wanted to be investigated, with an unlimited level of difficulty. Moreover, they do not present a hard deadline, and the reliability and safety of the system are minor requirements.
- *Public demonstrations* add other extra requirements not tackled in the other scenarios, mainly the need of a high level of safety since the system is presented in front of a large number of people, including kids.

The success achieved in the international competitions, the proposed challenges, and the public demonstrations validated the performance of the proposed system architecture to achieve the fully autonomous navigation of a fleet of aerial robots performing complex dynamic missions in challenging unstructured environments.

Proposed algorithms to increase the autonomy level of the aerial robotic systems

The second contribution of this thesis is the development of different algorithms that provides an increased level of the autonomy for the aerial robotic systems. These algorithms were developed in the context of several particular applications, and they are therefore highly dependent on the hardware setup, the mission required to be executed, or the environment where the robot has to move. This thesis has presented several algorithms for the autonomous operation of aerial multi-robot systems, with different level of detail, gathered in the following five groups:

- Perception algorithms, including feature extraction, state estimation and mapping algorithms for different applications and robots ensuring the adequate estimation of the state of the robot and the environment.
- Control algorithms for different applications and robots, which ensure that the commanded references are correctly tracked and transformed into actuator commands.
- Planning and task execution algorithms for different applications and robots that permit the development of complex missions in complex environments.
- Intelligent and cognitive algorithms that ensure the adaptation of the robots to changing environments, system failures, or dynamic missions.
- Social and interaction capabilities between robotics agents and human agents, guaranteeing the safe and efficient coexistence and cooperation.

More concretely, the most important components presented in this thesis are the following:

- Helipad detection and reconstruction for shipboard landing, where an helipad mark is detected using computer vision techniques and its 3D pose is extracted by exploiting the knowledge of the dimensions of the mark.
- Perception based on odometry and visual markers with environment reconstruction that allows the estimation of the state of the aerial robot using visual markers together with the measurements provided

by several sensors. In addition, this solution provides an environment reconstruction formed by geometric primitives.
- Perception based on odometry and computer vision for gridded maps that allows the estimation of the state of the aerial robot for an environment with a grid, as in the mission 7 of the IARC Competition (see Section 12.2.3 for more information about the competition).
- Perception based on multi-sensor fusion with environment reconstruction that allows a versatile and robust estimation of the state of the aerial robot, by combining the information coming from different sources such as sensors, feature extraction components, and state estimation algorithms (such as other positioning or SLAM algorithms). In addition, this solution provides an environment reconstruction formed by geometric primitives.
- Collision-free path planning for dynamic environments, that generates collision-free trajectories to reach a specific point of a dynamic and changing 2D environment, using the knowledge of the current state of the robot and the map.

Other components presented with a lower level of detail, used only for validation purposes, are the following: multirotor controllers, mission planners, an action specialist, and different multi-modal user interfaces.

All these components were evaluated individually, demonstrating their performance. Nevertheless, the importance of these components appeared when integrated into the complete system.

Software framework for aerial robotics

The last contribution of this thesis is the development of a software framework, called *Aerostack*, that facilitates the efficient implementation of the designed system architecture and the developed algorithms by means of software components. The software is organized in a modular way using the concept of software packages, and gathering them by their functionality, their dependencies, and their life state. The proposed software framework relies on the widely used ROS middleware for interprocess communication. Additionally, the software framework uses an asynchronous multiprocess paradigm where every elementary functionality is implemented as a single process. This provides reliability against failures and simplicity in the development but also allows a distributed processing.

The proposed software framework has demonstrated to be versatile and scalable, where the developers can reuse as many software modules as possible, and the developers can modify or develop new modules without the need of changing any other components. This is translated into a reduction of the development time and cost.

Finally, to reach the largest possible number of people, contributing to the scientific community, the software framework has been made open-source.

13.2. Future work

As the main objective of the thesis is very challenging, and the work carried out to achieve it has been very variate, the possible future work is as well very diverse.

First of all, further improvements might be done in the proposed system architecture. Despite trying to target all the possible applications that could be carried out by aerial robots, considering different environments, the system architecture has not been tested in all of them. Applications such as aerial manipulation or physical interaction have been omitted and the proposed system architecture might require minor adaptations, for instance, new component functionalities or new messages of on the ontology.

The second possible future work is related to the software framework. Its design, development, and implementation have been done as a small software project, that has been grown in a controlled and centralized way. Nevertheless, multiple steps executed on big software projects has been omitted. Developing a proper software engineering on the software framework, including such as software design diagrams, or planning its life cycle, or giving it the adequate diffusion, are included in the future work to permit a further controlled distributed growing of the software framework. Additionally, despite working perfectly and being completely integrated, some of the existing components require to be adapted to the last version of the system architecture and software framework, and some others require to be re-coded using a more efficient programming.

The last future work line is connected to the proposed algorithms for increasing the autonomy level of the aerial robotic systems. The existing components have been developed to fulfill a specific application, working in a particular environment for a concrete hardware setup. Developing novel components that either substitute the current ones providing a higher versatility, or implement new functionalities, is an immediate research line. Indeed, this is one of the advantages of using Aerostack, that allows the developers to focus on their own interest work but with the possibility of testing their new components in a fully autonomous system. For example, a

developer might develop a versatile vision-based SLAM algorithm that might substitute the current localization and mapping components to perform a particular mission in a variate number of environments.

Appendices

Appendix A

3D Transformations

A.1. Representation of rotations

A rotation is a motion of a certain space that preserves at least one point. Mathematically, a rotation is a map. All rotations about a fixed point called the center of rotation, form a group under composition called the rotation group (of a particular space). This rotation group is a Lie group of rotations about the center of rotation. In physics, the concept of rotation is understood as a coordinate transformation (of an orthonormal basis).

Rotations in a three-dimensional euclidean space are not commutative, so the order in which rotations are applied is important. According to Euler's rotation theorem, the rotation of a rigid body (or three-dimensional coordinate system) is described by a single rotation about some axis. Such a rotation may be uniquely described by a minimum of three real parameters.

A three-dimensional rotation can be specified in a number of ways, (Shuster, 1993). Many of these representations use more than the necessary minimum of three parameters, although each of them still has only three degrees of freedom. Examples of representations, described bellow, are:

- Euler angles
- Axis-angle
- Rotation matrix
- Quaternion
- ...

A.1.1. Euler angles

The idea behind Euler rotations is to split the complete rotation of the coordinate system into three simpler constitutive rotations, being each one of them an increment on one of the Euler angles. However, the definition of Euler angles is not unique and in the literature, many different conventions are used. These conventions depend on the axes about which the rotations are carried out, and their sequence (since rotations are not commutative). There are 12 possible definitions of the Euler angles, as indicated in Table A.1. In aviation and aerial robotics, the orientation of the aircraft is usually expressed as intrinsic Tait-Bryan angles following w(yaw)-v(pitch)-u(roll) convention.

Despite its compactness (using only three parameters to represent the three degrees of freedom), they cannot concatenate rotations easily. They also present the gimbal lock problem.

intrinsic rotation	extrinsic rotation
wuw	zxz
uvu	xyx
vwv	yzy
wvw	zyz
uwu	xzx
vuv	yxy

Table A.1: Different Euler angles definitions.

A.1.2. Axis-angle

From Euler's rotation theorem, it is known that any rotation can be expressed as a single rotation about some axis. The axis is the unit vector (unique except for sign) which remains unchanged by the rotation. The magnitude of the angle is also unique, with its sign being determined by the sign of the rotation axis.

The rotation vector, or Euler vector, $\boldsymbol{v} = \phi \cdot \boldsymbol{u}$, represents the right-handed rotation of ϕ rad around the axis given by the unit vector $\boldsymbol{u} = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix}^T$.

Combining two successive rotations, each represented by a Euler axis and angle, is not straightforward.

A.1.3. Rotation matrices

A rotation matrix, also called direction cosine matrix, is a 3×3 matrix (in the tridimensional Euclidean space) whose columns define the unit vectors of the orthonormal basis of the rotated reference frame in coordinates of the fixed reference frame:

$$\boldsymbol{R}_R^W = \begin{bmatrix} \boldsymbol{i}_R^W & \boldsymbol{j}_R^W & \boldsymbol{k}_R^W \end{bmatrix} \tag{A.1}$$

Additionally, its rows, define the unit vectors of the orthonormal basis of the fixed reference frame in coordinates of the rotated reference frame:

$$\boldsymbol{R}_R^W = \begin{bmatrix} \left(\boldsymbol{i}_W^R\right)^T \\ \left(\boldsymbol{j}_W^R\right)^T \\ \left(\boldsymbol{k}_W^R\right)^T \end{bmatrix} \tag{A.2}$$

As it is defined an orthonormal basis, the rotation matrix is an orthonormal matrix, fulfilling the following properties (among others):

- $\boldsymbol{R}^{-1} = \boldsymbol{R}^T$
- $det(\boldsymbol{R}) = 1$

The concatenation of rotations is done thanks to the matrix multiplication:

$$\boldsymbol{R}_{1\to 2} = \boldsymbol{R}_1 \cdot \boldsymbol{R}_2 \tag{A.3}$$

The main drawback of this representation is its high degree of redundancy, requiring 9 elements to represent 3 degrees of freedom, which might lead to inconsistencies due to computation rounding errors.

A.1.4. Quaternions

The reader is referred to Appendix B for a complete description of the quaternions and their algebra.

A.1.5. Conversions

The reader is referred to (Barrientos et al., 1997) for a complete review on conversions between attitude representations. Also the reader is referred to Appendix B for quaternion conversions.

A.2. Representation of poses

To represent a pose, it is needed to take into account the position (3 degrees of freedom), the attitude (3 degrees of freedom), and the scale (1 degree of freedom). A total of $3+3+1=7$ degrees of freedom need to be represented.

The pose of frame A in coordinates of frame B is represented by: p_A^B.

There exist multiple formalisms, described bellow, that can be used to represent poses, such as:

- VAS: vector, Euler angles, scale.
- VRS: vector, rotation matrix, scale.
- VQS: vector, quaternion, scale.
- HTM: Homogeneous transformation matrix.
- ...

A.2.1. VAS: vector, Euler angles, and scale

The pose is represented by a set of three elements: a 3×1 vector to represent the position, an element with 3 numbers to represent the attitude as Euler angles, and a scalar number to represent the scale:

$$p_A^B = \left\{ \boldsymbol{t}_A^B, \boldsymbol{\psi}_A^B, s_A^B \right\} \tag{A.4}$$

Its main advantage is its compactness because it represents 7 degrees of freedom with only 7 elements. Its main disadvantages are due to the usage of Euler angles for representing the attitude, such as the ambiguity due to the large number of representation, the impossibility to concatenate rotations, and the gimbal lock problem.

A.2.2. VRS: vector, rotation matrix, and scale

The pose is represented by a set of three elements: a 3×1 vector to represent the position, a 3×3 rotation matrix to represent the attitude, and a scalar number to represent the scale:

$$p_A^B = \left\{ \boldsymbol{t}_A^B, \boldsymbol{R}_A^B, s_A^B \right\} \tag{A.5}$$

Its main disadvantage is its high degree of redundancy, requiring $3 \cdot 3 + 3 + 1 = 13$ elements to represent 7 degrees of freedom, which might lead to inconsistencies due to computation rounding errors. As strengths, it allows the composition and inversion operations, as shown in Appendix A.3.

A.2.3. VQS: vector, quaternion, and scale

The pose is represented by a set of three elements: a 3×1 vector to represent the position, a quaternion to represent the attitude, and a scalar number to represent the scale:

$$p_A^B = \left\{ \boldsymbol{t}_A^B, \boldsymbol{q}_A^B, s_A^B \right\} \tag{A.6}$$

Its main advantage is its relatively high compactness since it represents 7 degrees of freedom with only 8 elements. As strengths, it allows the composition and inversion operations, as shown in Appendix A.3.

A.2.4. HTM: Homogeneous transformation matrix

The pose is represented by a 4×4 matrix with $3 \cdot 3 + 3 + 1 = 13$ non-zero elements, called homogeneous transformation matrix with some special properties:

$$p_A^B = \left\{ \boldsymbol{H}_A^B \right\} \tag{A.7}$$

Its main disadvantage is its high degree of redundancy, requiring 13 elements to represent 7 degrees of freedom, which might lead to inconsistencies due to computation rounding errors. Its main advantage is the simplicity of its composition and inversion operations, as shown in Appendix A.3.

A.2.5. Conversions

VAS, VRS and VQS

The conversion of a pose representation between VAS, VRS and VQS is very simple, since they share two of the three elements of the set, and only the attitude representation needs to be converted:

$$\begin{cases} \boldsymbol{t}_{VAS} = \boldsymbol{t}_{VRS} = \boldsymbol{t}_{VQS} & \text{(A.8)} \\ \boldsymbol{\psi}_{VAS} \leftrightarrow \boldsymbol{R}_{VRS} \leftrightarrow \boldsymbol{q}_{VQS} & \text{(A.9)} \\ s_{VAS} = s_{VRS} = s_{VQS} & \text{(A.10)} \end{cases}$$

V⋆S vs. HTM

To convert a pose representation between any V⋆S and HTM, and intermediate transformation to VRS has to be done, as:

$$V \star S \leftrightarrow VRS \leftrightarrow HTM \tag{A.11}$$

To convert a pose representation from VRS to HTM:

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}_{1\times 3} & s \end{bmatrix} \tag{A.12}$$

To convert a pose representation from HTM to VRS:

$$\begin{cases} \boldsymbol{t} = \boldsymbol{H}_{i4} & \forall i = 1..3 \quad (A.13) \\ \boldsymbol{R} = \boldsymbol{H}_{ij} & \forall i,j = 1..3 \quad (A.14) \\ s = \boldsymbol{H}_{44} & (A.15) \end{cases}$$

A.3. Poses algebra

This section presents the two elementary operations that can be used when operating with poses: composition and inversion.

A.3.1. Composition of poses

Having the situation represented in Fig. A.1, where the pose of A in coordinates of B, p_A^B, and the pose of B in coordinates of C, p_A^C are known, the pose of A in coordinates of C, p_A^C, can be obtained by composition as:

$$p_A^C = p_B^C \oplus p_A^B \tag{A.16}$$

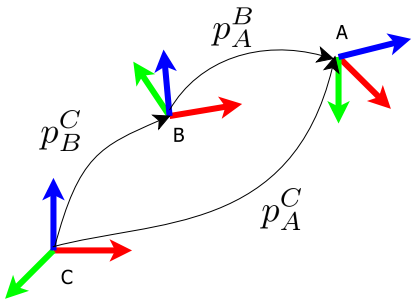


Figure A.1: Composition of poses: $p_A^C = p_B^C \oplus p_A^B$.

Properties

The composition of poses has the following properties:

- Associative:

$$p_D^A = p_C^A \oplus p_D^C = \left(p_B^A \oplus p_C^B\right) \oplus p_D^C = p_B^A \oplus p_C^B \oplus p_D^C = p_B^A \oplus \left(p_C^B \oplus p_D^C\right) = p_B^A \oplus p_D^B = p_D^A \tag{A.17}$$

- No commutative, as rotation is not commutative:

$$p_A^C = p_B^C \oplus p_A^B \neq p_A^B \oplus p_B^C \tag{A.18}$$

- Neutral element, p_I, that satisfies:

$$p_A^B = p_A^B \oplus p_I = p_I \oplus p_A^B \tag{A.19}$$

Using HTM

Having $p_A^B = \{\boldsymbol{H}_A^B\}$, and $p_B^C = \{\boldsymbol{H}_B^C\}$, then, $p_A^C = \{\boldsymbol{H}_A^C\}$ is given by:

$$\boldsymbol{H}_A^C = \boldsymbol{H}_B^C \cdot \boldsymbol{H}_A^B \tag{A.20}$$

The neutral element, p_I, is:

$$\boldsymbol{H}_I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.21}$$

Using VRS

Having $p_A^B = \{\boldsymbol{t}_A^B, \boldsymbol{R}_A^B, s_A^B\}$, and $p_B^C = \{\boldsymbol{t}_B^C, \boldsymbol{R}_B^C, s_B^C\}$, then $p_A^C = \{\boldsymbol{t}_A^C, \boldsymbol{R}_A^C, s_A^C\}$, is given by:

$$\begin{cases} \boldsymbol{t}_A^C = \boldsymbol{R}_B^C \cdot \boldsymbol{t}_A^B + s_A^B \cdot \boldsymbol{t}_B^C & \text{(A.22)} \\ \boldsymbol{R}_A^C = \boldsymbol{R}_B^C \cdot \boldsymbol{R}_A^B & \text{(A.23)} \\ s_A^C = s_B^C \cdot s_A^B & \text{(A.24)} \end{cases}$$

The neutral element, p_I, is:

$$\begin{cases} \boldsymbol{t}_I = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T & \text{(A.25)} \\ \boldsymbol{R}_I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \text{(A.26)} \\ s_I = 1 & \text{(A.27)} \end{cases}$$

Using VQS

Having $p_A^B = \{\boldsymbol{t}_A^B, \boldsymbol{q}_A^B, s_A^B\}$, and $p_B^C = \{\boldsymbol{t}_B^C, \boldsymbol{q}_B^C, s_B^C\}$, then $p_A^C = \{\boldsymbol{t}_A^C, \boldsymbol{q}_A^C, s_A^C\}$, is given by:

$$\begin{cases} \bar{\boldsymbol{t}}_A^C = \boldsymbol{q}_B^C \otimes \bar{\boldsymbol{t}}_A^B \otimes \left(\boldsymbol{q}_B^C\right)^* + s_A^B \cdot \bar{\boldsymbol{t}}_B^C & \text{(A.28)} \\ \boldsymbol{q}_A^C = \boldsymbol{q}_B^C \otimes \boldsymbol{q}_A^B & \text{(A.29)} \\ s_A^C = s_B^C \cdot s_A^B & \text{(A.30)} \end{cases}$$

where $\bar{\boldsymbol{t}}_i^j = \begin{bmatrix} 0 \\ \boldsymbol{t}_i^j \end{bmatrix}$.

The neutral element, p_I, is:

$$\begin{cases} \boldsymbol{t}_I = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T & \text{(A.31)} \\ \boldsymbol{q}_I = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T & \text{(A.32)} \\ s_I = 1 & \text{(A.33)} \end{cases}$$

A.3.2. Inversion of a pose

Having the situation represented in Fig. A.2, where the pose of A in coordinates of B, p_A^B, is known, the pose of B in coordinates of A, p_B^A, can be obtained by inversion as:

$$p_B^A = \ominus p_A^B \tag{A.34}$$

Properties

The inversion of a pose has the following properties:

- Anti-associative with respect to the composition:

$$p_A^C = \ominus p_C^A = \ominus(p_B^A \oplus p_C^B) = \left(\ominus p_C^B\right) \oplus \left(\ominus p_B^A\right) = p_B^C \oplus p_A^B = p_A^C \tag{A.35}$$

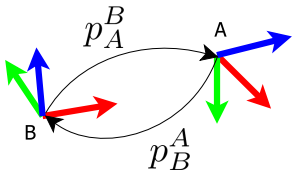


Figure A.2: Inversion of a pose: $p_B^A = \ominus p_A^B$.

Using HTM

Having $p_A^B = \{\boldsymbol{H}_A^B\}$, then $p_B^A = \{\boldsymbol{H}_B^A\}$, is given by:

$$\boldsymbol{H}_B^A = \left(\boldsymbol{H}_A^B\right)^{-1} \tag{A.36}$$

Using VRS

Having $p_A^B = \{\boldsymbol{t}_A^B, \boldsymbol{R}_A^B, s_A^B\}$, then $p_B^A = \{\boldsymbol{t}_B^A, \boldsymbol{R}_B^A, s_B^A\}$, is given by:

$$\begin{cases} \boldsymbol{t}_B^A = -\frac{1}{s_A^B} \cdot \left(\boldsymbol{R}_A^B\right)^T \cdot \boldsymbol{t}_A^B & \text{(A.37)} \\ \boldsymbol{R}_B^A = \left(\boldsymbol{R}_A^B\right)^{-1} = \left(\boldsymbol{R}_A^B\right)^T & \text{(A.38)} \\ s_B^A = \frac{1}{s_A^B} & \text{(A.39)} \end{cases}$$

Using VQS

Having $p_A^B = \{\boldsymbol{t}_A^B, \boldsymbol{q}_A^B, s_A^B\}$, then $p_B^A = \{\boldsymbol{t}_B^A, \boldsymbol{q}_B^A, s_B^A\}$, is given by:

$$\begin{cases} \bar{\boldsymbol{t}}_B^A = -\frac{1}{s_A^B} \cdot \left(\boldsymbol{q}_A^B\right)^* \otimes \bar{\boldsymbol{t}}_A^B \otimes \boldsymbol{q}_A^B & \text{(A.40)} \\ \boldsymbol{q}_B^A = \left(\boldsymbol{q}_A^B\right)^* & \text{(A.41)} \\ s_B^A = \frac{1}{s_A^B} & \text{(A.42)} \end{cases}$$

where $\bar{\boldsymbol{t}}_i^j = \begin{bmatrix} 0 \\ \boldsymbol{t}_i^j \end{bmatrix}$.

Appendix B

Algebra of the Quaternions

In this appendix, the quaternion algebra is summed up, avoiding detailed demonstrations. Some of the information gathered in this appendix has been taken from (Sola, 2016; Trawny and Roumeliotis, 2005).

B.1. Definition

The quaternions were invented by William Rowan Hamilton in 1843 in Dublin, Ireland. To define a quaternion, its basis elements need to be previously defined.

Basis elements

Three imaginary unit numbers, i, j, and k, are defined, following:

$$i^2 = -1 \quad j^2 = -1 \quad k^2 = -1 \quad i \cdot j \cdot k = -1 \tag{B.1}$$

from which can be derived:

$$i \cdot j = -j \cdot i = k \quad j \cdot k = -k \cdot j = i \quad k \cdot i = -i \cdot k = j \tag{B.2}$$

Axiomatic property of the basis elements

Being r a scalar number, $r \in \mathbb{R}$:

$$r \cdot i = i \cdot r \quad r \cdot j = j \cdot r \quad r \cdot k = k \cdot r \tag{B.3}$$

Quaternion definition

Being q_w, q_x, q_y, q_z scalar numbers, $q_w, q_x, q_y, q_z \in \mathbb{R}$, a quaternion $\boldsymbol{q} \in \mathbb{H}$ is defined as:

$$\boldsymbol{q} = q_w + q_x \cdot i + q_y \cdot j + q_z \cdot k \tag{B.4}$$

Representations of quaternions

A quaternion $\boldsymbol{q} \in \mathbb{H}$ can be represented, abusing of the notation, in the following ways:

- As a sum of a scalar and a vector:

$$\boldsymbol{q} = q_w + \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} = q_w + \boldsymbol{q_v} \tag{B.5}$$

  Being q_w the real or scalar part (also represented as q_s or q_r); and $\boldsymbol{q_v}$ the imaginary or vector part.
- As an ordered pair scalar-vector:

$$\boldsymbol{q} = < q_w, \boldsymbol{q_v} > \tag{B.6}$$

- As a vector:

$$\boldsymbol{q} = \begin{bmatrix} q_w \\ \boldsymbol{q_v} \end{bmatrix} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} \tag{B.7}$$

Special quaternions

Two special quaternions $\boldsymbol{q} \in \mathbb{H}$ can be defined:

- **Real quaternion**: $\boldsymbol{q}$ is a real quaternion if its imaginary part is null, $\boldsymbol{q_v} = \boldsymbol{0}_{3\times 1}$.
- **Vector or pure quaternion**: $\boldsymbol{q}$ is a pure quaternion if its real part is null, $q_r = 0$.

B.2. Quaternions algebra

B.2.1. Addition of quaternions

Being $\boldsymbol{p}$ and $\boldsymbol{q}$ two quaternions, $\boldsymbol{p}, \boldsymbol{q} \in \mathbb{H}$, the quaternion addition, $\boldsymbol{p} + \boldsymbol{q}$ is defined as:

$$\boldsymbol{p} + \boldsymbol{q} = \begin{bmatrix} p_w \\ p_x \\ p_y \\ p_z \end{bmatrix} + \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} p_w + q_w \\ p_x + q_x \\ p_y + q_y \\ p_z + q_z \end{bmatrix} = \begin{bmatrix} p_w + q_w \\ \boldsymbol{p_v} + \boldsymbol{q_v} \end{bmatrix} \tag{B.8}$$

Properties

Being $\boldsymbol{p}$, $\boldsymbol{q}$, and $\boldsymbol{r}$ quaternions, $\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r} \in \mathbb{H}$:

- Associativity:

$$\boldsymbol{p} + \boldsymbol{q} + \boldsymbol{r} = (\boldsymbol{p} + \boldsymbol{q}) + \boldsymbol{r} = \boldsymbol{p} + (\boldsymbol{q} + \boldsymbol{r}) \tag{B.9}$$

- Commutativity:

$$\boldsymbol{p} + \boldsymbol{q} = \boldsymbol{q} + \boldsymbol{p} \tag{B.10}$$

- Scalar: Being α a scalar number, $\alpha \in \mathbb{R}$,

$$\alpha \cdot (\boldsymbol{p} + \boldsymbol{q}) = \alpha \cdot \boldsymbol{p} + \alpha \cdot \boldsymbol{q} \tag{B.11}$$

Identity quaternion with respect to the addition

The identity quaternion $\boldsymbol{q_I} \in \mathbb{H}$ with respect to the addition must meet for every quaternion $\boldsymbol{q} \in \mathbb{H}$ that:

$$\boldsymbol{q_I} + \boldsymbol{q} = \boldsymbol{q} + \boldsymbol{q_I} = \boldsymbol{q} \tag{B.12}$$

being therefore:

$$\boldsymbol{q_I} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{0}_{3\times 1} \end{bmatrix} \tag{B.13}$$

B.2.2. Multiplication of quaternions or quaternion product

Being $\boldsymbol{p}$ and $\boldsymbol{q}$ two quaternions, $\boldsymbol{p},\boldsymbol{q} \in \mathbb{H}$, the quaternion product, $\boldsymbol{p} \otimes \boldsymbol{q}$, is defined as:

$$\begin{aligned}
\boldsymbol{p} \otimes \boldsymbol{q} &= \begin{bmatrix} p_w \\ \boldsymbol{p_v} \end{bmatrix} \otimes \begin{bmatrix} q_w \\ \boldsymbol{q_v} \end{bmatrix} \\
&= (p_w + ip_x + jp_y + kp_z) \cdot (q_w + iq_x + jq_y + kq_z) \\
&= (p_w q_w - p_x q_x - p_y q_y - p_z q_z) + i(p_w q_x + p_x q_w + p_y q_z - p_z q_y) \\
&\quad + j(p_w q_y + p_y q_w + p_z q_x - p_x q_z) + k(p_w q_z + p_z q_w + p_x q_y - p_y q_x) \\
&= \begin{bmatrix} p_w q_w - p_x q_x - p_y q_y - p_z q_z \\ p_w q_x + p_x q_w + p_y q_z - p_z q_y \\ p_w q_y - p_x q_z + p_y q_w + p_z q_x \\ p_w q_z + p_x q_y - p_y q_x + p_z q_w \end{bmatrix} \\
&= \begin{bmatrix} p_w q_w - \boldsymbol{p_v}^T \boldsymbol{q_v} \\ p_w \boldsymbol{q_v} + q_w \boldsymbol{p_v} + \boldsymbol{p_v} \times \boldsymbol{q_v} \end{bmatrix}
\end{aligned} \tag{B.14}$$

Properties

Being $\boldsymbol{p}$, $\boldsymbol{q}$, and $\boldsymbol{r}$ quaternions, $\boldsymbol{p},\boldsymbol{q},\boldsymbol{r} \in \mathbb{H}$:

- Associativity:

$$\boldsymbol{p} \otimes \boldsymbol{q} \otimes \boldsymbol{r} = (\boldsymbol{p} \otimes \boldsymbol{q}) \otimes \boldsymbol{r} = \boldsymbol{p} \otimes (\boldsymbol{q} \otimes \boldsymbol{r}) \tag{B.15}$$

- Distributivity over addition:

$$(\boldsymbol{p} + \boldsymbol{q}) \otimes \boldsymbol{r} = \boldsymbol{p} \otimes \boldsymbol{r} + \boldsymbol{q} \otimes \boldsymbol{r} \tag{B.16}$$

$$\boldsymbol{p} \otimes (\boldsymbol{q} + \boldsymbol{r}) = \boldsymbol{p} \otimes \boldsymbol{q} + \boldsymbol{p} \otimes \boldsymbol{r} \tag{B.17}$$

- No commutativity: The quaternion product is not commutative:

$$\boldsymbol{p} \otimes \boldsymbol{q} \neq \boldsymbol{q} \otimes \boldsymbol{p} \tag{B.18}$$

- Scalar: Being α a scalar number, $\alpha \in \mathbb{R}$,

$$\alpha \cdot (\boldsymbol{p} \otimes \boldsymbol{q}) = (\alpha \cdot \boldsymbol{p}) \otimes \boldsymbol{q} = \boldsymbol{p} \otimes (\alpha \cdot \boldsymbol{q}) \tag{B.19}$$

Identity quaternion with respect to the multiplication

The identity quaternion $\boldsymbol{q_I} \in \mathbb{H}$ with respect to the multiplication must meet for every quaternion $\boldsymbol{q} \in \mathbb{H}$ that:

$$\boldsymbol{q_I} \otimes \boldsymbol{q} = \boldsymbol{q} \otimes \boldsymbol{q_I} = \boldsymbol{q} \tag{B.20}$$

being therefore:

$$\boldsymbol{q_I} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ \boldsymbol{0}_{3\times 1} \end{bmatrix} \tag{B.21}$$

Special cases

Being $\boldsymbol{p}$ and $\boldsymbol{q}$ quaternions, $\boldsymbol{p},\boldsymbol{q} \in \mathbb{H}$:

- Product of parallel quaternions: if $\boldsymbol{p}$ and $\boldsymbol{q}$ are parallel:

$$\boldsymbol{p} \otimes \boldsymbol{q} = \begin{bmatrix} -\boldsymbol{p_v}^T \boldsymbol{q_v} \\ \boldsymbol{0}_{3\times 1} \end{bmatrix} \tag{B.22}$$

- Product of perpendicular quaternions: if $\boldsymbol{p}$ and $\boldsymbol{q}$ are perpendicular:

$$\boldsymbol{p} \otimes \boldsymbol{q} = \begin{bmatrix} 0 \\ \boldsymbol{p_v} \times \boldsymbol{q_v} \end{bmatrix} \tag{B.23}$$

- Product of pure quaternions: if $\boldsymbol{p}$ and $\boldsymbol{q}$ are two pure quaternions, where $p_w = q_w = 0$, then:

$$\boldsymbol{p} \otimes \boldsymbol{q} = \begin{bmatrix} -\boldsymbol{p_v}^T \boldsymbol{q_v} \\ \hline \boldsymbol{p_v} \times \boldsymbol{q_v} \end{bmatrix} \tag{B.24}$$

Which implies:

$$\boldsymbol{p} \otimes \boldsymbol{q} - \boldsymbol{q} \otimes \boldsymbol{p} = \begin{bmatrix} 0 \\ \hline 2\boldsymbol{p_v} \times \boldsymbol{q_v} \end{bmatrix} \tag{B.25}$$

B.2.3. Quaternion product as a matrix multiplication

The product of two quaternions, $\boldsymbol{p} \otimes \boldsymbol{q}$, is a bi-linear operation and can be expressed as two equivalent matrix products:

$$\boldsymbol{p} \otimes \boldsymbol{q} = \boldsymbol{Q}_+(\boldsymbol{p}) \cdot \boldsymbol{q} = \boldsymbol{Q}_-(\boldsymbol{q}) \cdot \boldsymbol{p} \tag{B.26}$$

Being:

$$\begin{aligned} \boldsymbol{Q}_+(\boldsymbol{p}) &= \begin{bmatrix} p_w & -p_x & -p_y & -p_z \\ p_x & p_w & -p_z & p_y \\ p_y & p_z & p_w & -p_x \\ p_z & -p_y & p_x & p_w \end{bmatrix} \\ &= p_w \cdot \boldsymbol{I}_{4\times4} + \begin{bmatrix} 0 & -\boldsymbol{p_v}^T \\ \boldsymbol{p_v} & [\boldsymbol{p_v}]_\times \end{bmatrix} \\ &= \begin{bmatrix} p_w & -\boldsymbol{p_v}^T \\ \boldsymbol{p_v} & p_w \cdot \boldsymbol{I}_{3\times3} + [\boldsymbol{p_v}]_\times \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{p} & \boldsymbol{Q}_{s+}(\boldsymbol{p}) \end{bmatrix} \end{aligned} \tag{B.27}$$

$$\begin{aligned} \boldsymbol{Q}_-(\boldsymbol{q}) &= \begin{bmatrix} q_w & -q_x & -q_y & -q_z \\ q_x & q_w & q_z & -q_y \\ q_y & -q_z & q_w & q_x \\ q_z & q_y & -q_x & q_w \end{bmatrix} \\ &= q_w \cdot \boldsymbol{I}_{4\times4} + \begin{bmatrix} 0 & -\boldsymbol{q_v}^T \\ \boldsymbol{q_v} & -[\boldsymbol{q_v}]_\times \end{bmatrix} \\ &= \begin{bmatrix} q_w & -\boldsymbol{q_v}^T \\ \boldsymbol{q_v} & q_w \cdot \boldsymbol{I}_{3\times3} - [\boldsymbol{q_v}]_\times \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{q} & \boldsymbol{Q}_{s-}(\boldsymbol{q}) \end{bmatrix} \end{aligned} \tag{B.28}$$

Where $[\bullet]_\times$ represents the skew-symmetric matrix (see Appendix B.2.14).

Properties

Relationship with inverse matrix

Being $\boldsymbol{q}$ a quaternion, $\boldsymbol{q} \in \mathbb{H}$:

$$\boldsymbol{Q}_+(\boldsymbol{q}) \cdot \boldsymbol{Q}_+(\boldsymbol{q})^{-1} = \boldsymbol{Q}_+(\boldsymbol{q})^{-1} \cdot \boldsymbol{Q}_+(\boldsymbol{q}) = \boldsymbol{I}_{4\times4} \tag{B.29}$$

$$\boldsymbol{Q}_-(\boldsymbol{q}) \cdot \boldsymbol{Q}_-(\boldsymbol{q})^{-1} = \boldsymbol{Q}_-(\boldsymbol{q})^{-1} \cdot \boldsymbol{Q}_-(\boldsymbol{q}) = \boldsymbol{I}_{4\times4} \tag{B.30}$$

Relationship with inverse quaternion

Being $\boldsymbol{q}$ a quaternion, $\boldsymbol{q} \in \mathbb{H}$, using the definition of inverse quaternion $\boldsymbol{q} \otimes \boldsymbol{q}^{-1} = \boldsymbol{q}^{-1} \otimes \boldsymbol{q} = \boldsymbol{q}_I$:

$$\boldsymbol{Q}_+(\boldsymbol{q}) \cdot \boldsymbol{Q}_+(\boldsymbol{q}^{-1}) = \boldsymbol{I}_{4\times4} \tag{B.31}$$

$$\boldsymbol{Q}_+(\boldsymbol{q}^{-1}) \cdot \boldsymbol{Q}_+(\boldsymbol{q}) = \boldsymbol{I}_{4\times4} \tag{B.32}$$

$$\boldsymbol{Q}_-(\boldsymbol{q}) \cdot \boldsymbol{Q}_-(\boldsymbol{q}^{-1}) = \boldsymbol{I}_{4\times4} \tag{B.33}$$

$$\boldsymbol{Q}_-(\boldsymbol{q}^{-1}) \cdot \boldsymbol{Q}_-(\boldsymbol{q}) = \boldsymbol{I}_{4\times4} \tag{B.34}$$

And therefore:

$$\boldsymbol{Q}_+(\boldsymbol{q}^{-1}) = \boldsymbol{Q}_+(\boldsymbol{q})^{-1} \tag{B.35}$$
$$\boldsymbol{Q}_+(\boldsymbol{q}^{-1})^{-1} = \boldsymbol{Q}_+(\boldsymbol{q}) \tag{B.36}$$
$$\boldsymbol{Q}_-(\boldsymbol{q}^{-1}) = \boldsymbol{Q}_-(\boldsymbol{q})^{-1} \tag{B.37}$$
$$\boldsymbol{Q}_-(\boldsymbol{q}^{-1})^{-1} = \boldsymbol{Q}_-(\boldsymbol{q}) \tag{B.38}$$

Multiple Products

Being $\boldsymbol{q}$, $\boldsymbol{r}$ and $\boldsymbol{p}$ quaternions, $\boldsymbol{q},\boldsymbol{r},\boldsymbol{p} \in \mathbb{H}$, using the associativity property of the quaternion product:

$$\begin{aligned}\boldsymbol{q}\otimes\boldsymbol{r}\otimes\boldsymbol{p} &= (\boldsymbol{q}\otimes\boldsymbol{r})\otimes\boldsymbol{p} = \boldsymbol{Q}_-(\boldsymbol{p})\cdot\boldsymbol{Q}_+(\boldsymbol{q})\cdot\boldsymbol{r} \\ &= \boldsymbol{q}\otimes(\boldsymbol{r}\otimes\boldsymbol{p}) = \boldsymbol{Q}_+(\boldsymbol{q})\cdot\boldsymbol{Q}_-(\boldsymbol{p})\cdot\boldsymbol{r}\end{aligned} \tag{B.39}$$

And then:

$$\boldsymbol{Q}_-(\boldsymbol{p})\cdot\boldsymbol{Q}_+(\boldsymbol{q}) = \boldsymbol{Q}_+(\boldsymbol{q})\cdot\boldsymbol{Q}_-(\boldsymbol{p}) \tag{B.40}$$

Pure quaternions

Being $\boldsymbol{q}$ a pure quaternion, $\boldsymbol{q} \in \mathbb{H}$:

$$\boldsymbol{Q}_+(\boldsymbol{q}) = \begin{bmatrix} 0 & -\boldsymbol{q}_v^T \\ \boldsymbol{q}_v & [\boldsymbol{q}_v]_\times \end{bmatrix} \tag{B.41}$$
$$\boldsymbol{Q}_-(\boldsymbol{q}) = \begin{bmatrix} 0 & -\boldsymbol{q}_v^T \\ \boldsymbol{q}_v & -[\boldsymbol{q}_v]_\times \end{bmatrix} \tag{B.42}$$

Submatrices

Two submatrices of the quaternion product are defined as:

$$\boldsymbol{Q}_{s+}(\boldsymbol{q}) = \begin{bmatrix} -\boldsymbol{q}_v^T \\ q_w\cdot\boldsymbol{I}_{3\times3} + [\boldsymbol{q}_v]_\times \end{bmatrix} \tag{B.43}$$
$$\boldsymbol{Q}_{s-}(\boldsymbol{q}) = \begin{bmatrix} -\boldsymbol{q}_v^T \\ q_w\cdot\boldsymbol{I}_{3\times3} - [\boldsymbol{q}_v]_\times \end{bmatrix} \tag{B.44}$$

with dimension 4×3.

Properties

The submatrices have the following properties:

$$\boldsymbol{Q}_{s+}(\boldsymbol{q})^T\cdot\boldsymbol{Q}_{s+}(\boldsymbol{q}) = \|\boldsymbol{q}\|^2\cdot\boldsymbol{I}_{3\times3} \tag{B.45}$$
$$\boldsymbol{Q}_{s+}(\boldsymbol{q})\cdot\boldsymbol{Q}_{s+}(\boldsymbol{q})^T = \|\boldsymbol{q}\|^2\cdot\boldsymbol{I}_{4\times4} - \boldsymbol{q}\cdot\boldsymbol{q}^T \tag{B.46}$$
$$\boldsymbol{Q}_{s+}(\boldsymbol{q})^T\cdot\boldsymbol{q} = \boldsymbol{0}_{3\times1} \tag{B.47}$$
$$\boldsymbol{Q}_{s-}(\boldsymbol{q})^T\cdot\boldsymbol{Q}_{s-}(\boldsymbol{q}) = \|\boldsymbol{q}\|^2\cdot\boldsymbol{I}_{3\times3} \tag{B.48}$$
$$\boldsymbol{Q}_{s-}(\boldsymbol{q})\cdot\boldsymbol{Q}_{s-}(\boldsymbol{q})^T = \|\boldsymbol{q}\|^2\cdot\boldsymbol{I}_{4\times4} - \boldsymbol{q}\cdot\boldsymbol{q}^T \tag{B.49}$$
$$\boldsymbol{Q}_{s-}(\boldsymbol{q})^T\cdot\boldsymbol{q} = \boldsymbol{0}_{3\times1} \tag{B.50}$$

B.2.4. Derivative of the quaternion product

Having a variable $x \in \mathbb{R}$ and two quaternions $\boldsymbol{p} = \boldsymbol{p}(x)$ and $\boldsymbol{q} = \boldsymbol{q}(x)$, the derivative of the quaternion product is:

$$\left[\frac{\partial(\boldsymbol{p}\otimes\boldsymbol{q})}{\partial x}\right]_{4\times1} = \boldsymbol{Q}_+(\boldsymbol{p})\cdot\frac{\partial\boldsymbol{q}}{\partial x} + \boldsymbol{Q}_-(\boldsymbol{q})\cdot\frac{\partial\boldsymbol{p}}{\partial x} \tag{B.51}$$

Having a variable $\boldsymbol{r} \in \mathbb{R}^n$ and two quaternions $\boldsymbol{p} = \boldsymbol{p}(\boldsymbol{r})$ and $\boldsymbol{q} = \boldsymbol{q}(\boldsymbol{r})$, the derivative of the quaternion product is:

$$\left[\frac{\partial(\boldsymbol{p}\otimes\boldsymbol{q})}{\partial \boldsymbol{r}}\right]_{4\times n_{\boldsymbol{r}}} = \boldsymbol{Q}_+(\boldsymbol{p})\cdot\frac{\partial\boldsymbol{q}}{\partial\boldsymbol{r}} + \boldsymbol{Q}_-(\boldsymbol{q})\cdot\frac{\partial\boldsymbol{p}}{\partial\boldsymbol{r}} \tag{B.52}$$

B.2.5. Jacobian matrices of the quaternion product

The Jacobian matrices of the product of two quaternions $\boldsymbol{p}$ and $\boldsymbol{q}$ can be calculated as:

$$\left[\frac{\partial(\boldsymbol{p}\otimes\boldsymbol{q})}{\partial\boldsymbol{p}}\right]_{4\times4} = \boldsymbol{Q}_{-}(\boldsymbol{q}) \tag{B.53}$$

$$\left[\frac{\partial(\boldsymbol{p}\otimes\boldsymbol{q})}{\partial\boldsymbol{q}}\right]_{4\times4} = \boldsymbol{Q}_{+}(\boldsymbol{p}) \tag{B.54}$$

B.2.6. Conjugate quaternion

The conjugate quaternion $\boldsymbol{q}^*$ of a quaternion $\boldsymbol{q}\in\mathbb{H}$ is defined as:

$$\boldsymbol{q}^* = q_w - \boldsymbol{q_v} = q_w - (q_x i + q_y j + q_z k) = \left[\begin{array}{c} q_w \\ \hdashline -\boldsymbol{q_v} \end{array}\right]. \tag{B.55}$$

Properties

Being $\boldsymbol{q}$ and $\boldsymbol{p}$ quaternions, $\boldsymbol{q},\boldsymbol{p}\in\mathbb{H}$:

- Associativity with respect to the multiplication:

$$(\boldsymbol{p}\otimes\boldsymbol{q})^* = \boldsymbol{q}^*\otimes\boldsymbol{p}^* \tag{B.56}$$

- Associativity with respect to the addition:

$$(\boldsymbol{p}+\boldsymbol{q})^* = \boldsymbol{p}^*+\boldsymbol{q}^* \tag{B.57}$$

- Scalar: Being α a scalar number, $\alpha\in\mathbb{R}$,

$$(\alpha\cdot\boldsymbol{q})^* = \alpha\cdot(\boldsymbol{q})^* \tag{B.58}$$

Jacobian matrix of the conjugate quaternion

$$\left[\frac{\partial(\boldsymbol{q}^*)}{\partial\boldsymbol{q}}\right]_{4\times4} = \begin{bmatrix} 1 & \boldsymbol{0}_{1\times3} \\ \boldsymbol{0}_{3\times1} & -\boldsymbol{I}_{3\times3} \end{bmatrix} \tag{B.59}$$

B.2.7. Norm of a quaternion

The norm of a quaternion $\boldsymbol{q}\in\mathbb{H}$ is defined as:

$$\|\boldsymbol{q}\|^2 = \boldsymbol{q}\otimes\boldsymbol{q}^* = \boldsymbol{q}^*\otimes\boldsymbol{q} = q_w^2+q_x^2+q_y^2+q_z^2 = q_w^2+\boldsymbol{q_v}^T\cdot\boldsymbol{q_v} \tag{B.60}$$

Jacobian matrix of the norm of a quaternion

$$\begin{aligned}\left[\frac{\partial(\|\boldsymbol{q}\|)}{\partial\boldsymbol{q}}\right]_{1\times4} &= \frac{\partial}{\partial\boldsymbol{q}}(\sqrt{\boldsymbol{q}\otimes\boldsymbol{q}^*}) \\ &= \frac{1}{\|\boldsymbol{q}\|}\cdot\boldsymbol{q}^T\end{aligned} \tag{B.61}$$

$$\left[\frac{\partial(\|\boldsymbol{q}\|^2)}{\partial\boldsymbol{q}}\right]_{1\times4} = 2\cdot\boldsymbol{q}^T \tag{B.62}$$

B.2.8. Inverse quaternion

The inverse quaternion $\boldsymbol{q}^{-1}$ of a quaternion $\boldsymbol{q} \in \mathbb{H}$ is defined such that:

$$\boldsymbol{q} \otimes \boldsymbol{q}^{-1} = \boldsymbol{q}^{-1} \otimes \boldsymbol{q} = \boldsymbol{q}_I \tag{B.63}$$

And can be computed with:

$$\boldsymbol{q}^{-1} = \frac{\boldsymbol{q}^*}{\|\boldsymbol{q}\|^2} \tag{B.64}$$

Properties

Being $\boldsymbol{q}$ and $\boldsymbol{p}$ quaternions, $\boldsymbol{q}, \boldsymbol{p} \in \mathbb{H}$:

- Associativity with respect to the multiplication:

$$(\boldsymbol{p} \otimes \boldsymbol{q})^{-1} = \boldsymbol{q}^{-1} \otimes \boldsymbol{p}^{-1} \tag{B.65}$$

Jacobian matrix of the inverse quaternion

$$\left[\frac{\partial(\boldsymbol{q}^{-1})}{\partial \boldsymbol{q}}\right]_{4\times 4} = \frac{1}{\|\boldsymbol{q}\|^4} \cdot \left(\|\boldsymbol{q}\|^2 \cdot \left[\frac{\partial(\boldsymbol{q}^*)}{\partial \boldsymbol{q}}\right] - 2 \cdot \boldsymbol{q}^* \cdot \boldsymbol{q}^T\right) \tag{B.66}$$

B.2.9. Unit or normalized quaternion

A unit quaternion $\boldsymbol{q} \in \mathbb{H}$ is a quaternion, whose norm is unitary:

$$\|\boldsymbol{q}\| = 1 \tag{B.67}$$

Inverse of unit quaternion

The inverse of a unit quaternion $\boldsymbol{q} \in \mathbb{H}$ is simplified to:

$$\boldsymbol{q}^{-1} = \boldsymbol{q}^* \tag{B.68}$$

Unit quaternions to define rotations

As it is discussed in Appendix B.3, unit quaternions can be interpreted as an orientation specification, or as a rotation operator.

A unit quaternion $\boldsymbol{q}$ can always be written in the form:

$$\boldsymbol{q} = \left[\begin{array}{c} \cos\theta \\ \hdashline \boldsymbol{u} \cdot \sin\theta \end{array}\right] \tag{B.69}$$

where $\boldsymbol{u} = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix}^T$ is a unit vector and θ is a scalar.

Jacobian matrix of the inverse quaternion of a unit quaternion

Being $\boldsymbol{q}$ a unit quaternion:

$$\left[\frac{\partial(\boldsymbol{q}^{-1})}{\partial \boldsymbol{q}}\right]_{4\times 4} = \left[\frac{\partial(\boldsymbol{q}^*)}{\partial \boldsymbol{q}}\right] \tag{B.70}$$

Quaternion product as a matrix multiplication

Relationship with the inverse matrix

Being $\boldsymbol{q}$ a unit quaternion, the matrices are simplified to:

$$\boldsymbol{Q}_-(\boldsymbol{q})^T = \boldsymbol{Q}_-(\boldsymbol{q})^{-1} \tag{B.71}$$

$$\boldsymbol{Q}_+(\boldsymbol{q})^T = \boldsymbol{Q}_+(\boldsymbol{q})^{-1} \tag{B.72}$$

Relationship with the inverse and conjugate quaternion

Being $\boldsymbol{q}$ a unit quaternion, the matrices are simplified to:

$$\boldsymbol{Q}_-(\boldsymbol{q}^{-1}) = \boldsymbol{Q}_-(\boldsymbol{q}^*) = \boldsymbol{Q}_-(\boldsymbol{q})^T \tag{B.73}$$

$$\boldsymbol{Q}_+(\boldsymbol{q}^{-1}) = \boldsymbol{Q}_+(\boldsymbol{q}^*) = \boldsymbol{Q}_+(\boldsymbol{q})^T \tag{B.74}$$

Relationship with transpose

Being $\boldsymbol{q}$ a unit quaternion, the matrices are simplified to:

$$\boldsymbol{Q}_-(\boldsymbol{q}) \cdot \boldsymbol{Q}_-(\boldsymbol{q})^T = \boldsymbol{Q}_-(\boldsymbol{q})^T \cdot \boldsymbol{Q}_-(\boldsymbol{q}) = \boldsymbol{I}_{4\times 4} \tag{B.75}$$

$$\boldsymbol{Q}_+(\boldsymbol{q}) \cdot \boldsymbol{Q}_+(\boldsymbol{q})^T = \boldsymbol{Q}_+(\boldsymbol{q})^T \cdot \boldsymbol{Q}_+(\boldsymbol{q}) = \boldsymbol{I}_{4\times 4} \tag{B.76}$$

Multiple products

Being $\boldsymbol{q}$ and $\boldsymbol{p}$ are unit quaternions, the matrices are simplified:

$$\boldsymbol{Q}_+(\boldsymbol{q}) \cdot \boldsymbol{Q}_-(\boldsymbol{p})^T = \boldsymbol{Q}_-(\boldsymbol{p})^T \cdot \boldsymbol{Q}_+(\boldsymbol{q}) \tag{B.77}$$

B.2.10. Powers of quaternions

The n-th power of the quaternion $\boldsymbol{q}$ using the quaternion product $\otimes$ is given by $\boldsymbol{q}^n$.

Powers of pure quaternions

If $\boldsymbol{q} = [0, \boldsymbol{q_v}]^T$ is a pure quaternion (with $q_w = 0$), then $\boldsymbol{q} = \boldsymbol{q_v} = \boldsymbol{u} \cdot \theta$, with $\theta = \|\boldsymbol{q}\| = \|\boldsymbol{q_v}\| \in \mathbb{R}$ and therefore $\boldsymbol{u} = \frac{\boldsymbol{q_v}}{\|\boldsymbol{q_v}\|}$ unitary.

The powers of a pure quaternion are given therefore by:

$$\boldsymbol{q}^2 = -\theta^2 \tag{B.78}$$

$$\boldsymbol{q}^3 = -\boldsymbol{u}\theta^3 \tag{B.79}$$

$$\boldsymbol{q}^4 = \theta^4 \tag{B.80}$$

$$\boldsymbol{q}^5 = \boldsymbol{u}\theta^5 \tag{B.81}$$

$$\boldsymbol{q}^6 = -\theta^6 \tag{B.82}$$

$$\ldots$$

B.2.11. Exponential of quaternions

The exponential of the quaternion $\boldsymbol{q}$ is defined as $e^{\boldsymbol{q}}$.

Exponential of pure quaternions

The exponential of a pure quaternion $\boldsymbol{q} = [0, \boldsymbol{q_v}]^T = \boldsymbol{q_v}$ is a new quaternion defined by the absolutely convergent series:

$$e^{\boldsymbol{q_v}} = \sum_{k=0}^{\infty} \frac{\boldsymbol{q_v}^n}{k!} \quad \in \mathbb{H} \tag{B.83}$$

Being $\boldsymbol{q_v} = \boldsymbol{u} \cdot \theta$, with $\theta = \|\boldsymbol{q_v}\| \in \mathbb{R}$ and therefore $\boldsymbol{u} = \frac{\boldsymbol{q_v}}{\|\boldsymbol{q_v}\|}$ unitary. Considering the equations of the powers of pure quaternions, the scalar and vector terms in the series can be grouped and recognized in them, respectively, the series of $\cos\theta$ and $\sin\theta$ [1], resulting:

$$\begin{aligned} e^{\boldsymbol{q_v}} &= e^{\boldsymbol{u}\cdot\theta} = \cos\theta + \boldsymbol{u}\cdot\sin\theta = \left[\begin{array}{c} \cos\theta \\ \hdashline \boldsymbol{u}\cdot\sin\theta \end{array}\right] \\ &= \cos\|\boldsymbol{q_v}\| + \frac{\boldsymbol{q_v}}{\|\boldsymbol{q_v}\|}\cdot\sin\|\boldsymbol{q_v}\| = \left[\begin{array}{c} \cos\|\boldsymbol{q_v}\| \\ \hdashline \frac{\boldsymbol{q_v}}{\|\boldsymbol{q_v}\|}\cdot\sin\|\boldsymbol{q_v}\| \end{array}\right] \end{aligned} \tag{B.84}$$

which constitutes an extension of the Euler formula $e^{i\theta} = \cos\theta + i\cdot\sin\theta$

Notice that since $\|e^{\boldsymbol{q_v}}\|^2 = \cos^2\theta + \sin^2\theta = 1$, the exponential of a pure quaternion is a unit quaternion.

[1] Note that: $\cos\theta = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \ldots$ and $\sin\theta = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \ldots$

Exponential of general quaternions

Given a general quaternion $\boldsymbol{q} = [q_w, \boldsymbol{q_v}]^T = q_w + \boldsymbol{q_v}$:

$$e^{\boldsymbol{q}} = e^{q_w + \boldsymbol{q_v}} = e^{q_w + \boldsymbol{q_v}} = e^{q_w} \cdot e^{\boldsymbol{q_v}} \tag{B.85}$$

Then, using equation B.84, can be obtained:

$$e^{\boldsymbol{q}} = e^{q_w} \cdot (\cos\|\boldsymbol{q_v}\| + \frac{\boldsymbol{q_v}}{\|\boldsymbol{q_v}\|} \cdot \sin\|\boldsymbol{q_v}\|) = e^{q_w} \cdot \begin{bmatrix} \cos\|\boldsymbol{q_v}\| \\ \frac{\boldsymbol{q_v}}{\|\boldsymbol{q_v}\|} \cdot \sin\|\boldsymbol{q_v}\| \end{bmatrix} \tag{B.86}$$

B.2.12. Logarithm of quaternions

The logarithm of the quaternion $\boldsymbol{q}$ is defined as $\log \boldsymbol{q} = \boldsymbol{p} \Leftrightarrow \boldsymbol{q} = e^{\boldsymbol{p}}$.

Logarithm of unit quaternions

In the case that $\boldsymbol{q}$ is a unit quaternion:

$$\log \boldsymbol{q} = \log(\cos\theta + \boldsymbol{u} \cdot \sin\theta) = \log(e^{\boldsymbol{u}\theta}) = \boldsymbol{u}\theta = \begin{bmatrix} 0 \\ \boldsymbol{u}\theta \end{bmatrix} \tag{B.87}$$

that is, the logarithm of a unit quaternion is a pure quaternion.

Logarithm of general quaternions

By extension of the previous equation, if $\boldsymbol{q}$ is a general quaternion:

$$\begin{aligned} \log \boldsymbol{q} &= \log(\|\boldsymbol{q}\| \frac{\boldsymbol{q}}{\|\boldsymbol{q}\|}) = \log\|\boldsymbol{q}\| + \log\frac{\boldsymbol{q}}{\|\boldsymbol{q}\|} \\ &= \log\|\boldsymbol{q}\| + \boldsymbol{u}\theta = \begin{bmatrix} \log\|\boldsymbol{q}\| \\ \boldsymbol{u}\theta \end{bmatrix} \end{aligned} \tag{B.88}$$

B.2.13. Expanded vectors: vectors in quaternion notation

To be able to operate with a vector $\boldsymbol{a} \in \mathbb{R}^3$ in quaternion notation, it is needed to define the expanded vector $\bar{\boldsymbol{a}}$ as:

$$\bar{\boldsymbol{a}} = \begin{bmatrix} 0 \\ \boldsymbol{a} \end{bmatrix} \in \mathbb{H} \tag{B.89}$$

Jacobian matrices of expanded vectors

Two Jacobian matrices that relates the vector $\boldsymbol{a} \in \mathbb{R}^3$ and $\bar{\boldsymbol{a}} \in \mathbb{H}$ are defined:

$$\left[\frac{\partial \bar{\boldsymbol{a}}}{\partial \boldsymbol{a}}\right]_{4\times 3} = \begin{bmatrix} \boldsymbol{0}_{1\times 3} \\ \boldsymbol{I}_{3\times 3} \end{bmatrix} \tag{B.90}$$

$$\left[\frac{\partial \boldsymbol{a}}{\partial \bar{\boldsymbol{a}}}\right]_{3\times 4} = \begin{bmatrix} \boldsymbol{0}_{3\times 1} & \boldsymbol{I}_{3\times 3} \end{bmatrix} \tag{B.91}$$

B.2.14. Cross product and skew-symmetric matrix

Although the cross product and the skew-symmetric matrix are not part of the quaternion algebra, they are included here, due to their intensive usage in the quaternion algebra.

The skew operator $[\bullet]_\times$ produces the cross-product matrix of a vector $\boldsymbol{a} \in \mathbb{R}^3$:

$$[\boldsymbol{a}]_\times = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \tag{B.92}$$

which is a skew-symmetric matrix, $[\boldsymbol{a}]_\times^T = -[\boldsymbol{a}]_\times$, left-hand-equivalent to the cross product:

$$\boldsymbol{a} \times \boldsymbol{b} = [\boldsymbol{a}]_\times \cdot \boldsymbol{b}\,, \quad \forall \boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^3 \tag{B.93}$$

Properties of the cross product and skew-symmetric matrix

Being $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ three vectors, $\boldsymbol{a},\boldsymbol{b},\boldsymbol{c} \in \mathbb{R}^3$, the cross product and the skew-symmetric matrix follow the following properties:

- Anti-commutativity:

$$[\boldsymbol{a}]_\times = -[\boldsymbol{a}]_\times^T \tag{B.94}$$

Also:

$$\begin{aligned} \boldsymbol{a}\times\boldsymbol{b} &= -\boldsymbol{b}\times\boldsymbol{a} \\ [\boldsymbol{a}]_\times\cdot\boldsymbol{b} &= -[\boldsymbol{b}]_\times\cdot\boldsymbol{a} \end{aligned} \tag{B.95}$$

Getting transposes and reordering:

$$\begin{aligned} (\boldsymbol{a}\times\boldsymbol{b})^T &= -(\boldsymbol{b}\times\boldsymbol{a})^T \\ \boldsymbol{b}^T\cdot[\boldsymbol{a}]_\times^T &= -\boldsymbol{a}^T\cdot[\boldsymbol{b}]_\times^T \\ -\boldsymbol{b}^T\cdot[\boldsymbol{a}]_\times &= \boldsymbol{a}^T\cdot[\boldsymbol{b}]_\times \end{aligned} \tag{B.96}$$

- Distributivity over addition:

$$[\boldsymbol{a}]_\times + [\boldsymbol{b}]_\times = [\boldsymbol{a}+\boldsymbol{b}]_\times \tag{B.97}$$

- Scalar multiplication: being $\alpha \in \mathbb{R}$ a scalar number:

$$\alpha\cdot[\boldsymbol{a}]_\times = [\alpha\cdot\boldsymbol{a}]_\times \tag{B.98}$$

- Cross product of parallel vectors: if $\boldsymbol{a}$ and $\boldsymbol{b}$ two parallel vector, then $\boldsymbol{b} = \alpha\cdot\boldsymbol{a}$, being $\alpha \in \mathbb{R}$ a scalar number. Then, the cross product of two parallel vectors is:

$$\boldsymbol{a}\times\boldsymbol{b} = \boldsymbol{a}\times(\alpha\cdot\boldsymbol{a}) = \alpha\cdot(\boldsymbol{a}\times\boldsymbol{a}) = \boldsymbol{0}_{3\times1} \tag{B.99}$$

And therefore:

$$[\boldsymbol{a}]_\times\cdot\boldsymbol{a} = -(\boldsymbol{a}^T\cdot[\boldsymbol{a}]_\times)^T = \boldsymbol{0}_{3\times1} \tag{B.100}$$

- Lagrange's Formula:

$$\begin{aligned} &\boldsymbol{a}\times(\boldsymbol{b}\times\boldsymbol{c}) = \boldsymbol{b}\cdot(\boldsymbol{a}^T\cdot\boldsymbol{c}) - \boldsymbol{c}\cdot(\boldsymbol{a}^T\cdot\boldsymbol{b}) \\ &\Leftrightarrow [\boldsymbol{a}]_\times[\boldsymbol{b}]_\times = \boldsymbol{b}\cdot\boldsymbol{a}^T - (\boldsymbol{a}^T\cdot\boldsymbol{b})\cdot\boldsymbol{I}_{3\times3} \end{aligned} \tag{B.101}$$

With the following consequences:

$$[\boldsymbol{a}]_\times[\boldsymbol{b}]_\times + \boldsymbol{a}\cdot\boldsymbol{b}^T = [\boldsymbol{b}]_\times[\boldsymbol{a}]_\times + \boldsymbol{b}\cdot\boldsymbol{a}^T \tag{B.102}$$

$$[\boldsymbol{a}\times\boldsymbol{b}]_\times = \boldsymbol{b}\cdot\boldsymbol{a}^T - \boldsymbol{a}\cdot\boldsymbol{b}^T \tag{B.103}$$

- Jacobi Identity:

$$\boldsymbol{a}\times\boldsymbol{b}\times\boldsymbol{c} + \boldsymbol{b}\times\boldsymbol{c}\times\boldsymbol{a} + \boldsymbol{c}\times\boldsymbol{a}\times\boldsymbol{b} = \boldsymbol{0}_{3\times1} \tag{B.104}$$

- Skew-symmetric matrix and rotations: being $\boldsymbol{C}$ a rotation matrix, $\boldsymbol{C} \in \mathbb{R}^{3\times3}$:

$$[\boldsymbol{C}\cdot\boldsymbol{a}]_\times = \boldsymbol{C}\cdot[\boldsymbol{a}]_\times\cdot\boldsymbol{C}^T \tag{B.105}$$

$$\boldsymbol{C}\cdot(\boldsymbol{a}\times\boldsymbol{b}) = (\boldsymbol{C}\cdot\boldsymbol{a})\times(\boldsymbol{C}\cdot\boldsymbol{b}) \tag{B.106}$$

Powers of the skew-symmetric matrix

The powers of the skew-symmetric matrix of a vector $\boldsymbol{a} \in \mathbb{R}^3$ are given by:

$$
\begin{aligned}
[\boldsymbol{a}]_\times^2 =& \boldsymbol{a}\cdot\boldsymbol{a}^T - \|\boldsymbol{a}\|^2\cdot\boldsymbol{I}_{3\times3} && \text{(B.107)}\\
[\boldsymbol{a}]_\times^3 =& (\boldsymbol{a}\cdot\boldsymbol{a}^T - \|\boldsymbol{a}\|^2\cdot\boldsymbol{I}_{3\times3})[\boldsymbol{a}]_\times\\
=& \boldsymbol{a}\cdot\boldsymbol{a}^T[\boldsymbol{a}]_\times - \|\boldsymbol{a}\|^2[\boldsymbol{a}]_\times\\
=& \boldsymbol{a}(-[\boldsymbol{a}]_\times\cdot\boldsymbol{a})^T - \|\boldsymbol{a}\|^2[\boldsymbol{a}]_\times\\
=& -\boldsymbol{a}\cdot(\boldsymbol{a}\times\boldsymbol{a})^T - \|\boldsymbol{a}\|^2[\boldsymbol{a}]_\times\\
=& -\|\boldsymbol{a}\|^2[\boldsymbol{a}]_\times && \text{(B.108)}\\
[\boldsymbol{a}]_\times^4 =& [\boldsymbol{a}]_\times^3\cdot[\boldsymbol{a}]_\times\\
=& -\|\boldsymbol{a}\|^2[\boldsymbol{a}]_\times^2 && \text{(B.109)}\\
[\boldsymbol{a}]_\times^5 =& [\boldsymbol{a}]_\times^3\cdot[\boldsymbol{a}]_\times^2\\
=& -\|\boldsymbol{a}\|^2[\boldsymbol{a}]_\times\cdot(\boldsymbol{a}\cdot\boldsymbol{a}^T - \|\boldsymbol{a}\|^2\cdot\boldsymbol{I}_{3\times3})\\
=& \|\boldsymbol{a}\|^4[\boldsymbol{a}]_\times && \text{(B.110)}\\
[\boldsymbol{a}]_\times^6 =& [\boldsymbol{a}]_\times^5\cdot[\boldsymbol{a}]_\times\\
=& \|\boldsymbol{a}\|^4[\boldsymbol{a}]_\times^2 && \text{(B.111)}\\
[\boldsymbol{a}]_\times^7 =& [\boldsymbol{a}]_\times^5\cdot[\boldsymbol{a}]_\times^2\\
=& -\|\boldsymbol{a}\|^6[\boldsymbol{a}]_\times && \text{(B.112)}
\end{aligned}
$$

and so on.

Cross-product in quaternion notation

Using the concept of expanded vectors, being $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ three vectors, $\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}\in\mathbb{R}^3$, the cross-product of the expanded vectors is defined as:

$$
\bar{\boldsymbol{c}} = \bar{\boldsymbol{a}}\times\bar{\boldsymbol{b}} = \cdot\begin{bmatrix}0\\ \boldsymbol{a}\end{bmatrix}\cdot\times\cdot\begin{bmatrix}0\\ \boldsymbol{b}\end{bmatrix}\cdot = \cdot\begin{bmatrix}0\\ \boldsymbol{a}\times\boldsymbol{b}\end{bmatrix}\cdot = \cdot\begin{bmatrix}0\\ \boldsymbol{c}\end{bmatrix}\cdot \tag{B.113}
$$

A very useful consequence appears:

$$
\begin{aligned}
\bar{\boldsymbol{c}} =& \frac{1}{2}(\bar{\boldsymbol{a}}\otimes\bar{\boldsymbol{b}} + \bar{\boldsymbol{b}}^{-1}\otimes\bar{\boldsymbol{a}})\\
=& \frac{1}{2}(Q_-(\bar{\boldsymbol{b}}) + Q_+(\bar{\boldsymbol{b}}^{-1}))\cdot\bar{\boldsymbol{a}}\\
=& \frac{1}{2}(Q_-(\bar{\boldsymbol{b}}) + Q_+(\bar{\boldsymbol{b}})^T)\cdot\bar{\boldsymbol{a}}\\
=& \frac{1}{2}\begin{bmatrix}0 & -\boldsymbol{b}^T+\boldsymbol{b}^T\\ \boldsymbol{b}-\boldsymbol{b} & -[\boldsymbol{b}]_\times - [\boldsymbol{b}]_\times^T\end{bmatrix}\cdot\bar{\boldsymbol{a}}\\
=& \frac{1}{2}\begin{bmatrix}0 & \boldsymbol{0}_{1\times3}\\ \boldsymbol{0}_{3\times1} & -2\cdot[\boldsymbol{b}]_\times\end{bmatrix}\cdot\bar{\boldsymbol{a}}\\
=& \cdot\begin{bmatrix}0\\ -\boldsymbol{b}\times\boldsymbol{a}\end{bmatrix}\cdot = \cdot\begin{bmatrix}0\\ \boldsymbol{a}\times\boldsymbol{b}\end{bmatrix}\cdot
\end{aligned} \tag{B.114}
$$

Derivative of the cross-product

Given a vector $\boldsymbol{c}\in\mathbb{R}^3$ and two other vectors $\boldsymbol{a}=\boldsymbol{a}(\boldsymbol{c})\in\mathbb{R}^3$ and $\boldsymbol{b}=\boldsymbol{b}(\boldsymbol{c})\in\mathbb{R}^3$ that depend on $\boldsymbol{c}$, the derivative of the cross product is given by:

$$
\left[\frac{\partial(\boldsymbol{a}\times\boldsymbol{b})}{\partial\boldsymbol{c}}\right] = [\boldsymbol{a}]_\times\cdot\frac{\partial\boldsymbol{b}}{\partial\boldsymbol{c}} - [\boldsymbol{b}]_\times\cdot\frac{\partial\boldsymbol{a}}{\partial\boldsymbol{c}} \tag{B.115}
$$

B.3. Rotations using unit quaternions

As it was previously introduced, unit quaternions can be used to represent pure rotations in the euclidean space.

B.3.1. Vector rotation formula

Given a vector $\boldsymbol{x} \in \mathbb{R}^3$, the right-hand rotated vector $\boldsymbol{x}' \in \mathbb{R}^3$ defined by a unit vector $\boldsymbol{u} \in \mathbb{R}^3$, and an angle $\phi \in \mathbb{R}$ can be done by decomposing the vector $\boldsymbol{x}$ into a part $\boldsymbol{x}_\parallel$ parallel to $\boldsymbol{u}$, and a part $\boldsymbol{x}_\perp$ orthogonal to $\boldsymbol{u}$, so that,

$$\boldsymbol{x} = \boldsymbol{x}_\parallel + \boldsymbol{x}_\perp \tag{B.116}$$

The parts can be easily computed:

$$\boldsymbol{x}_\parallel = \boldsymbol{u} \cdot \boldsymbol{u}^T \cdot \boldsymbol{x} \tag{B.117}$$

$$\boldsymbol{x}_\perp = \boldsymbol{x} - \boldsymbol{u} \cdot \boldsymbol{u}^T \cdot \boldsymbol{x} \tag{B.118}$$

Upon rotation, the parallel part does not rotate:

$$\boldsymbol{x}'_\parallel = \boldsymbol{x}_\parallel \tag{B.119}$$

and the orthogonal part experiences a planar rotation in the plane normal to $\boldsymbol{u}$. An orthogonal base $\{\boldsymbol{e}_1, \boldsymbol{e}_2\}$ of this plane can be created with:

$$\boldsymbol{e}_1 = \boldsymbol{x}_\perp \tag{B.120}$$

$$\boldsymbol{e}_2 = \boldsymbol{u} \times \boldsymbol{x}_\perp = \boldsymbol{u} \times \boldsymbol{x} \tag{B.121}$$

satisfying $\|\boldsymbol{e}_1\| = \|\boldsymbol{e}_2\|$.

A rotation of ϕ *rad* on this plane, produces:

$$\begin{aligned} \boldsymbol{x}'_\perp &= \boldsymbol{e}_1 \cdot \cos\phi + \boldsymbol{e}_2 \cdot \sin\phi \\ &= \boldsymbol{x}_\perp \cdot \cos\phi + (\boldsymbol{u} \times \boldsymbol{x}) \cdot \sin\phi \end{aligned} \tag{B.122}$$

And the rotated vector $\boldsymbol{x}'$ is given:

$$\begin{aligned} \boldsymbol{x}' &= \boldsymbol{x}'_\parallel + \boldsymbol{x}'_\perp \\ &= \boldsymbol{x}_\parallel + \boldsymbol{x}_\perp \cdot \cos\phi + (\boldsymbol{u} \times \boldsymbol{x}) \cdot \sin\phi \end{aligned} \tag{B.123}$$

which is known as the vector rotation formula.

The vector $\boldsymbol{v} = \phi \cdot \boldsymbol{u}$ is called rotation vector.

B.3.2. Rotation matrix and rotation vector

Rotation vector to rotation matrix transformation: Rodrigues rotation formula

The rotation matrix $\boldsymbol{R} \in \mathbb{R}^{3\times3}$, is defined from the rotation vector $\boldsymbol{v} = \phi \cdot \boldsymbol{u}$,

$$\boldsymbol{R} = \boldsymbol{R}\{\boldsymbol{v}\} = e^{[\boldsymbol{v}]_\times} \tag{B.124}$$

Writing the Taylor expansion:

$$\begin{aligned} \boldsymbol{R}\{\boldsymbol{v}\} &= e^{[\boldsymbol{v}]_\times} = e^{\phi\cdot[\boldsymbol{u}]_\times} = \boldsymbol{I} + \phi \cdot [\boldsymbol{u}]_\times + \frac{1}{2}\phi^2 \cdot [\boldsymbol{u}]_\times^2 + \frac{1}{3}\phi^3 \cdot [\boldsymbol{u}]_\times^3 + \ldots \\ &= \boldsymbol{I} + \sin\phi\,[\boldsymbol{u}]_\times + (1-\cos\phi)\,[\boldsymbol{u}]_\times^2 \end{aligned} \tag{B.125}$$

which is called Rodrigues rotation formula.

B.3.3. Rotation of a vector using rotation matrices

Rotating a vector $\boldsymbol{x}$ by ϕ around $\boldsymbol{u}$ is performed with the linear product:

$$\boldsymbol{x}' = \boldsymbol{R} \cdot \boldsymbol{x} \tag{B.126}$$

B.3.4. Unit quaternion and rotation vector

Rotation vector to quaternion transformation

Given the rotation vector $\boldsymbol{v} = \phi \cdot \boldsymbol{u}$, representing a right-handed rotation of ϕ rad around the axis given by the unit vector $\boldsymbol{u} = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix}^T$, the following unit quaternion is built:

$$\boldsymbol{q} = \boldsymbol{q}\{\boldsymbol{v}\} = e^{\frac{\boldsymbol{v}}{2}} \tag{B.127}$$

What is called the rotation vector to quaternion conversion.

Developing the previous equation using the extension of the Euler formula:

$$\boldsymbol{q} = \boldsymbol{q}\{\boldsymbol{v}\} = \boldsymbol{q}\{\phi \cdot \boldsymbol{u}\} = e^{\frac{\boldsymbol{v}}{2}} = e^{\frac{\phi}{2} \cdot \boldsymbol{u}} = \cos\frac{\phi}{2} + \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \cdot \sin\frac{\phi}{2} = \begin{bmatrix} \cos\frac{\phi}{2} \\ \boldsymbol{u} \cdot \sin\frac{\phi}{2} \end{bmatrix} \tag{B.128}$$

which, as expected, gives, as a result, a unit quaternion.

Putting the expression in terms of $\boldsymbol{v}$, $\phi = \|\boldsymbol{v}\|$, and $\boldsymbol{u} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}$:

$$\boldsymbol{q} = \begin{bmatrix} \cos\frac{\|\boldsymbol{v}\|}{2} \\ \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|} \cdot \sin\frac{\|\boldsymbol{v}\|}{2} \end{bmatrix} \tag{B.129}$$

which is valid only if $\|\boldsymbol{v}\| \neq 0$.

In case that $\|\boldsymbol{v}\| \approx 0$, the small angle approximation is used, $\cos\frac{\|\boldsymbol{v}\|}{2} \approx 1$ and $\sin\frac{\|\boldsymbol{v}\|}{2} \approx \frac{\|\boldsymbol{v}\|}{2}$, getting:

$$\boldsymbol{q} \approx \begin{bmatrix} 1 \\ \frac{\boldsymbol{v}}{2} \end{bmatrix} \tag{B.130}$$

Quaternion to rotation vector transformation

The inverse transformation of the rotation vector to quaternion transformation can be obtained:

$$\phi = 2 \cdot \arctan\left(\frac{\|\boldsymbol{q}_{\boldsymbol{v}}\|}{q_w}\right) \tag{B.131}$$

$$\boldsymbol{u} = \frac{\boldsymbol{q}_{\boldsymbol{v}}}{\|\boldsymbol{q}_{\boldsymbol{v}}\|} \tag{B.132}$$

which is valid only if $\|\boldsymbol{q}_{\boldsymbol{v}}\| \neq 0$.

In case that $\|\boldsymbol{q}_{\boldsymbol{v}}\| \approx 0$:

$$\phi \approx 0 \tag{B.133}$$

$$\boldsymbol{u} \approx \boldsymbol{q}_{\boldsymbol{v}} \tag{B.134}$$

Jacobian matrices of the rotation vector to quaternion transformation

In the case that $\|\boldsymbol{v}\| \neq 0$:

$$\left[\frac{\partial \boldsymbol{q}\{\boldsymbol{v}\}}{\partial \boldsymbol{v}}\right]_{4\times 3} = \begin{bmatrix} -\frac{\boldsymbol{v}^T}{2\cdot\|\boldsymbol{v}\|} \cdot \sin\frac{\|\boldsymbol{v}\|}{2} \\ \frac{1}{\|\boldsymbol{v}\|} \cdot \left(\boldsymbol{I}_{3\times 3} - \frac{\boldsymbol{v}\cdot\boldsymbol{v}^T}{\|\boldsymbol{v}\|^2}\right) \cdot \sin\frac{\|\boldsymbol{v}\|}{2} + \frac{\boldsymbol{v}\cdot\boldsymbol{v}^T}{2\cdot\|\boldsymbol{v}\|^2} \cdot \cos\frac{\|\boldsymbol{v}\|}{2} \end{bmatrix} \tag{B.135}$$

In case that $\|\boldsymbol{v}\| \approx 0$:

$$\left[\frac{\partial \boldsymbol{q}\{\boldsymbol{v}\}}{\partial \boldsymbol{v}}\right]_{4\times 3} \approx \begin{bmatrix} \boldsymbol{0}_{1\times 3} \\ \frac{1}{2}\boldsymbol{I}_{3\times 3} \end{bmatrix} \tag{B.136}$$

B.3.5. Rotation of a vector using quaternions: sandwich product

Rotating a vector $\boldsymbol{x} \in \mathbb{R}^3$ by an angle of $\phi \in \mathbb{R}$ around the axis defined by the unit vector $\boldsymbol{u} \in \mathbb{R}^3$ is performed with the double quaternion product, known as the sandwich product:

$$\bar{\boldsymbol{x}}' = \boldsymbol{q} \otimes \bar{\boldsymbol{x}} \otimes \boldsymbol{q}^* \tag{B.137}$$

being $\bar{\boldsymbol{x}}$ the expanded vector of $\boldsymbol{x}$.

Jacobian matrices of the sandwich product

$$\left[\frac{\partial\left(\boldsymbol{q}\otimes\bar{\boldsymbol{x}}\otimes\boldsymbol{q}^{*}\right)}{\partial\boldsymbol{x}}\right]_{4\times3}=\boldsymbol{Q}_{+}(\boldsymbol{q})\cdot\boldsymbol{Q}_{-}(\boldsymbol{q}^{*})\cdot\begin{bmatrix}\boldsymbol{0}_{1\times3}\\ \boldsymbol{I}_{3\times3}\end{bmatrix} \tag{B.138}$$

$$\left[\frac{\partial\left(\boldsymbol{q}\otimes\bar{\boldsymbol{x}}\otimes\boldsymbol{q}^{*}\right)}{\partial\boldsymbol{q}}\right]_{4\times4}=\boldsymbol{Q}_{+}(\boldsymbol{q})\cdot\boldsymbol{Q}_{+}(\bar{\boldsymbol{x}})\cdot\begin{bmatrix}1 & \boldsymbol{0}_{1\times3}\\ \boldsymbol{0}_{3\times1} & -\boldsymbol{I}_{3\times3}\end{bmatrix}+\boldsymbol{Q}_{-}(\bar{\boldsymbol{x}}\otimes\boldsymbol{q}^{*}) \tag{B.139}$$

$$=\boldsymbol{Q}_{+}(\boldsymbol{q}\otimes\bar{\boldsymbol{x}})\cdot\begin{bmatrix}1 & \boldsymbol{0}_{1\times3}\\ \boldsymbol{0}_{3\times1} & -\boldsymbol{I}_{3\times3}\end{bmatrix}+\boldsymbol{Q}_{-}(\bar{\boldsymbol{x}}\otimes\boldsymbol{q}^{*}) \tag{B.140}$$

Jacobian matrices of the inverse of the sandwich product

$$\left[\frac{\partial\left(\boldsymbol{q}^{*}\otimes\bar{\boldsymbol{x}}\otimes\boldsymbol{q}\right)}{\partial\boldsymbol{x}}\right]_{4\times3}=\boldsymbol{Q}_{+}(\boldsymbol{q}^{*})\cdot\boldsymbol{Q}_{-}(\boldsymbol{q})\cdot\begin{bmatrix}\boldsymbol{0}_{1\times3}\\ \boldsymbol{I}_{3\times3}\end{bmatrix} \tag{B.141}$$

$$\left[\frac{\partial\left(\boldsymbol{q}^{*}\otimes\bar{\boldsymbol{x}}\otimes\boldsymbol{q}\right)}{\partial\boldsymbol{q}}\right]_{4\times4}=\boldsymbol{Q}_{+}(\boldsymbol{q}^{*})\cdot\boldsymbol{Q}_{+}(\bar{\boldsymbol{x}})+\boldsymbol{Q}_{-}(\bar{\boldsymbol{x}}\otimes\boldsymbol{q})\cdot\begin{bmatrix}1 & \boldsymbol{0}_{1\times3}\\ \boldsymbol{0}_{3\times1} & -\boldsymbol{I}_{3\times3}\end{bmatrix} \tag{B.142}$$

$$=\boldsymbol{Q}_{+}(\boldsymbol{q}^{*}\otimes\bar{\boldsymbol{x}})+\boldsymbol{Q}_{-}(\bar{\boldsymbol{x}}\otimes\boldsymbol{q})\cdot\begin{bmatrix}1 & \boldsymbol{0}_{1\times3}\\ \boldsymbol{0}_{3\times1} & -\boldsymbol{I}_{3\times3}\end{bmatrix} \tag{B.143}$$

Properties

Quaternion sign ambiguity

The rotation performed by the unit quaternion $\boldsymbol{q}$ and the unit quaternion $-\boldsymbol{q}$ produce the same result, since:

$$\boldsymbol{x}'=(-\boldsymbol{q})\otimes\bar{\boldsymbol{x}}\otimes(-\boldsymbol{q})^{*}=\boldsymbol{q}\otimes\bar{\boldsymbol{x}}\otimes\boldsymbol{q}^{*} \tag{B.144}$$

Null rotation

The identity quaternion, $\boldsymbol{q}_{I}$, encodes the null rotation, since:

$$\boldsymbol{x}'=\boldsymbol{q}_{I}\otimes\bar{\boldsymbol{x}}\otimes\boldsymbol{q}_{I}^{*}=\bar{\boldsymbol{x}} \tag{B.145}$$

Inverse rotation

The conjugate quaternion encodes the inverse rotation, since:

$$\boldsymbol{x}'=(\boldsymbol{q}^{*})\otimes\bar{\boldsymbol{x}}\otimes(\boldsymbol{q}^{*})^{*}=\boldsymbol{q}^{*}\otimes\bar{\boldsymbol{x}}\otimes\boldsymbol{q} \tag{B.146}$$

B.3.6. Rotation matrix and quaternion

Unit quaternions act as rotation operators in a way somewhat similar to rotation matrices:

$$\bar{\boldsymbol{x}}'=\boldsymbol{q}\otimes\bar{\boldsymbol{x}}\otimes\boldsymbol{q}^{*}\quad\boldsymbol{x}'=\boldsymbol{R}\cdot\boldsymbol{x} \tag{B.147}$$

Therefore:

$$\bar{\boldsymbol{x}}'=\boldsymbol{q}\otimes\bar{\boldsymbol{x}}\otimes\boldsymbol{q}^{*}=\left[\begin{array}{c}0\\ \hdashline \boldsymbol{R}\cdot\boldsymbol{x}\end{array}\right] \tag{B.148}$$

Quaternion to rotation matrix transformation

As both sides of equation B.148 are linear in $\boldsymbol{x}$, an expression of the rotation matrix equivalent to the quaternion is found by developing the left-hand side and identifying terms on the right, yielding:

$$\boldsymbol{R}=\boldsymbol{R}\{\boldsymbol{q}\}=\begin{bmatrix}q_w^2+q_x^2-q_y^2-q_z^2 & 2q_xq_y-2q_wq_z & 2q_xq_z+2q_wq_y\\ 2q_xq_y+2q_wq_z & q_w^2-q_x^2+q_y^2-q_z^2 & 2q_yq_z-2q_wq_x\\ 2q_xq_z-2q_wq_y & 2q_yq_z+2q_wq_x & q_w^2-q_x^2-q_y^2+q_z^2\end{bmatrix} \tag{B.149}$$

Other alternative formula for the rotation matrix is using the following expression:

$$\bar{\boldsymbol{x}}'=\boldsymbol{q}\otimes\bar{\boldsymbol{x}}\otimes\boldsymbol{q}^{*}=\boldsymbol{Q}_{-}(\boldsymbol{q}^{*})\cdot\boldsymbol{Q}_{+}(\boldsymbol{q})\cdot\bar{\boldsymbol{x}}=\left[\begin{array}{c}0\\ \hdashline \boldsymbol{R}\cdot\boldsymbol{x}\end{array}\right] \tag{B.150}$$

Obtaining:

$$\boldsymbol{R}=\left(q_w^2-\boldsymbol{q}_{\boldsymbol{v}}^{T}\cdot\boldsymbol{q}_{\boldsymbol{v}}\right)\cdot\boldsymbol{I}_{3\times3}+2\boldsymbol{q}_{\boldsymbol{v}}\cdot\boldsymbol{q}_{\boldsymbol{v}}^{T}+2\left[\boldsymbol{q}_{\boldsymbol{v}}\right]_{x} \tag{B.151}$$

Rotation matrix to quaternion transformation

Given a rotation matrix $\boldsymbol{R} \in \mathbb{R}^{3\times 3}$:

$$\boldsymbol{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{B.152}$$

The equivalent unit quaternion $\boldsymbol{q}$ can be calculated in four different ways:

- Method 1:

$$q_w = \frac{1}{2}\sqrt{1 + r_{11} + r_{22} + r_{33}} \tag{B.153}$$

$$q_x = \frac{1}{4q_w}(r_{32} - r23) \tag{B.154}$$

$$q_y = \frac{1}{4q_w}(r_{13} - r31) \tag{B.155}$$

$$q_z = \frac{1}{4q_w}(r_{21} - r12) \tag{B.156}$$

- Method 2:

$$q_w = \frac{1}{4q_x}(r_{32} - r23) \tag{B.157}$$

$$q_x = \frac{1}{2}\sqrt{1 + r_{11} - r_{22} - r_{33}} \tag{B.158}$$

$$q_y = \frac{1}{4q_x}(r_{12} + r21) \tag{B.159}$$

$$q_z = \frac{1}{4q_x}(r_{13} + r31) \tag{B.160}$$

- Method 3:

$$q_w = \frac{1}{4q_y}(r_{13} - r31) \tag{B.161}$$

$$q_x = \frac{1}{4q_y}(r_{12} + r21) \tag{B.162}$$

$$q_y = \frac{1}{2}\sqrt{1 - r_{11} + r_{22} - r_{33}} \tag{B.163}$$

$$q_z = \frac{1}{4q_y}(r_{23} + r32) \tag{B.164}$$

- Method 4:

$$q_w = \frac{1}{4q_z}(r_{21} - r12) \tag{B.165}$$

$$q_x = \frac{1}{4q_z}(r_{13} + r31) \tag{B.166}$$

$$q_y = \frac{1}{4q_z}(r_{23} + r32) \tag{B.167}$$

$$q_z = \frac{1}{2}\sqrt{1 - r_{11} - r_{22} + r_{33}} \tag{B.168}$$

Properties of the rotation matrix with respect to the quaternions

The rotation matrix $\boldsymbol{R}$ has the following properties with respect to the quaternions:

- The identity quaternion, $\boldsymbol{q}_I$, encodes the null rotation.

$$\boldsymbol{R}\left\{\left[\begin{array}{c} 1 \\ \hdashline \mathbf{0}_{3\times 1} \end{array}\right]\right\} = \boldsymbol{I}_{3\times 3} \tag{B.169}$$

- A quaternion $\boldsymbol{q}$ and its negative $-\boldsymbol{q}$ encode the same rotation.

$$\boldsymbol{R}\{-\boldsymbol{q}\} = \boldsymbol{R}\{\boldsymbol{q}\} \tag{B.170}$$

- The conjugate quaternion encodes the inverse rotation.

$$\boldsymbol{R}\{\boldsymbol{q}^*\} = \boldsymbol{R}\{\boldsymbol{q}\}^T \tag{B.171}$$

- The quaternion product composes consecutive rotation in the same order as rotation matrices do.

$$\boldsymbol{R}\{\boldsymbol{q_1} \otimes \boldsymbol{q_2}\} = \boldsymbol{R}\{\boldsymbol{q_1}\} \cdot \boldsymbol{R}\{\boldsymbol{q_2}\} \tag{B.172}$$

B.3.7. Angular error using quaternions for rotations

As presented in Appendix B.3.4, given a rotation vector, $\boldsymbol{\theta} = \phi \cdot \boldsymbol{u}$, the quaternion, $\boldsymbol{q}$, that encodes this rotation can be calculated by means of the rotation vector to quaternion conversion:

$$\boldsymbol{q} = \boldsymbol{q}\{\boldsymbol{\theta}\} = e^{\frac{\boldsymbol{\theta}}{2}}$$

If the rotation angle tends to be small, $\phi \to 0$, then, the rotation vector tends to be small, $\boldsymbol{\theta} \to \mathbf{0}$, and therefore, the quaternion that encodes this rotation tends to be the identity quaternion, $\boldsymbol{q} \to \boldsymbol{q}_I$.

For a small rotation angle, $\delta\phi \to 0$, given by a rotation vector, $\delta\boldsymbol{\theta} = \delta\phi \cdot \boldsymbol{u} \to \mathbf{0}$, the quaternion that encodes this small rotation is defined as:

$$\delta\boldsymbol{q} = \delta\boldsymbol{q}\{\delta\boldsymbol{\theta}\} = e^{\frac{\delta\boldsymbol{\theta}}{2}} \approx \begin{bmatrix} 1 \\ \frac{1}{2} \cdot \delta\boldsymbol{\theta} \end{bmatrix} \tag{B.173}$$

This angular error, $\delta\boldsymbol{\theta}$, can be used to calculate an error quaternion, $\delta\boldsymbol{q}$, using a 3-dimensional magnitude, and therefore the minimum amount of information. This angular error is specially useful in state estimation when using error-values.

Jacobian matrices

$$\left[\frac{\partial \delta\boldsymbol{q}}{\partial \delta\boldsymbol{\theta}}\right]_{4\times3} = \begin{bmatrix} \mathbf{0}_{1\times3} \\ \frac{1}{2} I_{3\times3} \end{bmatrix} \tag{B.174}$$

$$\left[\frac{\partial \delta\boldsymbol{\theta}}{\partial \delta\boldsymbol{q}}\right]_{3\times4} = \begin{bmatrix} \mathbf{0}_{3\times1} & 2 \cdot I_{3\times3} \end{bmatrix} \tag{B.175}$$

B.3.8. Decomposition of rotations

A quaternion can be decomposed in two rotations, as proposed in (Hu et al., 2006).

B.4. Perturbations and time-derivatives

B.4.1. Local perturbations

A perturbed orientation, $\tilde{\boldsymbol{q}}$ or $\tilde{\boldsymbol{R}}$, may be expressed as the composition of the unperturbed orientation, $\boldsymbol{q}$ or $\boldsymbol{R}$, with a small local perturbation, $\Delta\boldsymbol{q}_L$ or $\Delta\boldsymbol{R}_L$. Because of the Hamilton convention, this local perturbation appears at the right hand side of the composition product:

$$\tilde{\boldsymbol{q}} = \boldsymbol{q} \otimes \Delta\boldsymbol{q}_L \tag{B.176}$$

$$\tilde{\boldsymbol{R}} = \boldsymbol{R} \cdot \Delta\boldsymbol{R}_L \tag{B.177}$$

If the perturbation angle $\Delta\phi$ of the perturbation rotation vector $\Delta\boldsymbol{\theta}_L = \Delta\phi \cdot \boldsymbol{u}$ is small, then the perturbation quaternion (and the rotation matrix) can be approximated by the Taylor expansion of the rotation vector to quaternion transformation up to the linear terms:

$$\Delta\boldsymbol{q}_L = \begin{bmatrix} 1 \\ \frac{1}{2}\Delta\boldsymbol{\theta}_L \end{bmatrix} + O\left(\|\Delta\boldsymbol{\theta}_L\|^2\right) \tag{B.178}$$

$$\Delta\boldsymbol{R}_L = \boldsymbol{I}_{3\times3} + [\Delta\boldsymbol{\theta}_L]_\times + O\left(\|\Delta\boldsymbol{\theta}_L\|^2\right) \tag{B.179}$$

Thanks to this equation, the expressions for the time-derivatives can be easily developed.

Considering $\boldsymbol{q} = \boldsymbol{q}(t)$ as the original state, $\tilde{\boldsymbol{q}} = \boldsymbol{q}(t+\Delta t)$ as the perturbed state, and applying the definition of the derivative:

$$\dot{\boldsymbol{q}} = \frac{\mathrm{d}\boldsymbol{q}(t)}{\mathrm{d}t} = \lim_{\Delta t \to 0} \frac{\boldsymbol{q}(t+\Delta t) - \boldsymbol{q}(t)}{\Delta t} \tag{B.180}$$

Defining $\boldsymbol{\omega}_L = \boldsymbol{\omega}_{L|G}^L$ as the angular rates vector in the local frame:

$$\boldsymbol{\omega}_L(t) = \frac{\mathrm{d}\boldsymbol{\theta}_L(t)}{\mathrm{d}t} = \lim_{\Delta t \to 0} \frac{\Delta \boldsymbol{\theta}_L}{\Delta t} \tag{B.181}$$

The time derivative is given by:

$$\begin{aligned}
\dot{\boldsymbol{q}} &= \lim_{\Delta t \to 0} \frac{\boldsymbol{q}(t+\Delta t) - \boldsymbol{q}(t)}{\Delta t} \\
&= \lim_{\Delta t \to 0} \frac{\boldsymbol{q} \otimes \Delta \boldsymbol{q}_L - \boldsymbol{q}}{\Delta t} \\
&= \lim_{\Delta t \to 0} \frac{\boldsymbol{q} \otimes \left(\begin{bmatrix} 1 \\ \frac{1}{2}\Delta \boldsymbol{\theta}_L \end{bmatrix} - \begin{bmatrix} 1 \\ \mathbf{0}_{3\times 1} \end{bmatrix} \right)}{\Delta t} \\
&= \lim_{\Delta t \to 0} \frac{\boldsymbol{q} \otimes \begin{bmatrix} 0 \\ \frac{1}{2}\Delta \boldsymbol{\theta}_L \end{bmatrix}}{\Delta t} \\
&= \frac{1}{2} \cdot \boldsymbol{q} \otimes \cdot \begin{bmatrix} 0 \\ \boldsymbol{\omega}_L \end{bmatrix} \cdot \\
&= \frac{1}{2} \cdot \boldsymbol{q} \otimes \bar{\boldsymbol{\omega}}_L
\end{aligned} \tag{B.182}$$

And it can be defined:

$$\boldsymbol{\Omega}_L(\boldsymbol{\omega}) = \boldsymbol{Q}_-(\bar{\boldsymbol{\omega}}) = \begin{bmatrix} 0 & -\boldsymbol{\omega}^T \\ \boldsymbol{\omega} & -[\boldsymbol{\omega}]_\times \end{bmatrix} \tag{B.183}$$

Getting therefore:

$$\dot{\boldsymbol{q}} = \frac{1}{2} \cdot \boldsymbol{q} \otimes \bar{\boldsymbol{\omega}}_L = \frac{1}{2} \cdot \boldsymbol{\Omega}_L(\boldsymbol{\omega}) \cdot \boldsymbol{q} \tag{B.184}$$

The equivalent equation using rotation matrices is:

$$\dot{\boldsymbol{R}} = \boldsymbol{R} \cdot [\boldsymbol{\omega}_L]_\times \tag{B.185}$$

B.4.2. Global perturbations

Global perturbations appear at the left hand side of the product:

$$\tilde{\boldsymbol{q}} = \Delta \boldsymbol{q}_G \otimes \boldsymbol{q} \tag{B.186}$$

$$\tilde{\boldsymbol{R}} = \Delta \boldsymbol{R}_G \cdot \boldsymbol{R} \tag{B.187}$$

Defining $\boldsymbol{\omega}_G = \boldsymbol{\omega}_{L|G}^G$ as the angular rates vector in the global frame:

$$\boldsymbol{\omega}_G = \frac{\mathrm{d}\boldsymbol{\theta}_G(t)}{\mathrm{d}t} = \lim_{\Delta t \to 0} \frac{\Delta \boldsymbol{\theta}_G}{\Delta t} \tag{B.188}$$

The time-derivatives are given by:

$$\dot{\boldsymbol{q}} = \frac{1}{2} \cdot \bar{\boldsymbol{\omega}}_G \otimes \boldsymbol{q} \tag{B.189}$$

And it can be defined:

$$\boldsymbol{\Omega}_G(\boldsymbol{\omega}) = \boldsymbol{Q}_+(\bar{\boldsymbol{\omega}}) \tag{B.190}$$

Getting therefore:

$$\dot{\boldsymbol{q}} = \frac{1}{2} \cdot \bar{\boldsymbol{\omega}}_G \otimes \boldsymbol{q} = \frac{1}{2} \cdot \boldsymbol{\Omega}_G(\boldsymbol{\omega}) \cdot \boldsymbol{q} \tag{B.191}$$

The equivalent equation using rotation matrices is:

$$\dot{\boldsymbol{R}} = [\boldsymbol{\omega}_G]_\times \cdot \boldsymbol{R} \tag{B.192}$$

B.4.3. Global-to-local relations

Using both the equation for local and global time-derivatives for quaternions:

$$\dot{\boldsymbol{q}} = \frac{1}{2} \cdot \boldsymbol{q} \otimes \bar{\boldsymbol{\omega}}_L$$
$$\dot{\boldsymbol{q}} = \frac{1}{2} \cdot \bar{\boldsymbol{\omega}}_G \otimes \boldsymbol{q}$$

Therefore, the relationship between local and global perturbations is given by:

$$\bar{\boldsymbol{\omega}}_L = \boldsymbol{q}^* \otimes \bar{\boldsymbol{\omega}}_G \otimes \boldsymbol{q} \tag{B.193}$$
$$\bar{\boldsymbol{\omega}}_G = \boldsymbol{q} \otimes \bar{\boldsymbol{\omega}}_L \otimes \boldsymbol{q}^* \tag{B.194}$$

The equivalent equations can be used for rotation matrices:

$$\dot{\boldsymbol{R}} = \boldsymbol{R} \cdot [\boldsymbol{\omega}_L]_\times$$
$$\dot{\boldsymbol{R}} = [\boldsymbol{\omega}_G]_\times \cdot \boldsymbol{R}$$

And therefore, the relationship between local and global perturbations is given by:

$$[\boldsymbol{\omega}_L]_\times = \boldsymbol{R}^T \cdot [\boldsymbol{\omega}_G]_\times \cdot \boldsymbol{R} \tag{B.195}$$
$$[\boldsymbol{\omega}_G]_\times = \boldsymbol{R} \cdot [\boldsymbol{\omega}_L]_\times \cdot \boldsymbol{R}^T \tag{B.196}$$

B.4.4. Time-derivative of the quaternion product

To obtain the time-derivative of the quaternion product, the regular formula for the derivative applied to the product is used:

$$\dot{(\boldsymbol{q}_1 \otimes \boldsymbol{q}_2)} = \dot{\boldsymbol{q}}_1 \otimes \boldsymbol{q}_2 + \boldsymbol{q}_1 \otimes \dot{\boldsymbol{q}}_2 \tag{B.197}$$

The rotation matrix equivalent is given by:

$$\dot{(\boldsymbol{R}_1 \cdot \boldsymbol{R}_2)} = \dot{\boldsymbol{R}}_1 \cdot \boldsymbol{R}_2 + \boldsymbol{R}_1 \cdot \dot{\boldsymbol{R}}_2 \tag{B.198}$$

Note that, since the quaternion product is not commutative, then:

$$\dot{(\boldsymbol{q}^2)} = \dot{(\boldsymbol{q} \otimes \boldsymbol{q})} = \dot{\boldsymbol{q}} \otimes \boldsymbol{q} + \boldsymbol{q} \otimes \dot{\boldsymbol{q}} \neq 2\boldsymbol{q} \otimes \dot{\boldsymbol{q}} \neq 2\dot{\boldsymbol{q}} \otimes \boldsymbol{q} \tag{B.199}$$

B.4.5. Other useful expressions with the derivative

Using the equations obtained for the local and global rotation rates using quaternions, the following expressions can be derived:

$$\bar{\boldsymbol{\omega}}_L = 2 \cdot \boldsymbol{q}^* \otimes \dot{\boldsymbol{q}} \tag{B.200}$$
$$\bar{\boldsymbol{\omega}}_G = 2 \cdot \dot{\boldsymbol{q}} \otimes \boldsymbol{q}^* \tag{B.201}$$

Similarly, using the equations obtained for the local and global rotation rates using rotation matrices, the following expressions can be derived:

$$[\boldsymbol{\omega}_L]_\times = \boldsymbol{R}^T \cdot \dot{\boldsymbol{R}} \tag{B.202}$$
$$[\boldsymbol{\omega}_G]_\times = \dot{\boldsymbol{R}} \cdot \boldsymbol{R}^T \tag{B.203}$$

B.5. Time-integration of rotation rates

Accumulating rotation over time in quaternion form is done by integrating the differential equation appropriate to the rotation rate definition, that is equation B.184 for a local rotation rate definition, and equation B.191 for a global rotation rate definition.

The discrete time is defined as $t_n = n \cdot \Delta t$.

B.5.1. Local Rotation Rate

Using the differential equation for a local rotation rate definition:

$$\dot{\boldsymbol{q}}(t) = \frac{1}{2} \cdot \boldsymbol{q}(t) \otimes \bar{\boldsymbol{\omega}}_L(t)$$

Developing the Taylor series of $\boldsymbol{q}(t_n + \Delta t) = \boldsymbol{q}_{n+1}$:

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n + \dot{\boldsymbol{q}}_n \Delta t + \frac{1}{2!}\ddot{\boldsymbol{q}}_n \Delta t^2 + \frac{1}{3!}\boldsymbol{q}_n^{(3)} \Delta t^3 + \frac{1}{4!}\boldsymbol{q}_n^{(4)} \Delta t^4 + \ldots \tag{B.204}$$

Being $\boldsymbol{q} = \boldsymbol{q}(t)$, $\boldsymbol{q}_n = \boldsymbol{q}(t_n)$, and $\boldsymbol{q}_n^{(i)}$ is the i-th derivative of $\boldsymbol{q}$ evaluated in $t = t_n$.

Zeroth order integration

In the case where the angular rate $\boldsymbol{\omega}_n$ is held constant over the period $\Delta t = t_{n+1} - t_n$, we have $\dot{\boldsymbol{\omega}} = 0$.

Forward integration

Assuming that the constant value of the angular velocity over the period Δt corresponds to $\boldsymbol{\omega}_n$, the velocity at the beginning of the period, the Taylor series, can be written as:

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n \otimes \left(1 + \frac{1}{2}\boldsymbol{\omega}_n \Delta t + \frac{1}{2!}\left(\frac{1}{2}\boldsymbol{\omega}_n \Delta t\right)^2 + \frac{1}{3!}\left(\frac{1}{2}\boldsymbol{\omega}_n \Delta t\right)^3 + \frac{1}{4!}\left(\frac{1}{2}\boldsymbol{\omega}_n \Delta t\right)^4 + \ldots\right) \tag{B.205}$$

in which the Taylor series of the exponential $e^{\boldsymbol{\omega}_n \Delta t/2}$ can be identified:

$$e^{\boldsymbol{\omega}_n \Delta t/2} = 1 + \frac{1}{2}\boldsymbol{\omega}_n \Delta t + \frac{1}{2!}\left(\frac{1}{2}\boldsymbol{\omega}_n \Delta t\right)^2 + \frac{1}{3!}\left(\frac{1}{2}\boldsymbol{\omega}_n \Delta t\right)^3 + \frac{1}{4!}\left(\frac{1}{2}\boldsymbol{\omega}_n \Delta t\right)^4 + \ldots \tag{B.206}$$

From rotation vector to quaternion conversion, this exponential corresponds to the quaternion representing the incremental rotation $\Delta\boldsymbol{\theta} = \boldsymbol{\omega}_n \Delta t$:

$$e^{\boldsymbol{\omega} \Delta t/2} = \boldsymbol{q}\{\boldsymbol{\omega} \Delta t\}$$

So, therefore:

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n \otimes \boldsymbol{q}\{\boldsymbol{\omega}_n \Delta t\} \tag{B.207}$$

Backwards integration

Considering that the constant velocity over the period Δt corresponds to $\boldsymbol{\omega}_{n+1}$, the velocity at the end of the period. A similar procedure can be used getting:

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n \otimes \boldsymbol{q}\{\boldsymbol{\omega}_{n+1} \Delta t\} \tag{B.208}$$

Midward integration

Similarly, if the velocity is considered constant at the mean rate over the period Δt (which is not necessary the velocity at the midpoint of the period):

$$\boldsymbol{\omega}_m = \frac{\boldsymbol{\omega}_{n+1} + \boldsymbol{\omega}_n}{2} \tag{B.209}$$

Therefore:

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n \otimes \boldsymbol{q}\{\boldsymbol{\omega}_m \Delta t\} \tag{B.210}$$

First order integration

The angular rate $\boldsymbol{\omega}(t)$ is now linear with time. Its first derivative therefore is constant, and all higher ones are zero:

$$\dot{\boldsymbol{\omega}} = \frac{\boldsymbol{\omega}_{n+1} - \boldsymbol{\omega}_n}{2} \tag{B.211}$$

$$\boldsymbol{\omega}^{(i)} = 0 \quad , \quad \forall i > 1 \tag{B.212}$$

The mean rate is given by:

$$\boldsymbol{\omega}_m = \boldsymbol{\omega}_n + \frac{1}{2}\dot{\boldsymbol{\omega}}\Delta t \tag{B.213}$$

Substituting in the Taylor series of $\boldsymbol{q}_{n+1}$:

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n \otimes \boldsymbol{q}\{\bar{\boldsymbol{\omega}}\Delta t\} + \frac{\Delta t^2}{48}\boldsymbol{q}_n \otimes (\boldsymbol{\omega}_n \otimes \boldsymbol{\omega}_{n+1} - \boldsymbol{\omega}_{n+1} \otimes \boldsymbol{\omega}_n) + ... \tag{B.214}$$

Substituting and neglecting terms of high multiplicity, results:

$$\boldsymbol{q}_{n+1} \approx \boldsymbol{q}_n \otimes \boldsymbol{q}\{\boldsymbol{\omega}_m \Delta t\} + \frac{\Delta t^2}{24}\boldsymbol{q}_n \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_n \times \boldsymbol{\omega}_{n+1} \end{bmatrix} \tag{B.215}$$

The result of this equation, $\boldsymbol{q}_{n+1}$, is not a unit quaternion, since two unit quaternion are added, and the addition operation does not preserve the norm. Therefore, a quaternion normalization has to be done if needed.

B.5.2. Global Rotation Rate

Similarly than before, using the differential equation for a global rotation rate definition:

$$\dot{\boldsymbol{q}}(t) = \frac{1}{2} \cdot \bar{\boldsymbol{\omega}}_G(t) \otimes \boldsymbol{q}(t)$$

the expressions for the zeroth order integration, and the first order integration can be calculated.

B.6. Considerations about the notation

There are several ways to define the quaternions. They are basically related to four binary choices:

- The order of its elements: real part first or last

$$\boldsymbol{q} = \begin{bmatrix} q_w \\ \boldsymbol{q}_v \end{bmatrix} \quad vs. \quad \boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_v \\ q_w \end{bmatrix}$$

- The basis elements multiplication formula: definition of the quaternion algebra:

$$i \cdot j = -j \cdot i = k \quad vs. \quad j \cdot i = -i \cdot j = k$$

  which correspond to different handedness, respectively:

$$right - handed \quad vs. \quad left - handed$$

  which means that, given a rotation axis $\boldsymbol{u}$, one quaternion $\boldsymbol{q}\{\boldsymbol{u} \cdot \theta\}$ rotates vectors an angle θ around $\boldsymbol{u}$ using the right hand rule, while the other quaternion uses the left hand rule.
- The function of the rotation operator: rotating frames or rotating vectors:

$$Passive \quad vs. \quad Active.$$

- In the passive case, the direction of the rotation operation: local-to-global or global-to-local:

$$\boldsymbol{x}_{global} = \boldsymbol{q} \otimes \boldsymbol{x}_{local} \otimes \boldsymbol{q}^* \quad vs. \quad \boldsymbol{x}_{local} = \boldsymbol{q} \otimes \boldsymbol{x}_{global} \otimes \boldsymbol{q}^*$$

This variety of choices leads to 12 different combinations.

The two most commonly used conventions, which are also the best documented, are Hamilton and JPL, detailed in Table B.1. JPL is mostly used in the aerospace domain, while Hamilton is more common to other engineering areas such as robotics (for example in Eigen, ROS, Google Ceres).

Along this thesis, the Hamilton convention has been used.

Quaternion type	Hamilton	JPL
Components order	$\boldsymbol{q} = \begin{bmatrix} q_w \\ \boldsymbol{q}_v \end{bmatrix}$	$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_v \\ q_w \end{bmatrix}$
Algebra	$i \cdot j = k$	$i \cdot j = -k$
Handedness	Right-handed	Left-handed
Function	Passive	Passive
Right-to-left products mean	Local-to-global	Global-to-local
Default notation, $\boldsymbol{q}$	$\boldsymbol{q}_L^G$	$\boldsymbol{q}_G^L$
Default operation	$\boldsymbol{x}_G = \boldsymbol{q} \otimes \boldsymbol{x}_L \otimes \boldsymbol{q}^*$	$\boldsymbol{x}_L = \boldsymbol{q} \otimes \boldsymbol{x}_G \otimes \boldsymbol{q}^*$

Table B.1: Hamilton vs. JPL quaternion conventions with respect to the 4 binary choices.

Appendix C

Rigid Body

C.1. Kinematic equations of the center of a rigid body

In this section, the kinematic equations of the center of a rigid body are obtained, using two different formalisms to represent the pose. This point is a reference frame, that is, it has position and orientation.

Supposing to have the situation represented in Fig. C.1, where a rigid body is moving in a static world, whose inertial reference frame is labeled as world reference frame. The rigid body has a rigidly attached reference frame, called robot reference frame, that represents the center of interest of the robot (e.g. the reference frame used in control).

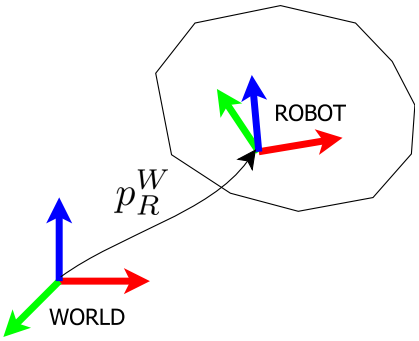


Figure C.1: Rigid body moving in an static world. The transformations between world and robot reference frames are indicated.

C.1.1. Using VQS

Pose

The pose of the robot in world coordinates, p_R^W, is represented with: $\boldsymbol{t}_R^W$ and $\boldsymbol{q}_R^W$

Linear velocity

The linear velocity of the robot with respect to world in world coordinates is defined as: $\dot{\boldsymbol{t}}_R^W = \boldsymbol{v}_{R|W}^W$.
The linear velocity of the robot with respect to world in robot coordinates can be obtained with:

$$\boldsymbol{v}_{R|W}^R = \left(\boldsymbol{q}_R^W\right)^* \otimes \boldsymbol{v}_{R|W}^W \otimes \boldsymbol{q}_R^W \tag{C.1}$$

Linear acceleration

The linear acceleration of the robot with respect to world in world coordinates is defined as: $\ddot{\boldsymbol{t}}_R^W = \boldsymbol{a}_{R|W}^W$.
The linear acceleration of the robot with respect to world in robot coordinates can be obtained with:

$$\boldsymbol{a}_{R|W}^R = \left(\boldsymbol{q}_R^W\right)^* \otimes \boldsymbol{a}_{R|W}^W \otimes \boldsymbol{q}_R^W \tag{C.2}$$

Angular velocity

As explained in Appendix B.4, when operating with rotations, they can be considered as a local or as a global rotations.

If a global rotation is used, then:

$$\dot{\boldsymbol{q}}_R^W = \frac{1}{2} \cdot \bar{\boldsymbol{\omega}}_{R|W}^W \otimes \boldsymbol{q}_R^W \tag{C.3}$$

If a local rotation is used, then:

$$\dot{\boldsymbol{q}}_R^W = \frac{1}{2} \cdot \boldsymbol{q}_R^W \otimes \bar{\boldsymbol{\omega}}_{R|W}^R \tag{C.4}$$

To find out the relationship between global and local angular velocity, equations C.3 and C.4 are combined, getting:

$$\bar{\boldsymbol{\omega}}_{R|W}^W = \boldsymbol{q}_R^W \otimes \bar{\boldsymbol{\omega}}_{R|W}^R \otimes \left(\boldsymbol{q}_R^W\right)^* \tag{C.5}$$

$$\bar{\boldsymbol{\omega}}_{R|W}^R = \left(\boldsymbol{q}_R^W\right)^* \otimes \bar{\boldsymbol{\omega}}_{R|W}^W \otimes \boldsymbol{q}_R^W \tag{C.6}$$

Angular acceleration

If a global rotation is used, deriving equation C.3, then:

$$\begin{aligned} \ddot{\boldsymbol{q}}_R^W &= \frac{1}{2} \cdot \dot{\bar{\boldsymbol{\omega}}}_{R|W}^W \otimes \boldsymbol{q}_R^W + \frac{1}{2} \cdot \bar{\boldsymbol{\omega}}_{R|W}^W \otimes \dot{\boldsymbol{q}}_R^W \\ &= \frac{1}{2} \cdot \bar{\boldsymbol{\alpha}}_{R|W}^W \otimes \boldsymbol{q}_R^W + \frac{1}{4} \cdot \bar{\boldsymbol{\omega}}_{R|W}^W \otimes \bar{\boldsymbol{\omega}}_{R|W}^W \otimes \boldsymbol{q}_R^W \end{aligned} \tag{C.7}$$

Being $\bar{\boldsymbol{\alpha}}_{R|W}^W = \dot{\bar{\boldsymbol{\omega}}}_{R|W}^W$ the angular acceleration of the robot with respect to world in world coordinates.

If a local rotation is used, deriving equation C.4, then:

$$\begin{aligned} \ddot{\boldsymbol{q}}_R^W &= \frac{1}{2} \cdot \dot{\boldsymbol{q}}_R^W \otimes \bar{\boldsymbol{\omega}}_{R|W}^R + \frac{1}{2} \cdot \boldsymbol{q}_R^W \otimes \dot{\bar{\boldsymbol{\omega}}}_{R|W}^R \\ &= \frac{1}{4} \cdot \boldsymbol{q}_R^W \otimes \bar{\boldsymbol{\omega}}_{R|W}^R \otimes \bar{\boldsymbol{\omega}}_{R|W}^R + \frac{1}{2} \cdot \boldsymbol{q}_R^W \otimes \bar{\boldsymbol{\alpha}}_{R|W}^R \end{aligned} \tag{C.8}$$

Being $\bar{\boldsymbol{\alpha}}_{R|W}^R = \dot{\bar{\boldsymbol{\omega}}}_{R|W}^R$ the angular acceleration of the robot with respect to world in robot coordinates.

To find out the relationship between global and local angular acceleration, equations C.7 and C.8 are combined, getting:

$$\bar{\boldsymbol{\alpha}}_{R|W}^W = \boldsymbol{q}_R^W \otimes \bar{\boldsymbol{\alpha}}_{R|W}^R \otimes \left(\boldsymbol{q}_R^W\right)^* \tag{C.9}$$

$$\bar{\boldsymbol{\alpha}}_{R|W}^R = \left(\boldsymbol{q}_R^W\right)^* \otimes \bar{\boldsymbol{\alpha}}_{R|W}^W \otimes \boldsymbol{q}_R^W \tag{C.10}$$

C.1.2. Using VRS

Pose

The pose of the robot in world coordinates, p_R^W, is represented with: $\boldsymbol{t}_R^W$ and $\boldsymbol{R}_R^W$.

Linear velocity

The linear velocity of the robot with respect to world in world coordinates is defined as: $\dot{\boldsymbol{t}}_R^W = \boldsymbol{v}_{R|W}^W$.
The linear velocity of the robot with respect to world in robot coordinates can be obtained with:

$$\boldsymbol{v}_{R|W}^R = \left(\boldsymbol{R}_R^W\right)^T \cdot \boldsymbol{v}_{R|W}^W \tag{C.11}$$

Linear acceleration

The linear acceleration of the robot with respect to world in world coordinates is defined as: $\ddot{\boldsymbol{t}}_R^W = \boldsymbol{a}_{R|W}^W$.
The linear acceleration of the robot with respect to world in robot coordinates can be obtained with:

$$\boldsymbol{a}_{R|W}^R = \left(\boldsymbol{R}_R^W\right)^T \cdot \boldsymbol{a}_{R|W}^W \tag{C.12}$$

Angular velocity

As explained in Appendix B.4, when operating with rotations, they can be considered as a local or as a global rotations.
Having for a global rotation:

$$\dot{\boldsymbol{R}}_R^W = \left[\boldsymbol{\omega}_{R|W}^W\right]_\times \cdot \boldsymbol{R}_R^W \tag{C.13}$$

Having for a local rotation:

$$\dot{\boldsymbol{R}}_R^W = \boldsymbol{R}_R^W \cdot \left[\boldsymbol{\omega}_{R|W}^R\right]_\times \tag{C.14}$$

Combining the previous equations, the relationship between global and local angular velocity can be obtained:

$$\boldsymbol{\omega}_{R|W}^W = \boldsymbol{R}_R^W \cdot \boldsymbol{\omega}_{R|W}^R \tag{C.15}$$

Angular acceleration

If a global rotation is used, deriving equation C.13, then:

$$\begin{aligned}\ddot{\boldsymbol{R}}_R^W =& \frac{\mathrm{d}}{\mathrm{d}t}\left(\left[\boldsymbol{\omega}_{R|W}^W\right]_\times\right) \cdot \boldsymbol{R}_R^W + \left[\boldsymbol{\omega}_{R|W}^W\right]_\times \cdot \dot{\boldsymbol{R}}_R^W \\ =& \left[\boldsymbol{\alpha}_{R|W}^W\right]_\times \cdot \boldsymbol{R}_R^W + \left[\boldsymbol{\omega}_{R|W}^W\right]_\times \cdot \left[\boldsymbol{\omega}_{R|W}^W\right]_\times \cdot \boldsymbol{R}_R^W\end{aligned} \tag{C.16}$$

Being $\bar{\boldsymbol{\alpha}}_{R|W}^W = \dot{\bar{\boldsymbol{\omega}}}_{R|W}^W$ the angular acceleration of the robot with respect to world in world coordinates.
If a global rotation is used, deriving equation C.14, then:

$$\begin{aligned}\ddot{\boldsymbol{R}}_R^W =& \dot{\boldsymbol{R}}_R^W \cdot \left[\boldsymbol{\omega}_{R|W}^R\right]_\times + \boldsymbol{R}_R^W \cdot \frac{\mathrm{d}}{\mathrm{d}t}\left(\left[\boldsymbol{\omega}_{R|W}^R\right]_\times\right) \\ =& \boldsymbol{R}_R^W \cdot \left[\boldsymbol{\omega}_{R|W}^R\right]_\times \cdot \left[\boldsymbol{\omega}_{R|W}^R\right]_\times + \boldsymbol{R}_R^W \cdot \left[\boldsymbol{\alpha}_{R|W}^R\right]_\times\end{aligned} \tag{C.17}$$

Being $\bar{\boldsymbol{\alpha}}_{R|W}^R = \dot{\bar{\boldsymbol{\omega}}}_{R|W}^R$ the angular acceleration of the robot with respect to world in robot coordinates.
To find out the relationship between global and local angular acceleration, equations C.16 and C.17 are combined, getting:

$$\boldsymbol{\alpha}_{R|W}^W = \boldsymbol{R}_R^W \cdot \boldsymbol{\alpha}_{R|W}^R \tag{C.18}$$

C.2. Kinematic equations of any point of a rigid body

In this section, the kinematic equations of any point of a rigid body are obtained, using two different formalisms to represent the pose. This point is a reference frame, that is, it has position and orientation.

Supposing to have the situation represented in Fig. C.2, where a rigid body is moving in a static world, whose inertial reference frame is labeled as world reference frame. The rigid body has a rigidly attached reference frame, called robot reference frame, that represents the center of interest of the robot (e.g. the reference frame used in control). The rigid body also have another rigidly attached reference frame, called sensor reference frame, that indicates the existence of a sensor that is able to perceive the state of the rigid body.

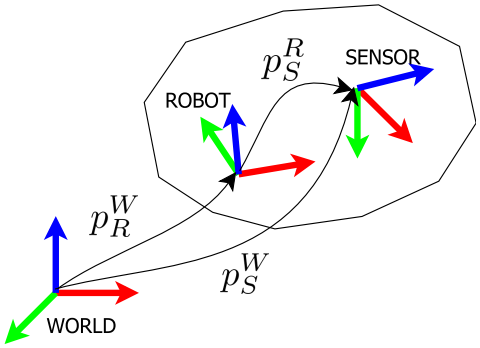


Figure C.2: Rigid body moving in an static world. The transformations between world, robot, and sensor reference frames are indicated.

Pose

The pose of the sensor in world coordinates is given by:

$$p_S^W = p_R^W \oplus p_S^R \tag{C.19}$$

It is important to highlight that p_S^R is constant, what means that it does not change over the time.

C.2.1. Using VQS

Pose

The pose of the sensor in world coordinates is obtained from equation C.19 as:

$$\vec{t}_S^W = \boldsymbol{q}_R^W \otimes \vec{t}_S^R \otimes \left(\boldsymbol{q}_R^W\right)^* + \vec{t}_R^W \tag{C.20}$$
$$\boldsymbol{q}_S^W = \boldsymbol{q}_R^W \otimes \boldsymbol{q}_S^R \tag{C.21}$$

Angular Velocity

Deriving equation C.21:

$$\dot{\boldsymbol{q}}_S^W = \frac{\mathrm{d}}{\mathrm{d}t}\left(\boldsymbol{q}_R^W \otimes \boldsymbol{q}_S^R\right) = \dot{\boldsymbol{q}}_R^W \otimes \boldsymbol{q}_S^R + \boldsymbol{q}_R^W \otimes \cancelto{0}{\dot{\boldsymbol{q}}_S^R} = \dot{\boldsymbol{q}}_R^W \otimes \boldsymbol{q}_S^R \tag{C.22}$$

If a global rotation is used, then:

$$\dot{\boldsymbol{q}}_S^W = \frac{1}{2} \cdot \bar{\omega}_{S|W}^W \otimes \boldsymbol{q}_S^W \tag{C.23}$$
$$\dot{\boldsymbol{q}}_S^W = \left(\frac{1}{2} \cdot \bar{\omega}_{R|W}^W \otimes \boldsymbol{q}_R^W\right) \otimes \boldsymbol{q}_S^R = \frac{1}{2} \cdot \bar{\omega}_{S|W}^W \otimes \boldsymbol{q}_S^W \tag{C.24}$$

And therefore:

$$\boldsymbol{\omega}_{S|W}^W = \boldsymbol{\omega}_{R|W}^W \tag{C.25}$$

If a local rotation is used, then:

$$\dot{\boldsymbol{q}}_S^W = \frac{1}{2} \cdot \boldsymbol{q}_S^W \otimes \bar{\omega}_{S|W}^S \tag{C.26}$$
$$\dot{\boldsymbol{q}}_S^W = \left(\frac{1}{2} \cdot \boldsymbol{q}_R^W \otimes \bar{\omega}_{R|W}^R\right) \otimes \boldsymbol{q}_S^R \tag{C.27}$$

And therefore:

$$\bar{\omega}_{S|W}^S = \left(\boldsymbol{q}_S^R\right)^* \otimes \bar{\omega}_{R|W}^R \otimes \boldsymbol{q}_S^R \tag{C.28}$$

Angular Acceleration

Deriving equation C.22:

$$\ddot{q}_S^W = \frac{\mathrm{d}}{\mathrm{d}t}\left(\dot{q}_R^W \otimes q_S^R\right) = \ddot{q}_R^W \otimes q_S^R + \dot{q}_R^W \otimes \cancelto{0}{\dot{q}_S^R} = \ddot{q}_R^W \otimes q_S^R \tag{C.29}$$

If a global rotation is used:

$$\ddot{q}_S^W = \frac{1}{2} \cdot \bar{\alpha}_{S|W}^W \otimes q_S^W + \frac{1}{4} \cdot \bar{\omega}_{S|W}^W \otimes \bar{\omega}_{S|W}^W \otimes q_S^W \tag{C.30}$$

$$\ddot{q}_S^W = \left(\frac{1}{2} \cdot \bar{\alpha}_{R|W}^W \otimes q_R^W + \frac{1}{4} \cdot \bar{\omega}_{R|W}^W \otimes \bar{\omega}_{R|W}^W \otimes q_R^W\right) \otimes q_S^R \tag{C.31}$$

And therefore:

$$\boldsymbol{\alpha}_{S|W}^W = \boldsymbol{\alpha}_{R|W}^W \tag{C.32}$$

If a local rotation is used:

$$\ddot{q}_S^W = \frac{1}{4} \cdot q_S^W \otimes \bar{\omega}_{S|W}^S \otimes \bar{\omega}_{S|W}^S + \frac{1}{2} \cdot q_S^W \otimes \bar{\alpha}_{S|W}^S \tag{C.33}$$

$$\ddot{q}_S^W = \left(\frac{1}{4} \cdot q_R^W \otimes \bar{\omega}_{R|W}^R \otimes \bar{\omega}_{R|W}^R + \frac{1}{2} \cdot q_R^W \otimes \bar{\alpha}_{R|W}^R\right) \otimes q_S^R \tag{C.34}$$

And therefore:

$$\bar{\alpha}_{S|W}^S = \left(q_S^R\right)^* \otimes \bar{\alpha}_{R|W}^R \otimes q_S^R \tag{C.35}$$

Linear Velocity

Deriving equation C.20:

$$\dot{\bar{t}}_S^W = \frac{\mathrm{d}}{\mathrm{d}t}\left(q_R^W \otimes \bar{t}_S^R \otimes \left(q_R^W\right)^* + \bar{t}_R^W\right) = \frac{\mathrm{d}}{\mathrm{d}t}\left(q_R^W \otimes \bar{t}_S^R \otimes \left(q_R^W\right)^*\right) + \dot{\bar{t}}_R^W \tag{C.36}$$

Being:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(q_R^W \otimes \bar{t}_S^R \otimes \left(q_R^W\right)^*\right) = \dot{q}_R^W \otimes \left(\bar{t}_S^R \otimes \left(q_R^W\right)^*\right) + q_R^W \otimes \frac{\mathrm{d}}{\mathrm{d}t}\left(\bar{t}_S^R \otimes \left(q_R^W\right)^*\right) \tag{C.37}$$

And being:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\bar{t}_S^R \otimes \left(q_R^W\right)^*\right) = \cancelto{0}{\dot{\bar{t}}_S^R} \otimes \left(q_R^W\right)^* + \bar{t}_S^R \otimes \left(\dot{q}_R^W\right)^* \tag{C.38}$$

So, therefore:

$$\dot{\bar{t}}_S^W = \dot{\bar{t}}_R^W + \dot{q}_R^W \otimes \bar{t}_S^R \otimes \left(q_R^W\right)^* + q_R^W \otimes \bar{t}_S^R \otimes \left(\dot{q}_R^W\right)^* \tag{C.39}$$

If a global rotation is used, then:

$$\begin{aligned}
\dot{\bar{t}}_S^W =& \dot{\bar{t}}_R^W + \left(\frac{1}{2} \cdot \bar{\omega}_{R|W}^W \otimes q_R^W\right) \otimes \bar{t}_S^R \otimes \left(q_R^W\right)^* + q_R^W \otimes \bar{t}_S^R \otimes \left(\frac{1}{2} \cdot \bar{\omega}_{R|W}^W \otimes q_R^W\right)^* \\
=& \dot{\bar{t}}_R^W + \frac{1}{2}\left(\bar{\omega}_{R|W}^W \otimes \left(q_R^W \otimes \bar{t}_S^R \otimes \left(q_R^W\right)^*\right) + \left(q_R^W \otimes \bar{t}_S^R \otimes \left(q_R^W\right)^*\right) \otimes \left(\bar{\omega}_{R|W}^W\right)^*\right) \\
=& \dot{\bar{t}}_R^W + \bar{\omega}_{R|W}^W \times \left(q_R^W \otimes \bar{t}_S^R \otimes \left(q_R^W\right)^*\right)
\end{aligned} \tag{C.40}$$

If a local rotation is used, then:

$$\begin{aligned}
\dot{\bar{t}}_S^W =& \dot{\bar{t}}_R^W + \left(\frac{1}{2} \cdot q_R^W \otimes \bar{\omega}_{R|W}^R\right) \otimes \bar{t}_S^R \otimes \left(q_R^W\right)^* + q_R^W \otimes \bar{t}_S^R \otimes \left(\frac{1}{2} \cdot q_R^W \otimes \bar{\omega}_{R|W}^R\right)^* \\
=& \dot{\bar{t}}_R^W + \frac{1}{2} \cdot q_R^W \otimes \left(\bar{\omega}_{R|W}^R \otimes \bar{t}_S^R + \bar{t}_S^R \otimes \left(\bar{\omega}_{R|W}^R\right)^*\right) \otimes \left(q_R^W\right)^* \\
=& \dot{\bar{t}}_R^W + q_R^W \otimes \left(\bar{\omega}_{R|W}^R \times \bar{t}_S^R\right) \otimes \left(q_R^W\right)^*
\end{aligned} \tag{C.41}$$

Linear Acceleration

Deriving equation C.39:

$$\begin{aligned}\ddot{\bar{t}}_S^W =& \frac{\mathrm{d}}{\mathrm{d}t}\left(\dot{\bar{t}}_R^W + \dot{\bar{q}}_R^W \otimes \bar{t}_S^R \otimes \left(\bar{q}_R^W\right)^* + \bar{q}_R^W \otimes \bar{t}_S^R \otimes \left(\dot{\bar{q}}_R^W\right)^*\right) \\ =& \ddot{\bar{t}}_R^W + \frac{\mathrm{d}}{\mathrm{d}t}\left(\dot{\bar{q}}_R^W \otimes \bar{t}_S^R \otimes \left(\bar{q}_R^W\right)^*\right) + \frac{\mathrm{d}}{\mathrm{d}t}\left(\bar{q}_R^W \otimes \bar{t}_S^R \otimes \left(\dot{\bar{q}}_R^W\right)^*\right)\end{aligned} \tag{C.42}$$

Being:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\left(\dot{\bar{q}}_R^W \otimes \bar{t}_S^R \otimes \left(\bar{q}_R^W\right)^*\right) =& \ddot{\bar{q}}_R^W \otimes \left(\bar{t}_S^R \otimes \left(\bar{q}_R^W\right)^*\right) + \dot{\bar{q}}_R^W \otimes \frac{\mathrm{d}}{\mathrm{d}t}\left(\bar{t}_S^R \otimes \left(\bar{q}_R^W\right)^*\right) \\ =& \ddot{\bar{q}}_R^W \otimes \bar{t}_S^R \otimes \left(\bar{q}_R^W\right)^* + \dot{\bar{q}}_R^W \otimes \bar{t}_S^R \otimes \left(\dot{\bar{q}}_R^W\right)^*\end{aligned} \tag{C.43}$$

And:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\left(\bar{q}_R^W \otimes \bar{t}_S^R \otimes \left(\dot{\bar{q}}_R^W\right)^*\right) =& \dot{\bar{q}}_R^W \otimes \left(\bar{t}_S^R \otimes \left(\dot{\bar{q}}_R^W\right)^*\right) + \bar{q}_R^W \otimes \frac{\mathrm{d}}{\mathrm{d}t}\left(\bar{t}_S^R \otimes \left(\dot{\bar{q}}_R^W\right)^*\right) \\ =& \dot{\bar{q}}_R^W \otimes \bar{t}_S^R \otimes \left(\dot{\bar{q}}_R^W\right)^* + \bar{q}_R^W \otimes \bar{t}_S^R \otimes \left(\ddot{\bar{q}}_R^W\right)^*\end{aligned} \tag{C.44}$$

So therefore:

$$\ddot{\bar{t}}_S^W = \ddot{\bar{t}}_R^W + \ddot{\bar{q}}_R^W \otimes \bar{t}_S^R \otimes \left(\bar{q}_R^W\right)^* + 2 \cdot \dot{\bar{q}}_R^W \otimes \bar{t}_S^R \otimes \left(\dot{\bar{q}}_R^W\right)^* + \bar{q}_R^W \otimes \bar{t}_S^R \otimes \left(\ddot{\bar{q}}_R^W\right)^* \tag{C.45}$$

If a global rotation is used, then, after some operations:

$$\ddot{\bar{t}}_S^W = \ddot{\bar{t}}_R^W + \bar{\omega}_{R|W}^W \times \bar{\omega}_{R|W}^W \times \left(\bar{q}_R^W \otimes \bar{t}_S^R \otimes \left(\bar{q}_R^W\right)^*\right) + \bar{\alpha}_{R|W}^R \times \left(\bar{q}_R^W \otimes \bar{t}_S^R \otimes \left(\bar{q}_R^W\right)^*\right) \tag{C.46}$$

If a local rotation is used, then, after some operations:

$$\ddot{\bar{t}}_S^W = \ddot{\bar{t}}_R^W + \bar{q}_R^W \otimes \left(\bar{\omega}_{R|W}^R \times \bar{\omega}_{R|W}^R \times \bar{t}_S^R\right) \otimes \left(\bar{q}_R^W\right)^* + \bar{q}_R^W \otimes \left(\bar{\alpha}_{R|W}^R \times \bar{t}_S^R\right) \otimes \left(\bar{q}_R^W\right)^* \tag{C.47}$$

C.2.2. Using VRS

Similarly as done in Appendix C.2.1, the kinematic equations of any point of a rigid body can be obtained using the VQS formalism.

Pose

The pose of the sensor in world coordinates is obtained from equation C.19 as:

$$\boldsymbol{t}_S^W = \boldsymbol{R}_R^W \cdot \boldsymbol{t}_S^R + \boldsymbol{t}_R^W \tag{C.48}$$

$$\boldsymbol{R}_S^W = \boldsymbol{R}_R^W \cdot \boldsymbol{R}_S^R \tag{C.49}$$

Angular Velocity

Deriving equation C.49:

$$\dot{\boldsymbol{R}}_S^W = \frac{\mathrm{d}}{\mathrm{d}t}\left(\boldsymbol{R}_R^W \cdot \boldsymbol{R}_S^R\right) = \dot{\boldsymbol{R}}_R^W \cdot \boldsymbol{R}_S^R + \boldsymbol{R}_R^W \cdot \cancelto{0}{\dot{\boldsymbol{R}}_S^R} = \dot{\boldsymbol{R}}_R^W \cdot \boldsymbol{R}_S^R \tag{C.50}$$

For a global rotation:

$$\dot{\boldsymbol{R}}_S^W = \left[\boldsymbol{\omega}_{S|W}^W\right]_\times \cdot \boldsymbol{R}_S^W \tag{C.51}$$

$$\dot{\boldsymbol{R}}_S^W = \left(\left[\boldsymbol{\omega}_{R|W}^W\right]_\times \cdot \boldsymbol{R}_R^W\right) \cdot \boldsymbol{R}_S^R \tag{C.52}$$

$$\tag{C.53}$$

So therefore:

$$\left[\boldsymbol{\omega}_{S|W}^W\right]_\times = \left[\boldsymbol{\omega}_{R|W}^W\right]_\times \tag{C.54}$$

And therefore;

$$\boldsymbol{\omega}_{S|W}^{W} = \boldsymbol{\omega}_{R|W}^{W} \tag{C.55}$$

For a local rotation:

$$\dot{\boldsymbol{R}}_{S}^{W} = \boldsymbol{R}_{S}^{W} \cdot \left[\boldsymbol{\omega}_{S|W}^{S}\right]_{\times} \tag{C.56}$$

$$\dot{\boldsymbol{R}}_{S}^{W} = \left(\boldsymbol{R}_{R}^{W} \cdot \left[\boldsymbol{\omega}_{R|W}^{R}\right]_{\times}\right) \cdot \boldsymbol{R}_{S}^{R} \tag{C.57}$$

$$\tag{C.58}$$

So therefore:

$$\left[\boldsymbol{\omega}_{S|W}^{S}\right]_{\times} = \left(\boldsymbol{R}_{S}^{R}\right)^{T} \cdot \left[\boldsymbol{\omega}_{R|W}^{R}\right]_{\times} \cdot \boldsymbol{R}_{S}^{R} = \left[\left(\boldsymbol{R}_{S}^{R}\right)^{T} \cdot \boldsymbol{\omega}_{R|W}^{R}\right]_{\times} \tag{C.59}$$

And therefore:

$$\boldsymbol{\omega}_{S|W}^{S} = \left(\boldsymbol{R}_{S}^{R}\right)^{T} \cdot \boldsymbol{\omega}_{R|W}^{R} \tag{C.60}$$

Angular Acceleration

Deriving equation C.50:

$$\ddot{\boldsymbol{R}}_{S}^{W} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\dot{\boldsymbol{R}}_{R}^{W} \cdot \boldsymbol{R}_{S}^{R}\right) = \ddot{\boldsymbol{R}}_{R}^{W} \cdot \boldsymbol{R}_{S}^{R} + \dot{\boldsymbol{R}}_{R}^{W} \cdot \cancelto{0}{\dot{\boldsymbol{R}}_{S}^{R}} = \ddot{\boldsymbol{R}}_{R}^{W} \cdot \boldsymbol{R}_{S}^{R} \tag{C.61}$$

For a global rotation, after some operations:

$$\boldsymbol{\alpha}_{S|W}^{W} = \boldsymbol{\alpha}_{R|W}^{W} \tag{C.62}$$

For a local rotation, after some operations:

$$\boldsymbol{\alpha}_{S|W}^{S} = \left(\boldsymbol{R}_{S}^{R}\right)^{T} \cdot \boldsymbol{\alpha}_{R|W}^{R} \tag{C.63}$$

Linear Velocity

Deriving equation C.48:

$$\dot{\boldsymbol{t}}_{S}^{W} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\boldsymbol{R}_{R}^{W} \cdot \boldsymbol{t}_{S}^{R} + \boldsymbol{t}_{R}^{W}\right) = \dot{\boldsymbol{R}}_{R}^{W} \cdot \boldsymbol{t}_{S}^{R} + \boldsymbol{R}_{R}^{W} \cdot \cancelto{0}{\dot{\boldsymbol{t}}_{S}^{R}} + \dot{\boldsymbol{t}}_{R}^{W} = \dot{\boldsymbol{t}}_{R}^{W} + \dot{\boldsymbol{R}}_{R}^{W} \cdot \boldsymbol{t}_{S}^{R} \tag{C.64}$$

For a global rotation, after some operations:

$$\dot{\boldsymbol{t}}_{S}^{W} = \dot{\boldsymbol{t}}_{R}^{W} + \left[\boldsymbol{\omega}_{R|W}^{W}\right]_{\times} \cdot \boldsymbol{R}_{R}^{W} \cdot \boldsymbol{t}_{S}^{R} \tag{C.65}$$

For a local rotation, after some operations:

$$\dot{\boldsymbol{t}}_{S}^{W} = \dot{\boldsymbol{t}}_{R}^{W} + \boldsymbol{R}_{R}^{W} \cdot \left[\boldsymbol{\omega}_{R|W}^{R}\right]_{\times} \cdot \boldsymbol{t}_{S}^{R} \tag{C.66}$$

Linear Acceleration

Deriving equation C.64:

$$\ddot{\boldsymbol{t}}_{S}^{W} = \frac{\mathrm{d}}{\mathrm{d}t}\left(\dot{\boldsymbol{t}}_{R}^{W} + \dot{\boldsymbol{R}}_{R}^{W} \cdot \boldsymbol{t}_{S}^{R}\right) = \ddot{\boldsymbol{t}}_{R}^{W} + \ddot{\boldsymbol{R}}_{R}^{W} \cdot \boldsymbol{t}_{S}^{R} + \dot{\boldsymbol{R}}_{R}^{W} \cdot \cancelto{0}{\dot{\boldsymbol{t}}_{S}^{R}} = \ddot{\boldsymbol{t}}_{R}^{W} + \ddot{\boldsymbol{R}}_{R}^{W} \cdot \boldsymbol{t}_{S}^{R} \tag{C.67}$$

For a global rotation, after some operations:

$$\ddot{\boldsymbol{t}}_{S}^{W} = \ddot{\boldsymbol{t}}_{R}^{W} + \left(\left[\boldsymbol{\alpha}_{R|W}^{W}\right]_{\times} \cdot \boldsymbol{R}_{R}^{W} + \left[\boldsymbol{\omega}_{R|W}^{W}\right]_{\times} \cdot \left[\boldsymbol{\omega}_{R|W}^{W}\right]_{\times} \cdot \boldsymbol{R}_{R}^{W}\right) \cdot \boldsymbol{t}_{S}^{R} \tag{C.68}$$

For a local rotation, after some operations:

$$\ddot{\boldsymbol{t}}_{S}^{W} = \ddot{\boldsymbol{t}}_{R}^{W} + \left(\boldsymbol{R}_{R}^{W} \cdot \left[\boldsymbol{\omega}_{R|W}^{R}\right]_{\times} \cdot \left[\boldsymbol{\omega}_{R|W}^{R}\right]_{\times} + \boldsymbol{R}_{R}^{W} \cdot \left[\boldsymbol{\alpha}_{R|W}^{R}\right]_{\times}\right) \cdot \boldsymbol{t}_{S}^{R} \tag{C.69}$$

Appendix D

Extended Kalman Filter Algorithm

D.1. Considerations

D.1.1. State vs. parameters

There might exist two different types of variables when working in state estimation. In the one hand, the state is a random variable which value changes over the time. In the other hand, parameters are random variables which values do not change over the time.

Due to the use of an extended Kalman filter algorithm, a normal distribution is assumed to represent both the state and the parameters. In the case that the covariance of the normal distribution that represent a parameter is zero, then the parameter is assumed to be deterministic.

The use of stochastic parameters and deterministic parameters allows incorporating uncertainty to some of the parameters without having the need of including them on the state. Thanks to this, the state estimation will be more accurate than considering all the parameters as deterministic.

D.1.2. Generic addition operation

In the following sections, a generic addition operation, $\oplus$, and its inverse operation, $\ominus$, has been used to compute the innovation of the measurement, $\boldsymbol{\nu}$, and to update the state.

D.2. EKF-SLAM

The extended Kalman filter algorithm used for simultaneous localization and mapping (EKF-SLAM) is a very well-known algorithm that can be found in (Terejanu, 2008; Sola, 2013). This section briefly reviews it and augments it with the use of parameters.

D.2.1. Problem statement

State, parameters and input commands

State $\boldsymbol{x}$ with dimension $n \times 1$. The state is modeled as a random variable that changes with the time. It is following a multi-variate normal distribution with an associated matrix of variances and covariances $\boldsymbol{P}$ with

dimension $n \times n$. The state is divided in:

- State of the robot $\boldsymbol{x}_R$ with dimension $n_R \times 1$.
- State of the map elements (landmarks) $\boldsymbol{x}_L$. There are $\mathring{n}_L$ map elements (landmarks) on the map. Every landmark i has a state $\boldsymbol{x}_{L_i}$ with a dimension n_{L_i}. The total dimension of the landmarks is $n_L = \sum n_{L_i}$

Input commands $\boldsymbol{u}$ with dimension $p \times 1$. The input commands are modeled as random variables that change with the time. They are following a multi-variate normal distribution with an associated matrix of variances and covariances $\boldsymbol{Q}_u$ with dimension $p \times p$

Parameters $\boldsymbol{\mu}$ with dimension $n_\mu \times 1$. The parameters are modeled as random variables that do not change with the time, modeled as a multi-variate normal distribution with an associated matrix of variances and covariances $\boldsymbol{Q}_\mu$ with dimension $n_\mu \times n_\mu$.

Discrete-time process model

The discrete time process model also called state model, is used to calculate the evolution of the state, depending on the current state, the input commands, and the parameters. It incorporates a random noise $\boldsymbol{n}$, modeled as a Gaussian noise.

The discrete time process model is given by:

$$\boldsymbol{x}(k) = \boldsymbol{f}(\boldsymbol{x}(k-1), \boldsymbol{u}(k-1), \boldsymbol{\mu}, \boldsymbol{n}_f) \tag{D.1}$$

The Jacobian matrices of the process model are given by:

$$\boldsymbol{F}_{\boldsymbol{x}} = \left[\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}(k-1)}\right] \tag{D.2}$$

$$\boldsymbol{F}_{\boldsymbol{u}} = \left[\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}(k-1)}\right] \tag{D.3}$$

$$\boldsymbol{F}_{\boldsymbol{\mu}} = \left[\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{\mu}}\right] \tag{D.4}$$

$$\boldsymbol{F}_{\boldsymbol{n}_f} = \left[\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{n}_f}\right] \tag{D.5}$$

Measurements

The observation $\boldsymbol{z}$ has dimension $m \times 1$. It is modeled as a random variable following a multi-variate normal distribution with an associated matrix of variances and covariances $\boldsymbol{R}$.

Measurement model

The observation model, also called measurement model, allows calculating the value of the measurements given the state and the parameters. It also incorporates a random noise $\boldsymbol{n}$, modeled as a Gaussian noise

The measurement model is given by:

$$\boldsymbol{z}(k) = \boldsymbol{h}(\boldsymbol{x}(k), \boldsymbol{\mu}, \boldsymbol{n}_h) \tag{D.6}$$

And its Jacobian matrices are:

$$\boldsymbol{H}_{\boldsymbol{x}} = \left[\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\right] \tag{D.7}$$

$$\boldsymbol{H}_{\boldsymbol{\mu}} = \left[\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{\mu}}\right] \tag{D.8}$$

$$\boldsymbol{H}_{\boldsymbol{n}_h} = \left[\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{n}_h}\right] \tag{D.9}$$

If the measurements are independent, then, the measurement model can be divided into several measurement models:

$$\boldsymbol{z}_i(k) = \boldsymbol{h}_i(\boldsymbol{x}(k), \boldsymbol{\mu}, \boldsymbol{n}) \tag{D.10}$$

Mapping model

Assuming that the state of the new map element is fully observable, that is, the measurement model $\boldsymbol{h}$ is invertible $\boldsymbol{g}_n = (\boldsymbol{h})^{-1}$, the state of the new map element is given by:

$$\boldsymbol{x}_n(k) = \boldsymbol{g}_n(\boldsymbol{x}(k), \boldsymbol{\mu}, \boldsymbol{z}(k)) \tag{D.11}$$

The mapping model, that augments the dimension of the state is given by:

$$\boldsymbol{x}^a(k) = \boldsymbol{g}(\boldsymbol{x}(k), \boldsymbol{\mu}, \boldsymbol{z}(k)) \tag{D.12}$$

And its Jacobian matrices are:

$$\boldsymbol{G}_{\boldsymbol{x}} = \left[\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}\right] \tag{D.13}$$

$$\boldsymbol{G}_{\boldsymbol{\mu}} = \left[\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{\mu}}\right] \tag{D.14}$$

$$\boldsymbol{G}_{\boldsymbol{z}} = \left[\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}}\right] \tag{D.15}$$

It is important to note that when a new map element is mapped, its value, $\boldsymbol{x}_n$ is added to the state, and therefore, the state is augmented:

$$\boldsymbol{x}^a = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{x}_n \end{bmatrix} \tag{D.16}$$

Note that non-invertible mapping models are not considered.

D.2.2. Prediction Stage

Steps:

1. Prediction of the state:
$$\hat{\boldsymbol{x}}(k|k-1) = \boldsymbol{f}(\hat{\boldsymbol{x}}(k-1|k-1), \boldsymbol{u}(k-1), \boldsymbol{\mu}) \tag{D.17}$$
2. Jacobian matrices of the process model: $\boldsymbol{F}_{\boldsymbol{x}}$, $\boldsymbol{F}_{\boldsymbol{u}}$ $\boldsymbol{F}_{\boldsymbol{\mu}}$ and $\boldsymbol{F}_{\boldsymbol{n}_f}$
3. Covariance of the predicted state $\hat{\boldsymbol{x}}(k|k-1)$:
$$\boldsymbol{P}(k|k-1) = \boldsymbol{F}_{\boldsymbol{x}} \cdot \boldsymbol{P}(k-1|k-1) \cdot \boldsymbol{F}_{\boldsymbol{x}}^T + \boldsymbol{F}_{\boldsymbol{u}} \cdot \boldsymbol{Q}_u \cdot \boldsymbol{F}_{\boldsymbol{u}}^T + \boldsymbol{F}_{\boldsymbol{\mu}} \cdot \boldsymbol{Q}_{\mu} \cdot \boldsymbol{F}_{\boldsymbol{\mu}}^T + \boldsymbol{F}_{\boldsymbol{n}_f} \cdot \boldsymbol{Q}_n \cdot \boldsymbol{F}_{\boldsymbol{n}_f}^T \tag{D.18}$$

   Note that:
$$\boldsymbol{Q}_{\boldsymbol{n}} = (\boldsymbol{Q}_{\boldsymbol{n}})_c \cdot \Delta t \tag{D.19}$$

   Being $(\boldsymbol{Q}_{\boldsymbol{n}})_c$ the continuous-time covariance matrix of the noise $\boldsymbol{n}_{f_c}$ associated to the continuous-time process model (see (Sola, 2016)).

D.2.3. Update Stage

It is executed after the prediction stage when a measurement, $\tilde{\boldsymbol{z}}$, arrives.

Measurement matching of associated measurements

The measurements, $\tilde{\boldsymbol{z}}$, and the predicted measurements, $\hat{\boldsymbol{z}}$, can be associated without any effort thanks to an id or similar.

Steps:

1. Measurement prediction:
$$\hat{\boldsymbol{z}}(k) = \boldsymbol{h}(\hat{\boldsymbol{x}}(k|k-1), \boldsymbol{\mu}) \tag{D.20}$$
2. Innovation of the measurement:
$$\boldsymbol{\nu}(k) = \tilde{\boldsymbol{z}}(k) \ominus \hat{\boldsymbol{z}}(k) \tag{D.21}$$
3. Jacobian matrices of the measurement model: $\boldsymbol{H}_{\boldsymbol{x}}$, $\boldsymbol{H}_{\boldsymbol{\mu}}$ and $\boldsymbol{H}_{\boldsymbol{n}_h}$.
4. Covariance of the innovation of the measurement:
$$\boldsymbol{S}(k) = \boldsymbol{H}_{\boldsymbol{x}} \cdot \boldsymbol{P}(k+1|k) \cdot \boldsymbol{H}_{\boldsymbol{x}}^T + \boldsymbol{H}_{\boldsymbol{\mu}} \cdot \boldsymbol{Q}_{\mu} \cdot \boldsymbol{H}_{\boldsymbol{\mu}}^T + \boldsymbol{H}_{\boldsymbol{n}_h} \cdot \boldsymbol{R}_n \cdot \boldsymbol{H}_{\boldsymbol{n}_h}^T \tag{D.22}$$

5. Individual Mahalanobis distance test on the innovation of every measurement:

$$d_{S_i}^2 = \boldsymbol{\nu}_{S_i}(k) \cdot (\boldsymbol{S}_{S_i})^{-1}(k) \cdot \boldsymbol{\nu}_{S_i}(k) \tag{D.23}$$

   The test is passed with a $(1.0-\alpha)\%$ accuracy if $d_{S_i}^2 \leq \chi_{m_i,\alpha}^2$.
6. Collective Mahalanobis distance test on the innovation of the measurement:

$$d^2 = \boldsymbol{\nu}^T(k) \cdot (\boldsymbol{S}(k))^{-1}(k) \cdot \boldsymbol{\nu}_\delta(k) \tag{D.24}$$

   The test is passed with a $(1.0-\alpha)\%$ accuracy if $d^2 \leq \chi_{m,\alpha}^2$.

Measurement matching of unassociated measurements

The measurements, $\tilde{\boldsymbol{z}}$, and the predicted measurements, $\hat{\boldsymbol{z}}$, are not associated and need to be matched. There are $\mathring{m}_u$ unassociated measurements, and $\mathring{m}$ independent measurement models.

For every non associated measurement $\forall j \in \mathring{m}_u$, and for every independent measurement model $\forall i \in \mathring{m}$, the following steps are done:

1. Prediction of the measurement i:

$$\hat{\boldsymbol{z}}_i(k) = \boldsymbol{h}_i(\hat{\boldsymbol{x}}(k|k-1), \boldsymbol{\mu}) \tag{D.25}$$

2. Jacobian matrices of the predicted measurement i: $\boldsymbol{H}_{i\boldsymbol{x}}$, $\boldsymbol{H}_{i\boldsymbol{\mu}}$ and $\boldsymbol{H}_{i\boldsymbol{n}_h}$
3. Association between the predicted measurement i and the unmatched measurement j based on the smallest Mahalanobis distance:
   The measurement innovation of the predicted measurement i with respect to the unmatched measurement j is given by:

$$\boldsymbol{\nu}_{ij}(k) = \hat{\boldsymbol{z}}_i(k) \ominus \tilde{\boldsymbol{z}}_j(k) \tag{D.26}$$

   Which associated matrix of variances and covariances $\boldsymbol{S}_{ij}(k)$ is the following:

$$\boldsymbol{S}_{ij}(k) = \boldsymbol{H}_{i\boldsymbol{x}} \cdot \boldsymbol{P}(k|k-1) \cdot \boldsymbol{H}_{i\boldsymbol{x}}^T + \boldsymbol{H}_{i\boldsymbol{\mu}} \cdot \boldsymbol{Q}_\mu \cdot \boldsymbol{H}_{i\boldsymbol{\mu}}^T + \boldsymbol{H}_{i\boldsymbol{n}_h}(k) \cdot \boldsymbol{R}_j(k) \cdot \boldsymbol{H}_{i\boldsymbol{n}_h}^T(k) \tag{D.27}$$

   The Mahalanobis distance between measurement i and estimated measurement j is:

$$d_{ij} = \sqrt{\boldsymbol{\nu}_{ij}^T(k) \cdot (\boldsymbol{S}_{ij}(k))^{-1} \cdot \boldsymbol{\nu}_{ij}(k)} \tag{D.28}$$

   The smallest d_{ij} produces an association candidate between landmark i and measurement j.
4. Individual Mahalanobis distance test on the innovation of every association: The test is passed with a $(1.0-\alpha)\%$ accuracy if $d_{ij}^2 \leq \chi_{m_i,\alpha}^2$, the estimated measurement i and the measurement j are associated.
5. Collective Mahalanobis distance test on the innovation of the measurement:

$$d^2 = \boldsymbol{\nu}^T(k) \cdot (\boldsymbol{S}(k))^{-1}(k) \cdot \boldsymbol{\nu}_\delta(k) \tag{D.29}$$

   The test is passed with a $(1.0-\alpha)\%$ accuracy if $d^2 \leq \chi_{m,\alpha}^2$.

State update

It is executed taking into account all the matched measurements if their Mahalanobis distance test has been passed.

Steps:

1. Kalman Gain:

$$\boldsymbol{K} = \boldsymbol{P}(k|k-1) \cdot \boldsymbol{H}_{\boldsymbol{x}}^T \cdot (\boldsymbol{S}(k))^{-1} \tag{D.30}$$

2. State increment:

$$\Delta\boldsymbol{x} = \boldsymbol{K} \cdot \boldsymbol{\nu} \tag{D.31}$$

3. State update:

$$\hat{\boldsymbol{x}}(k|k) = \hat{\boldsymbol{x}}(k|k-1) \oplus \Delta\boldsymbol{x} \tag{D.32}$$

4. Covariance of the updated state (two calculation options):

$$\boldsymbol{P}(k|k) = (\boldsymbol{I} - \boldsymbol{K} \cdot \boldsymbol{H}_{\boldsymbol{x}}) \cdot \boldsymbol{P}(k|k-1) \tag{D.33}$$

$$\boldsymbol{P}(k|k) = (\boldsymbol{I} - \boldsymbol{K} \cdot \boldsymbol{H}_{\boldsymbol{x}}) \cdot \boldsymbol{P}(k|k-1) \cdot (\boldsymbol{I} - \boldsymbol{K} \cdot \boldsymbol{H}_{\boldsymbol{x}})^T + \boldsymbol{K} \cdot \boldsymbol{R}_n \cdot \boldsymbol{K}^T \tag{D.34}$$

Mapping

It is executed taking into account all the unmatched measurements, $\mathring{m}_u$. The state is augmented thanks to the unmatched measurements.

Steps:

1. State update:
$$\hat{\boldsymbol{x}}^a(k|k) = \boldsymbol{g}(\hat{\boldsymbol{x}}(k|k), \boldsymbol{\mu}, \tilde{\boldsymbol{z}}(k)) \tag{D.35}$$
2. Jacobian matrices of the mapping model: $\boldsymbol{G}_{\boldsymbol{x}}$, $\boldsymbol{G}_{\boldsymbol{\mu}}$ and $\boldsymbol{G}_{\boldsymbol{z}}$
3. Covariance of the updated state:
$$\boldsymbol{P}^a(k|k) = \boldsymbol{G}_{\boldsymbol{x}} \cdot \boldsymbol{P}(k|k) \cdot \boldsymbol{G}_{\boldsymbol{x}}^T + \boldsymbol{G}_{\boldsymbol{\mu}} \cdot \boldsymbol{R}_{\mu-u} \cdot \boldsymbol{G}_{\boldsymbol{\mu}}^T + \boldsymbol{G}_{\boldsymbol{z}} \cdot \boldsymbol{R}_{z-u} \cdot \boldsymbol{G}_{\boldsymbol{z}}^T \tag{D.36}$$

D.3. Error-value EKF-SLAM

The error-value extended Kalman filter algorithm is a very well-known algorithm described in (Sola, 2016). This section explains the error-value extended Kalman filter algorithm used for simultaneous localization and mapping (EKF-SLAM), augmented with the use of parameters.

D.3.1. Error-value formulation

In error-value formulations, there are differences between: (1) true-value ($\boldsymbol{\upsilon}_t$), (2) nominal-value ($\boldsymbol{\upsilon}$), and (3) error-value ($\boldsymbol{\delta\upsilon}$). The true-value is expressed as a composition of the nominal-value and the error-value (Local perturbation: $\boldsymbol{\upsilon}_t = \boldsymbol{\upsilon} \oplus \boldsymbol{\delta\upsilon}$, global perturbation: $\boldsymbol{\upsilon}_t = \boldsymbol{\delta\upsilon} \oplus \boldsymbol{\upsilon}$). The idea is to consider the nominal-value as large-signal (integrable in a non-linear fashion) and the error-value as a small signal (linearly integrable and suitable for linear-Gaussian filtering).

During the prediction stages, the nominal-state is updated by integration. This nominal-state does not take into account noisy terms and other possible model imperfections. As a consequence, it will accumulate errors. These errors are collected in the error-state and estimated with the error-state Kalman filter (ESKF), this time incorporating all the noise and perturbations. The error-state consists of small-signal magnitudes, and its evolution function is correctly defined by a (time-variant) linear dynamic system, which its dynamic, control and measurement matrices are computed from the values of the nominal-state. In parallel with the integration of the nominal-state, the ESKF predicts a Gaussian estimate of the error-state.

The update stage is performed when a measurement arrives, which is able to render the errors observable and which happens generally at a much lower rate than the prediction phase. This correction provides a posterior Gaussian estimate of the error-state. After this, the error-state's mean is injected into the nominal-state, then reset to zero. The error-state's covariances matrix is conveniently updated to reflect this reset.

D.3.2. Problem statement

State, parameters and input commands

True-state $\boldsymbol{x}_t$ (stochastic) with dimension $n_{ts} \times 1$, modeled as a time-variant random variable following a multi-variate normal distribution with an associated matrix of variances and covariances $\boldsymbol{P}_{tx}$ with dimension $n_{ts} \times n_{ts}$, divided into:

- Nominal-state $\boldsymbol{x}$ (deterministic) with dimension $n_s \times 1$
- Error-state $\boldsymbol{\delta x}$ (stochastic) with dimension $n_{\delta s} \times 1$, and an associated matrix of variances and covariances $\boldsymbol{P}_{\delta x}$ with dimension $n_{\delta s} \times n_{\delta s}$

True-parameters $\boldsymbol{\mu}_t$ (stochastic) with dimension $n_{tp} \times 1$, modeled as a time-invariant random variable following a multi-variate normal distribution with an associated matrix of variances and covariances $\boldsymbol{Q}_{t\mu}$ with dimension $n_{tp} \times n_{tp}$, divided in:

- Nominal-parameters $\boldsymbol{\mu}$ (deterministic) with dimension $n_p \times 1$
- Error-parameters $\boldsymbol{\delta\mu}$ (stochastic) with dimension $n_{\delta p} \times 1$, and an associated matrix of variances and covariances $\boldsymbol{Q}_{\delta\mu}$ with dimension $n_{\delta p} \times n_{\delta p}$

True-commands $\boldsymbol{u}_t$ (stochastic) with dimension $n_{tc} \times 1$, modeled as a time-variant random variable following a multi-variate normal distribution with an associated matrix of variances and covariances $\boldsymbol{Q}_{tu}$ with dimension $n_{tc} \times n_{tc}$, divided in:

- Nominal-commands $\boldsymbol{u}$ (deterministic) with dimension $n_c \times 1$
- Error-commands $\boldsymbol{\delta u}$ (stochastic) with dimension $n_{\delta c} \times 1$, and an associated matrix of variances and covariances $\boldsymbol{Q}_{\delta u}$ with dimension $n_{\delta c} \times n_{\delta c}$

Process Model

Continuous-time process model

The continuous-time process model is divided into:

- True-value process model:

$$\dot{\boldsymbol{x}}_t(t) = \boldsymbol{f}_t(\boldsymbol{x}_t(t), \boldsymbol{u}_t(t), \boldsymbol{\mu}_t, \boldsymbol{n}_{tf}) \tag{D.37}$$

  Being the noise in the process model: $\boldsymbol{n}_{tf}$ with dimension n_{tf}.
- Nominal-value process model: modeled system without noises or perturbation:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{\mu}) \tag{D.38}$$

- Error-value process model: dynamics of the error-state

$$\dot{\delta\boldsymbol{x}}(t) = \boldsymbol{f}_\delta(\boldsymbol{x}(t), \delta\boldsymbol{x}(t), \boldsymbol{u}(t), \delta\boldsymbol{u}(t), \boldsymbol{\mu}, \delta\boldsymbol{\mu}, \boldsymbol{n}_{\delta f}) \tag{D.39}$$

  Being the noise in the process model: $\boldsymbol{n}_{\delta f}$ with dimension $n_{\delta f}$. After this step, the model is linearized.

Discrete-time process model

The discrete-time process model represents the system kinematics in discrete time, that is the integration of the continuous-time kinematics for $\Delta t > 0$. In some cases, it is possible to use exact closed-form solutions. In other cases, numerical integration of varying degree of accuracy may be employed.

The discrete-time process model is divided into:

- True-value process model:

$$\boldsymbol{x}_t(k) = \boldsymbol{f}_t\left(\boldsymbol{x}_t(k-1), \boldsymbol{u}_t(k-1), \boldsymbol{\mu}_t, \boldsymbol{n}_{tf}\right) \tag{D.40}$$

- Nominal-value process model:

$$\boldsymbol{x}(k) = \boldsymbol{f}\left(\boldsymbol{x}(k-1), \boldsymbol{u}(k-1), \boldsymbol{\mu}\right) \tag{D.41}$$

- Error-value process model:

$$\delta\boldsymbol{x}(k) = \boldsymbol{f}_\delta\left(\boldsymbol{x}(k-1), \delta\boldsymbol{x}(k-1), \boldsymbol{u}(k-1), \delta\boldsymbol{u}(k-1), \boldsymbol{\mu}, \delta\boldsymbol{\mu}, \boldsymbol{n}_{\delta f}\right) \tag{D.42}$$

  The error-value process model Jacobian matrices are given by:

$$\boldsymbol{F}_{\delta\boldsymbol{x}} = \left[\frac{\partial \boldsymbol{f}_\delta}{\partial \delta\boldsymbol{x}}\right]_{n_{\delta s} \times n_{\delta s}} \tag{D.43}$$

$$\boldsymbol{F}_{\delta\boldsymbol{u}} = \left[\frac{\partial \boldsymbol{f}_\delta}{\partial \delta\boldsymbol{u}}\right]_{n_{\delta s} \times n_{\delta c}} \tag{D.44}$$

$$\boldsymbol{F}_{\delta\boldsymbol{\mu}} = \left[\frac{\partial \boldsymbol{f}_\delta}{\partial \delta\boldsymbol{\mu}}\right]_{n_{\delta s} \times n_{\delta p}} \tag{D.45}$$

$$\boldsymbol{F}_{\boldsymbol{n}_{\delta f}} = \left[\frac{\partial \boldsymbol{f}_\delta}{\partial \boldsymbol{n}_{\delta f}}\right]_{n_{\delta s} \times n_{\delta f}} \tag{D.46}$$

  The linearized error-value process model is given by:

$$\delta\boldsymbol{x}(k) = \boldsymbol{F}_{\delta\boldsymbol{x}} \cdot \delta\boldsymbol{x}(k-1) + \boldsymbol{F}_{\delta\boldsymbol{u}} \cdot \delta\boldsymbol{u}(k-1) + \boldsymbol{F}_{\delta\boldsymbol{\mu}} \cdot \delta\boldsymbol{\mu} + \boldsymbol{F}_{\boldsymbol{n}_{\delta f}} \cdot \boldsymbol{n}_{\delta f} \tag{D.47}$$

Measurements

True-measurements $\boldsymbol{z}_t$ (stochastic) has dimension $n_{tm} \times 1$, modeled as a random variable following a multi-variate normal distribution and has an associated matrix of variances and covariances $\boldsymbol{R}_t$, divided into:

- Nominal-measurements $\boldsymbol{z}$ (deterministic) has dimension $n_m \times 1$.
- Error-measurements $\delta\boldsymbol{z}$ (stochastic) has dimension $n_{\delta m} \times 1$ and has an associated matrix of variances and covariances $\boldsymbol{R}_\delta$

There are $\mathring{m}_z$ measurements. All the measurements are assumed to be independent, so, therefore, the total associated matrix of variances and covariances is a block diagonal matrix with $\mathring{m}_z$ blocks.

Measurement model

The observation model, also called measurement model, allows to calculate the value of the measurements given the state and the parameters, and it is divided into:

- True-value observation model:

$$\boldsymbol{z}_t(k) = \boldsymbol{h}_t\left(\boldsymbol{x}_t(k), \boldsymbol{\mu}_t, \boldsymbol{n}_{tz}\right) \tag{D.48}$$

- Nominal-value observation model:

$$\boldsymbol{z}(k) = \boldsymbol{h}\left(\boldsymbol{x}(k), \boldsymbol{\mu}\right) \tag{D.49}$$

- Error-value observation model:

$$\boldsymbol{\delta z}(k) = \boldsymbol{h}_{\delta}\left(\boldsymbol{x}(k), \boldsymbol{\delta x}(k), \boldsymbol{\mu}, \boldsymbol{\delta \mu}, \boldsymbol{n}_{\delta z}\right) \tag{D.50}$$

The error-value observation model Jacobian matrices are given by:

$$\boldsymbol{H}_{\delta \boldsymbol{x}} = \left[\frac{\partial \boldsymbol{h}_{\delta}}{\partial \boldsymbol{\delta x}}\right]_{n_{\delta m} \times n_{\delta s}} \tag{D.51}$$

$$\boldsymbol{H}_{\delta \boldsymbol{\mu}} = \left[\frac{\partial \boldsymbol{h}_{\delta}}{\partial \boldsymbol{\delta \mu}}\right]_{n_{\delta m} \times n_{\delta p}} \tag{D.52}$$

$$\boldsymbol{H}_{\boldsymbol{n}_{\delta z}} = \left[\frac{\partial \boldsymbol{h}_{\delta}}{\partial \boldsymbol{n}_{\delta z}}\right]_{n_{\delta m} \times n_{\delta z}} \tag{D.53}$$

Mapping model

Assuming that the state of the new map element is fully observable, that is, the measurement model $\boldsymbol{h}$ is invertible $\boldsymbol{g}_n = (\boldsymbol{h})^{-1}$, the state of the new map element is given by:

$$\boldsymbol{x}_n(k) = \boldsymbol{g}_n\left(\boldsymbol{x}(k), \boldsymbol{\mu}, \boldsymbol{z}(k)\right) \tag{D.54}$$

The mapping model, that augments the dimension of the state is divided into:

- True-value mapping model:

$$\boldsymbol{x}^a{}_t(k) = \boldsymbol{g}_t\left(\boldsymbol{x}_t(k), \boldsymbol{\mu}_t(k), \boldsymbol{z}_t(k)\right) \tag{D.55}$$

- Nominal-value mapping model:

$$\boldsymbol{x}^a(k) = \boldsymbol{g}\left(\boldsymbol{x}(k), \boldsymbol{\mu}(k), \boldsymbol{z}(k)\right) \tag{D.56}$$

- Error-value mapping model:

$$\boldsymbol{\delta x}^a(k) = \boldsymbol{g}_{\delta}\left(\boldsymbol{x}(k), \boldsymbol{\delta x}(k), \boldsymbol{\mu}(k), \boldsymbol{\delta \mu}(k), \boldsymbol{z}(k), \boldsymbol{\delta z}(k)\right) \tag{D.57}$$

The error-value mapping model Jacobian matrices are given by:

$$\boldsymbol{G}_{\delta \boldsymbol{x}} = \left[\frac{\partial \boldsymbol{g}_{\delta}}{\partial \boldsymbol{\delta x}}\right]_{n_{\delta s'} \times n_{\delta s}} \tag{D.58}$$

$$\boldsymbol{G}_{\delta \boldsymbol{\mu}} = \left[\frac{\partial \boldsymbol{g}_{\delta}}{\partial \boldsymbol{\delta \mu}}\right]_{n_{\delta s'} \times n_{\delta p}} \tag{D.59}$$

$$\boldsymbol{G}_{\delta \boldsymbol{z}} = \left[\frac{\partial \boldsymbol{g}_{\delta}}{\partial \boldsymbol{\delta z}}\right]_{n_{\delta s'} \times n_{\delta z}} \tag{D.60}$$

It is important to note that when a new map element is mapped, its value, $\boldsymbol{x}_n$ is added to the state, and therefore, the state is augmented:

$$\boldsymbol{x}^a = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{x}_n \end{bmatrix} \tag{D.61}$$

Note that non-invertible mapping models are not considered.

D.3.3. Prediction Stage

Steps:

1. Prediction of the nominal-state (deterministic part):

$$\hat{\boldsymbol{x}}(k|k-1) = \boldsymbol{f}_n(\hat{\boldsymbol{x}}(k-1|k-1), \boldsymbol{u}(k-1), \boldsymbol{\mu}) \tag{D.62}$$

2. Prediction of the error-state (deterministic part):

$$\begin{aligned}\delta\hat{\boldsymbol{x}}(k|k-1) =& \boldsymbol{f}_\delta(\hat{\boldsymbol{x}}(k-1|k-1), \delta\hat{\boldsymbol{x}}(k-1|k-1), \boldsymbol{u}(k-1), \delta\boldsymbol{u}(k-1), \boldsymbol{\mu}, \delta\boldsymbol{\mu}) \\ =& \boldsymbol{F}_{\delta\boldsymbol{x}} \cdot \cancelto{0}{\delta\hat{\boldsymbol{x}}(k-1|k-1)} + \boldsymbol{F}_{\delta\boldsymbol{u}} \cdot \cancelto{0}{\delta\boldsymbol{u}(k-1)} + \boldsymbol{F}_{\delta\boldsymbol{\mu}} \cdot \cancelto{0}{\delta\boldsymbol{\mu}} \\ =& \boldsymbol{0}\end{aligned} \tag{D.63}$$

3. Jacobian matrices of the error-value process model evaluated on the nominal-state: $\boldsymbol{F}_{\delta\boldsymbol{x}}$, $\boldsymbol{F}_{\delta\boldsymbol{u}}$, $\boldsymbol{F}_{\delta\boldsymbol{\mu}}$ and $\boldsymbol{F}_{\boldsymbol{n}_{\delta f}}$
4. Covariance of the error-state $\delta\hat{\boldsymbol{x}}(k|k-1)$:

$$\begin{aligned}\boldsymbol{P}_{\delta\boldsymbol{x}}(k|k-1) =& \boldsymbol{F}_{\delta\boldsymbol{x}} \cdot \boldsymbol{P}_{\delta\boldsymbol{x}}(k-1|k-1) \cdot \boldsymbol{F}_{\delta\boldsymbol{x}}^T + \boldsymbol{F}_{\delta\boldsymbol{u}} \cdot \boldsymbol{Q}_{\delta\boldsymbol{u}} \cdot \boldsymbol{F}_{\delta\boldsymbol{u}}^T + \boldsymbol{F}_{\delta\boldsymbol{\mu}} \cdot \boldsymbol{Q}_{\delta\boldsymbol{\mu}} \cdot \boldsymbol{F}_{\delta\boldsymbol{\mu}}^T \\ &+ \boldsymbol{F}_{\boldsymbol{n}_{\delta f}} \cdot \boldsymbol{Q}_{\boldsymbol{n}_{\delta f}} \cdot \boldsymbol{F}_{\boldsymbol{n}_{\delta f}}^T\end{aligned} \tag{D.64}$$

Note that:

$$\boldsymbol{Q}_{\boldsymbol{n}_{\delta f}} = \left(\boldsymbol{Q}_{\boldsymbol{n}_{\delta f}}\right)_c \cdot \Delta t \tag{D.65}$$

Being $\left(\boldsymbol{Q}_{\boldsymbol{n}_{\delta f}}\right)_c$ the continuous-time covariance matrix of the noise $\boldsymbol{n}_{\delta f}$ associated to the continuous-time process model (see (Sola, 2016)).

D.3.4. Update Stage

It is executed after the prediction stage when a measurement, $\tilde{\boldsymbol{z}}$, arrives.

Measurement matching of associated measurements

The measurements, $\tilde{\boldsymbol{z}}$, and the predicted measurements, $\hat{\boldsymbol{z}}$, can be associated without any effort thanks to an id or similar.

Steps:

1. Measurement prediction (nominal):

$$\hat{\boldsymbol{z}}(k) = \boldsymbol{h}(\hat{\boldsymbol{x}}(k|k-1), \boldsymbol{\mu}) \tag{D.66}$$

2. Innovation of the measurement (local perturbation error-value measurement):

$$\boldsymbol{\nu}_\delta(k) = \ominus\hat{\boldsymbol{z}}(k) \oplus \tilde{\boldsymbol{z}}(k) = \delta\boldsymbol{z}(k) \tag{D.67}$$

Note that: $\tilde{\boldsymbol{z}}(k)$ is $\boldsymbol{z}_t$

3. Jacobian matrices of the error-value measurement model: $\boldsymbol{H}_{\delta\boldsymbol{x}}$, $\boldsymbol{H}_{\delta\boldsymbol{\mu}}$ and $\boldsymbol{H}_{\boldsymbol{n}_{\delta\boldsymbol{z}}}$
4. Covariance of the innovation of the error-value measurement:

$$\boldsymbol{S}_\delta(k) = \boldsymbol{H}_{\delta\boldsymbol{x}} \cdot \boldsymbol{P}_{\delta\boldsymbol{x}}(k|k-1) \cdot \boldsymbol{H}_{\delta\boldsymbol{x}}^T + \boldsymbol{H}_{\delta\boldsymbol{\mu}} \cdot \boldsymbol{Q}_{\delta\mu} \cdot \boldsymbol{H}_{\delta\boldsymbol{\mu}}^T + \boldsymbol{H}_{\boldsymbol{n}_{\delta\boldsymbol{z}}} \cdot \boldsymbol{R}_{\delta\boldsymbol{z}-m} \cdot \boldsymbol{H}_{\boldsymbol{n}_{\delta\boldsymbol{z}}}^T \tag{D.68}$$

5. Individual Mahalanobis distance test on the innovation of every measurement:

$$d_{S_i}^2 = \boldsymbol{\nu}_{S_i}^T(k) \cdot (\boldsymbol{S}_{S_i})^{-1}(k) \cdot \boldsymbol{\nu}_{S_i}(k) \tag{D.69}$$

The test is passed with a $(1.0-\alpha)\%$ accuracy if $d_{S_i}^2 \leq \chi_{m_i,\alpha}^2$.

6. Collective Mahalanobis distance test on the innovation of the measurement:

$$d^2 = \boldsymbol{\nu}_\delta^T(k) \cdot (\boldsymbol{S}_\delta)^{-1}(k) \cdot \boldsymbol{\nu}_\delta(k) \tag{D.70}$$

The test is passed with a $(1.0-\alpha)\%$ accuracy if $d^2 \leq \chi_{m,\alpha}^2$

Measurement matching of unassociated measurements

The measurements, $\tilde{\boldsymbol{z}}$, and the predicted measurements, $\hat{\boldsymbol{z}}$, are not associated and need to be matched. There are $\mathring{m}_u$ unassociated measurements, and $\mathring{m}$ independent measurement models.

It is done in an equivalent way than in Appendix D.2, and it is omitted here.

State update

It is executed taking into account all the matched measurements, if their Mahalanobis distance test has been passed.

Update of the error-state

Steps:

1. Kalman Gain:
$$\boldsymbol{K} = \boldsymbol{P}_{\delta\boldsymbol{x}}(k|k-1) \cdot \boldsymbol{H}_{\delta x}^T \cdot \boldsymbol{S}_\delta(k-1)^{-1} \tag{D.71}$$
2. Increment of the error-state:
$$\Delta\boldsymbol{\delta x}(k) = \boldsymbol{K} \cdot \boldsymbol{\nu}_\delta(k) \tag{D.72}$$
3. Update of the deterministic part of the error-state:
$$\boldsymbol{\delta\hat{x}}(k|k) = \cancelto{\boldsymbol{0}}{\boldsymbol{\delta\hat{x}}(k|k-1)} \oplus \Delta\boldsymbol{\delta x}(k) = \Delta\boldsymbol{\delta x}(k) \tag{D.73}$$
4. Update of the covariance of the error-state (two calculation options):
$$\boldsymbol{P}_{\delta\boldsymbol{x}}^-(k|k) = (\boldsymbol{I} - \boldsymbol{K} \cdot \boldsymbol{H}_{\delta x}) \cdot \boldsymbol{P}_{\delta\boldsymbol{x}}(k|k-1) \tag{D.74}$$
$$\boldsymbol{P}_{\delta\boldsymbol{x}}^-(k|k) = (\boldsymbol{I} - \boldsymbol{K} \cdot \boldsymbol{H}_{\boldsymbol{n}_{\delta\boldsymbol{z}}}) \cdot \boldsymbol{P}_{\delta\boldsymbol{x}}(k|k-1) \cdot (\boldsymbol{I} - \boldsymbol{K} \cdot \boldsymbol{H}_{\boldsymbol{n}_{\delta\boldsymbol{z}}})^T + \boldsymbol{K} \cdot \boldsymbol{R}_{n-m} \cdot \boldsymbol{K}^T \tag{D.75}$$

Update of the nominal-state and reset of the error-state

The operation error-reset that needs to be done is based on the fact that the true-state does not change on error-reset:

$$\begin{aligned} \boldsymbol{x}_t^+ &= \boldsymbol{x}_t^- \\ \boldsymbol{x}^+ \oplus \boldsymbol{\delta x}^+ &= \boldsymbol{x}^- \oplus \boldsymbol{\delta x}^- \end{aligned}$$

Steps:

1. Injection of the deterministic part of error-state into the nominal-state:
Getting the nominal values and imposing that the deterministic part of the error-state after the error reset is zero (only the deterministic part satisfies: $\boldsymbol{\delta x}^+ = 0$):
$$\hat{\boldsymbol{x}}(k|k) = \hat{\boldsymbol{x}}(k|k-1) \oplus \boldsymbol{\delta\hat{x}}(k|k) \tag{D.76}$$
2. Reset of the deterministic part of the error-state (because it has been injected in the nominal-state):
The operation that needs to be done is:
$$\boldsymbol{\delta x}^+ = \boldsymbol{\delta x}^- \ominus \boldsymbol{\delta\hat{x}}^-$$
So the deterministic part of the error-state results:
$$\boldsymbol{\delta\hat{x}}^+(k|k) = \boldsymbol{0} \tag{D.77}$$
3. Jacobian matrix of the error-state reset:
The error-state after the error-reset is:
$$\boldsymbol{\delta x}^+ = \ominus\boldsymbol{x}^+ \oplus \boldsymbol{x}^- \oplus \boldsymbol{\delta x}^-$$
Therefore:
$$\boldsymbol{G}_{\delta\boldsymbol{x}^-} = \frac{\partial\boldsymbol{\delta x}^+}{\partial\boldsymbol{\delta x}^-} = \frac{\partial}{\partial\boldsymbol{\delta x}^-}\left(\ominus\boldsymbol{x}^+ \oplus \boldsymbol{x}^- \oplus \boldsymbol{\delta x}^-\right) = \frac{\partial}{\partial\boldsymbol{\delta x}^-}\left(\ominus\hat{\boldsymbol{x}}(k|k) \oplus \hat{\boldsymbol{x}}(k|k-1) \oplus \boldsymbol{\delta x}^-\right) \tag{D.78}$$
4. Reset of the covariance of the error-state:
$$\boldsymbol{P}_{\delta\boldsymbol{x}}(k|k) = \boldsymbol{G}_{\delta\boldsymbol{x}^-} \cdot \boldsymbol{P}_{\delta\boldsymbol{x}}^-(k|k) \cdot \boldsymbol{G}_{\delta\boldsymbol{x}^-}^T \tag{D.79}$$

Mapping

It is executed taking into account all the unmatched measurements, $\mathring{m}_u$. The state is augmented thanks to the unmatched measurements.

Steps:

1. Update of the nominal-state (the dimension will grow):

$$\hat{\boldsymbol{x}}^a(k|k) = \boldsymbol{g}(\hat{\boldsymbol{x}}(k|k), \boldsymbol{\mu}, \tilde{\boldsymbol{z}}(k)) \tag{D.80}$$

2. Update of the deterministic part of the error-state (the dimension will grow):

$$\delta\hat{\boldsymbol{x}}^a(k|k) = \mathbf{0} \tag{D.81}$$

3. Jacobian matrices of the error-state mapping model evaluated on the nominal state: $\boldsymbol{G}_{\delta \boldsymbol{x}}$, $\boldsymbol{G}_{\delta \boldsymbol{\mu}}$ and $\boldsymbol{G}_{\delta \boldsymbol{z}}$
4. Update of the covariance of the error-state (the dimension will grow):

$$\boldsymbol{P}_{\delta \boldsymbol{x}}{}^a(k|k) = \boldsymbol{G}_{\delta \boldsymbol{x}} \cdot \boldsymbol{P}_{\delta \boldsymbol{x}}(k|k) \cdot \boldsymbol{G}_{\delta \boldsymbol{x}}^T + \boldsymbol{G}_{\delta \boldsymbol{\mu}} \cdot \boldsymbol{Q}_{\delta \boldsymbol{\mu}} \cdot \boldsymbol{G}_{\delta \boldsymbol{\mu}}^T + \boldsymbol{G}_{\delta \boldsymbol{z}} \cdot \boldsymbol{R}_{\delta \boldsymbol{z}-u} \cdot \boldsymbol{G}_{\delta \boldsymbol{z}}^T \tag{D.82}$$

Appendix E

Scientific Dissemination

E.1. Publications

This section includes a list of publications related to this thesis, organized in three categories: related publications in journals indexed in the JCR (Appendix E.1.1), related publications in peer-reviewed international conferences (Appendix E.1.2), and other remarkable publications by the author of this thesis (Appendix E.1.3 and Appendix E.1.4).

E.1.1. Publications in journals indexed in the JCR related to this thesis

Below there is a list of the publications derived and related to this thesis in journals indexed in the 2016 Journal Citation Report (JCR) of the Web of Science (WOS):

Accepted. **J. L. Sanchez-Lopez**, M. Molina, H. Bavle, C. Sampedro, R. A. Suárez Fernández, P. Campoy. *A Multi-Layered Component-Based Approach for the Development of Aerial Robotic Systems: The Aerostack Framework*. Journal of Intelligent & Robotic Systems.

J. L. Sanchez-Lopez, J. Pestana, P. de la Puente, P. Campoy. *A Reliable Open-Source System Architecture for the Fast Designing and Prototyping of Autonomous Multi-UAV Systems: Simulation and Experimentation*. Journal of Intelligent & Robotic Systems. Springer Netherlands. December 2016 (Online: Oct. 2015). Vol. 84. No. 1. Pp. 779-797. Online ISSN: 0921-0296. Print ISNN: 1573-0409. DOI: 10.1007/s10846-015-0288-x.

J. Pestana, **J. L. Sanchez-Lopez**, P. de la Puente, A. Carrio, P. Campoy. *A Vision-based Quadrotor Multi-Robot Solution for the Indoors Autonomy Challenge of the 2013 International Micro Air Vehicle Competition*. Journal of Intelligent & Robotic Systems. Springer Netherlands. December 2016 (Online: Nov. 2015). Vol. 84. No. 1. Pp. 601-620. Online ISSN: 0921-0296. Print ISNN: 1573-0409. DOI: 10.1007/s10846-015-0304-1.

J. L. Sanchez-Lopez, J. Pestana, S. Saripalli, P. Campoy. *An Approach Towards Visual Autonomous ship board landing of a VTOL UAV*. Journal of Intelligent & Robotic Systems. Springer Netherlands. April 2014 (Online: October 2013). Vol. 74. No. 1. Pp. 113-127. Online ISSN: 1573-0409. Print ISSN: 0921-0296. DOI:

10.1007/s10846-013-9926-3.

J. Pestana, I. Mellado-Bataller, C. Fu, **J. L. Sanchez-Lopez**, I. F. Mondragon, P. Campoy. *A General Purpose Configurable Controller for Indoors and Outdoors GPS-Denied Navigation for Multirotor Unmanned Aerial Vehicles*. Journal of Intelligent & Robotic Systems. Springer Netherlands. January 2014 (Online: October 2013). Vol. 73. No. 1. Pp. 387-400. Online ISSN: 1573-0409. Print ISSN: 0921-0296. DOI: 10.1007/s10846-013-9953-0.

E.1.2. Publications in peer-reviewed conferences related to this thesis (selection)

Below there is a selection of publications derived and related to this thesis in proceedings of peer-reviewed international conferences:

Accepted. **J. L. Sanchez-Lopez**, J. Pestana, P. Campoy. *A Robust Real Time Path Planner for the Collision Free Navigation of Multirotor Aerial Robots in Dynamic Environments*. 2017 International Conference on Unmanned Aircraft Systems (ICUAS).

Accepted. H. Bavle, **J. L. Sanchez-Lopez**, A. Rodríguez Ramos, C. Sampedro, P. Campoy. *Multi-Sensor Fusion for Estimating the Flight Altitude of Multirotor UAVs in Dynamic and Unstructured Indoor Environments*. 2017 International Conference on Unmanned Aircraft Systems (ICUAS).

Accepted. C. Sampedro, H. Bavle, A. Rodríguez Ramos, A. Carrio, R. A. Suárez Fernández, **J. L. Sanchez-Lopez**, P. Campoy. *A Fully-Autonomous UAV for Participating in the 2016 International Micro Air Vehicle Competition*. 2017 International Conference on Unmanned Aircraft Systems (ICUAS).

Accepted. **J. L. Sanchez-Lopez**, V. Arellano-Quintana, M. Tognon, P. Campoy, A. Franchi. *Visual Marker based Multi-Sensor Fusion State Estimation*. 2017 World Congress of the International Federation of Automatic Control (IFAC), Toulouse, France, Jul. 9-14, 2017.

J. L. Sanchez-Lopez, R. A. Suárez Fernández, H. Bavle, C. Sampedro, M. Molina, J. Pestana, P. Campoy. *AEROSTACK: An architecture and open-source software framework for aerial robotics*. 2016 International Conference on Unmanned Aircraft Systems (ICUAS), Arlington, VA, USA, Jun. 7-10, 2016, Pp. 332-341. DOI: 10.1109/ICUAS.2016.7502591.

R. A. Suárez Fernández, **J. L. Sanchez-Lopez**, C. Sampedro, H. Bavle, M. Molina, P. Campoy; *Natural user interfaces for human-drone multi-modal interaction*. 2016 International Conference on Unmanned Aircraft Systems (ICUAS), Arlington, VA, USA, Jun. 7-10, 2016, Pp. 1013-1022. DOI: 10.1109/ICUAS.2016.7502665.

C. Sampedro, H. Bavle, **J. L. Sanchez-Lopez**, R. A Suárez Fernández, A. Rodriguez-Ramos, M. Molina, P. Campoy. *A flexible and dynamic mission planning architecture for UAV swarm coordination*. 2016 International Conference on Unmanned Aircraft Systems (ICUAS), Arlington, VA, USA, Jun. 7-10, 2016, Pp. 355-363. DOI: 10.1109/ICUAS.2016.7502669.

J. L. Sanchez-Lopez, C. Fu, P. Campoy. *FuSeOn: a Low-cost Portable Multi Sensor Fusion Research Testbed for Robotics*. Robot 2015: Second Iberian Robotics Conference (ROBOT 2015). Lisbon (Portugal). Nov. 12-13, 2015. Book title: Robot 2015: Second Iberian Robotics Conference. Book subtitle: Advances in Robotics. Vol. 1. ISBN: 978-3-319-27146-0. DOI: 10.1007/978-3-319-27146-0_5.

J. L. Sanchez-Lopez, J. Pestana, J.-F. Collumeau, R. Suárez-Fernández, P. Campoy, M. Molina. *A Vision Based Aerial Robot solution for the Mission 7 of the International Aerial Robotics Competition*. Unmanned Aircraft Systems (ICUAS), 2015 International Conference on. Denver, (CO, USA). 9-12 June 2015. Pp: 1391 - 1400. Print ISBN: 978-1-4799-6009-5. DOI: 10.1109/ICUAS.2015.7152435.

J. Pestana, **J. L. Sanchez-Lopez**, S. Saripalli, P. Campoy. *Computer Vision Based General Object Following for GPS-denied Multirotor Unmanned Vehicles*. 2014 American Control Conference (ACC 14). Portland (OR, USA). Jun. 4-6, 2014. Pp: 1886-1891. ISSN: 0743-1619. DOI: 10.1109/ACC.2014.6858831.

J. L. Sanchez-Lopez, J. Pestana, P. de la Puente, R. Suárez-Fernández, P. Campoy. *A System for the Design and Development of Vision-based Multi-robot Quadrotor Swarms*. 2014 International Conference on Unmanned Aircraft Systems (ICUAS 2014). Orlando (FL, USA). May 27-30, 2014. Pp: 640-648. DOI: 10.1109/ICUAS.2014.6842308.

J. Pestana, **J. L. Sanchez-Lope**z, P. de la Puente, A. Carrio, P. Campoy. *A Vision-based Quadrotor Swarm for the participation in the 2013 International Micro Air Vehicle Competition*. 2014 International Conference on Unmanned Aircraft Systems (ICUAS 2014). Orlando (FL, USA). May 27-30, 2014. Pp: 617-622. DOI: 10.1109/ICUAS.2014.6842305.

J. L. Sanchez-Lopez, J. Pestana, P. de la Puente, A. Carrio, P. Campoy. *Visual Quadrotor Swarm for the IMAV 2013 Indoor Competition*. Robot 2013: First Iberian Robotics Conference (ROBOT 2013). Madrid (Spain). Nov. 28-29, 2013. Book title: Robot 2013: First Iberian Robotics Conference. Book subtitle: Advances in intelligent Systems and Computing. Vol. 253. Published year: 2014. Pp: 55 - 63. Print ISBN: 978-3-319-03652-6. Online ISBN: 978-3-319-03653-3. Publisher: Springer International Publishing. DOI: 10.1007/978-3-319-03653-3_5.

J. Pestana, I. Mellado-Bataller, **J. L. Sanchez-Lopez**, C. Fu, I. F. Mondragon, P. Campoy. *Floor Optical Flow Based Navigation Controller for Multirotor Aerial Vehicles*. Robot 2013: First Iberian Robotics Conference (ROBOT 2013). Madrid (Spain). Nov. 28-29, 2013. Book title: Robot 2013: First Iberian Robotics Conference. Book subtitle: Advances in intelligent Systems and Computing. Vol. 253. Published year: 2014. Pp: 91 - 106. Print ISBN: 978-3-319-03652-6. Online ISBN: 978-3-319-03653-3. Publisher: Springer International Publishing. DOI: 10.1007/978-3-319-03653-3_8.

J. Pestana, **J. L. Sanchez-Lopez**, S. Saripalli, P. Campoy. *Vision based GPS-denied Object tracking and Following for Unmanned Aerial Vehicles*. The 2013 (11th) IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR 2013). Linköping (Sweden). Oct 21-26, 2013. Best Paper Award Finalist. Pages: 1-6. Print ISBN: 978-1-4799-0879-0. DOI: 10.1109/SSRR.2013.6719359.

J. L. Sanchez-Lopez, S. Saripalli, P. Campoy, J. Pestana, C. Fu. *Toward Visual Autonomous Ship Board Landing of a VTOL UAV*. 2013 International Conference on Unmanned Aircraft Systems (ICUAS 2013). Atlanta, Georgia (USA). May 28-31, 2013. Print ISBN: 978-1-4799-0815-8. Publisher: IEEE. Pages: 779 - 788. DOI: 10.1109/ICUAS.2013.6564760.

J. Pestana, I. Mellado-Bataller, C. Fu, **J. L. Sanchez-Lopez**, I. Fernando Mondragon, P. Campoy. *A General Purpose Congurable Navigation Controller for Micro Aerial Multirotor Vehicles*. 2013 International Conference on Unmanned Aircraft Systems (ICUAS 2013). Atlanta, Georgia (USA). May 28-31, 2013. Print ISBN: 978-1-4799-0815-8. Publisher: IEEE. Pages: 557 - 564. DOI: 10.1109/ICUAS.2013.6564733.

E.1.3. Other non-related publications in journals indexed in the JCR

Below there is a selection of remarkable publications in journals indexed in the 2016 Journal Citation Report (JCR) of the Web of Science (WOS), that are not directly related to this thesis, but that are published by the author of this thesis:

B. Gonzalez-Rodrigo, R. Tendero-Caballero, M. Garcia-De-Viedma, J. Pestana-Puerta, A. Carrio-Fernandez, **J. L. Sanchez-Lopez**, R. A. Suarez Fernandez, P. Campoy, J. Bonatti-Gonzalez, J. Gregorio Rejas-Ayuga, R. Martinez-Marin, M. Marchamalo-Sacristan. *Thermal monitoring of facades by UAV: applications for building rehabilitation*. DYNA DYNA-ACELERADO. September 2016. Vol. 91. No. 5. Pp. 571-577. DOI: 10.6036/7899.

C. Cuerno Rejado, L. Garcia Hernandez, A. Sanchez Carmona, A. Carrio Fernandez, **J. L. Sanchez-Lopez**, P. Campoy Cervera. *Historical evolution of the unmanned aerial vehicles to the present*. DYNA DYNA-ACELERADO. May 2016. Vol. 91. No. 3. Pp. 282-288. DOI: 10.6036/7781.

M. A. Olivares-Mendez, **J. L. Sanchez-Lopez**, F. Jimenez, P. Campoy, S. A. Sajadi-Alamdari, H. Voos. *Vision-Based Steering Control, Speed Assistance and Localization for Inner-Cities Vehicles*. Sensors 2016.

March 2016. Vol. 16. No. 3. Pp. 362-394; ISSN:1424-8220. DOI: 10.3390/s16030362.

A. Carrio, C. Sampedro, **J. L. Sanchez-Lopez**, M. Pimienta, P. Campoy. *Automated low-cost smartphone-based lateral flow saliva test reader for drug-of-abuse detection*. Sensors 2015. November 2015. Vol. 15, No. 11. Pp. 29569-29593. ISSN: 29569-29593. DOI: 10.3390/s151129569.

C. Martinez, P. Campoy, I. F. Mondragon, **J. L. Sanchez-Lopez**, M. A. Olivares-Mendez. *HMPMR Strategy for Real-Time Tracking in Aerial Images, using Direct Methods*. Journal of Machine Vision and Applications. Springer Berlin Heidelberg. July 2014. Vol. 25, No. 5, Pp. 1283-1308. Online ISSN: 1432-1769. Print ISSN: 0932-8092. DOI: 10.1007/s00138-014-0617-2.

C. Martinez, I. Mondragon, P. Campoy, **J. L. Sanchez-Lopez**, M. A. Olivares-Mendez. *A Hierarchical Tracking Strategy for Vision-Based Applications On-board UAVs*. Journal of Intelligent & Robotic Systems. Springer Netherlands. December 2013 (Online: March 2013). Vol. 72. No. 3, Pp. 517-539. Online ISSN: 1573-0409. Print ISSN: 0921-0296. DOI: 10.1007/s10846-013-9814-x.

L. Hernández Hernández, J. Pestana, D. Casares Palomeque, P. Campoy, **J. L. Sanchez-Lopez**. *Identificación y control en cascada mediante inversión de no linealidades del cuatrirrotor para el Concurso de Ingeniería de Control CEA IFAC 2012*. Revista Iberoamericana de Automática e Informática Industrial RIAI. Elsevier. July-September 2013. Vol. 10. No. 3. Pp. 356-367. Online ISSN: 1697-7920. Print ISSN: 1697-7912. DOI: 10.1016/j.riai.2013.05.008.

E.1.4. Other non-related publications in peer-reviewed conferences (selection)

Below there is a selection of remarkable publications in proceedings of peer-reviewed international conferences that are not directly related to this thesis, but that are published by the author of this thesis:

A. Carrio, J. Pestana, **J. L. Sanchez-Lopez**, R. Suárez-Fernández, P. Campoy, R. Tendero, M. Garcia-De-Viedma, B. Gonzalez-Rodrigo, J. Bonatti, J. Gregorio Rejas-Ayuga, R. Martinez-Marin, Miguel Marchamalo-Sacristan. *UBRISTES: UAV-based building rehabilitation with visible and thermal infrared remote sensing*. Robot 2015: Second Iberian Robotics Conference (ROBOT 2015). Lisbon (Portugal). Nov. 12-13, 2015. Book title: Robot 2015: Second Iberian Robotics Conference. Book subtitle: Advances in Robotics. Vol. 1. ISBN: 978-3-319-27146-0. DOI: 10.1007/978-3-319-27146-0_19.

M. A. Olivares-Mendez, P. Campoy, I. Mellado-Bataller, I. F. Mondragon, C. Martinez, **J. L. Sanchez-Lopez**. *Autonomous Guided Car Using a Fuzzy Controller*. Recent Advances in Robotics and Automation. 2013. Series Vol. 480. Pp. 37-55. Publisher: Springer Berlin Heidelberg. Series Title: Studies in Computational Intelligence. Series ISSN: 1860-949X. Print ISBN: 978-3-642-37386-2. Online ISBN: 978-3-642-37387-9. DOI: 10.1007/978-3-642-37387-9_3.

C. Martinez, P. Campoy, I. Mondragon, **J. L. Sanchez-Lopez**, M. A. Olivares-Mendez. *A Hierarchical Strategy for Real-Time Tracking On-board UAVs*. 2012 International Conference on Unmanned Aircraft Systems (ICUAS 2012). Philadelphia (USA). Jun. 12-15, 2012.

E.2. Awards

This section includes all the awards (group and individual) received by the author of this thesis related to the work of his Ph.D. thesis:

E.2.1. Group awards

- IARC 2014 - Mission 7
  - Best System Control Award
  - Best Target Detection Award
- IMAV 2013 Competition - Indoors Autonomy
  - First place in the category Indoor Autonomy
- CEA 2012 Contest on Control Engineering

    - Autonomous path following of a quadrotor. Second (phase 1) and Fifth (final phase) places
- IMAV 2012 Indoor Flight Dynamics Competition - Rotory Wing MAV
    - Second place in the category Indoor Flight Dynamics - Rotory Wing MAV
    - Special Award in the category Best Automatic Performance - IMAV 2012

E.2.2. Individual awards

- Oct. 2016 - Present: UPM approved Fellowship: Project SALINE. Computer Vision and Aerial Robotics (CVAR) of the Centre for Automation and Robotics (CAR), UPM-CSIC. Advisor: Prof. Pascual Campoy Cervera
- Nov. 2015 - Sep. 2016: Doctoral Eiffel Excellence Scholarship by Campus France, 2015. Robotics and Interactions (RIS) of the LAAS-CNRS. Advisors: Prof. Simon Lacroix and Prof. Antonio Franchi
- Nov. 2011 - Oct. 2015: Pre-doctoral JAE Fellowship, 2011. Computer Vision Group (CVG) of the Centre for Automation and Robotics (CAR), UPM-CSIC. Advisor: Prof. Pascual Campoy Cervera
- May. 2011 - Oct. 2011: UPM approved Fellowship. Computer Vision Group (CVG) of the Centre for Automation and Robotics (CAR), UPM-CSIC. Advisor: Prof. Pascual Campoy Cervera

E.3. Open-source software

All the software generated by a consequence of this thesis has made open-source under different licenses (BSD or GNU) with two objectives: first of all to share it with the scientific community so they can take advantage of it on their future researches. Second, as a way to allow the scientific community to evaluate the developed research work proving what is stated in the associated publications.

Mostly all the software has been centralized on the Aerostack software framework:

- Aerostack: `https://github.com/Vision4UAV/Aerostack`
- Aerostack installer: `https://github.com/Vision4UAV/Aerostack_installer`
- Aerostack wiki: `https://github.com/Vision4UAV/Aerostack/wiki`

Besides, all the software repositories can be found on GitHub and Bitbucket:

- Vision4UAV:
    - `https://github.com/Vision4UAV`
    - `https://bitbucket.org/Vision4UAV`
- Thesis author:
    - `https://github.com/joselusl`
    - `https://bitbucket.org/joselusl`

E.4. Digital media

All the videos associated to this thesis have been published in the following video-sharing websites:

- Youtube Vision4UAV channel:
    - `https://www.youtube.com/channel/UCfry6dL6G5Sd3U49LC9wvYQ`
- Youtube thesis author channel:
    - `https://www.youtube.com/channel/UCmthQitewuA6r1lZ9Lxj-WQ`
- Youtube thesis list:
    - `https://www.youtube.com/playlist?list=PLPhn3zIScyQd_OhprXAgtf-1Wvrr-QuUR`

Bibliography

Achtelik, M., Achtelik, M., Weiss, S., and Siegwart, R. (2011). Onboard imu and monocular vision based control for mavs in unknown in- and outdoor environments. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3056–3063.

Al-Sabban, W. H., Gonzalez, L. F., and Smith, R. N. (2013). Wind-energy based path planning for unmanned aerial vehicles using markov decision processes. In *2013 IEEE International Conference on Robotics and Automation*, pages 784–789.

Alvarez-Santos, V., Iglesias, R., Pardo, X. M., Regueiro, C. V., and Canedo-Rodriguez, A. (2014). Gesture-based interaction with voice feedback for a tour-guide robot. *J. Vis. Comun. Image Represent.*, 25(2):499–509.

Arantes, M. d. S., Arantes, J. d. S., Toledo, C. F. M., and Williams, B. C. (2016). A hybrid multi-population genetic algorithm for uav path planning. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 853–860, New York, NY, USA. ACM.

Arkin, R. C., Riseman, E. M., and Hanson, A. R. (1987). Aura: An architecture for vision-based robot navigation. In *proceedings of the DARPA Image Understanding Workshop*, pages 417–431.

Bahill, A. T. and Gissing, B. (1998). Re-evaluating systems engineering concepts using systems thinking. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 28(4):516–527.

Bancroft, J. B. (2009). Multiple imu integration for vehicular navigation. In *Proceedings of ION GNSS*, volume 1, pages 1–13. Citeseer.

Bar-Itzhack, I. Y. and Oshman, Y. (1985). Attitude determination from vector observations: Quaternion estimation. *IEEE Transactions on Aerospace and Electronic Systems*, AES-21(1):128–136.

Barfoot, T., Forbes, J. R., and Furgale, P. T. (2011). Pose estimation using linearized rotations and quaternion algebra. *Acta Astronautica*, 68(1-2):101–112.

Barrientos, A., Peñín, L. F., Balaguer, C., and Aracil, R. (1997). *Fundamentos de robótica*, volume 256. McGraw-Hill.

Baudel, T. and Beaudouin-Lafon, M. (1993). Charade: Remote control of objects using free-hand gestures. *Commun. ACM*, 36(7):28–35.

Beer, J., Fisk, A. D., and Rogers, W. A. (2014). Toward a framework for levels of robot autonomy in human-robot interaction. *Journal of Human-Robot Interaction*, 3(2):74.

Blösch, M., Weiss, S., Scaramuzza, D., and Siegwart, R. (2010). Vision based mav navigation in unknown and unstructured environments. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 21–28.

bo Chen, Y., chen Luo, G., song Mei, Y., qiao Yu, J., and long Su, X. (2016). Uav path planning using artificial potential field method updated by optimal control theory. *International Journal of Systems Science*, 47(6):1407–1420.

Bohlin, R. and Kavraki, L. E. (2000). Path planning using lazy prm. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 1, pages 521–528 vol.1.

Bolt, R. A. (1980). *Put-that-There*: Voice and gesture at the graphics interface. *SIGGRAPH Comput. Graph.*, 14(3):262–270.

Bourquardez, O., Mahony, R., Guenard, N., Chaumette, F., Hamel, T., and Eck, L. (2009). Image-based visual servo control of the translation kinematics of a quadrotor aerial vehicle. *Robotics, IEEE Transactions on*, 25(3):743–749.

Brachman, R. J. (2002). Systems that know what they're doing. *IEEE Intelligent Systems*, 17(6):67–71.

Brisset, P., Drouin, A., Gorraz, M., Huard, P.-S., and Tyler, J. (2006). The paparazzi solution. In *MAV 2006, 2nd US-European Competition and Workshop on Micro Air Vehicles*.

Bristeau, P.-J., Callou, F., Vissiére, D., and Petit, N. (2011). The navigation and control technology inside the ar.drone micro {UAV}. {*IFAC*} *Proceedings Volumes*, 44(1):1477 – 1484. 18th {IFAC} World Congress.

Burri, M., D atwiler, M., Achtelik, M. W., and Siegwart, R. (2015). Robust state estimation for micro aerial vehicles based on system dynamics. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5278–5283.

Caron, F., Duflos, E., Pomorski, D., and Vanheeghe, P. (2006). Gps/imu data fusion using multisensor kalman filtering: introduction of contextual aspects. *Information Fusion*, 7(2):221 – 230.

Cauchard, J. R., E, J. L., Zhai, K. Y., and Landay, J. A. (2015). Drone & me: An exploration into natural human-drone interaction. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '15, pages 361–365, New York, NY, USA. ACM.

Chen, H., Chang, K., and Agate, C. S. (2013). Uav path planning with tangent-plus-lyapunov vector field guidance and obstacle avoidance. *IEEE Transactions on Aerospace and Electronic Systems*, 49(2):840–856.

Chen, Q. and Petriu, E. M. (2003). Optical character recognition for model-based object recognition applications. In *The 2nd IEEE Internatioal Workshop on Haptic, Audio and Visual Environments and Their Applications*, pages 77–82.

Churchill, W. and Newman, P. (2012a). Continually improving large scale long term visual navigation of a vehicle in dynamic urban environments. In *Proc. IEEE Intelligent Transportation Systems Conference (ITSC)*, Anchorage, USA.

Churchill, W. and Newman, P. (2012b). Practice makes perfect? managing and leveraging visual experiences for lifelong navigation. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Minnesota, USA.

Churchill, W. and Newman, P. (2013). Experience-based Navigation for Long-term Localisation. *The International Journal of Robotics Research (IJRR)*.

Clough, B. T. (2002). Metrics, schmetrics! how the heck do you determine a uav's autonomy anyway. Technical report, DTIC Document.

Comport, A. I., Mahony, R., and Spindler, F. (2011). A visual servoing model for generalised cameras: Case study of non-overlapping cameras. In *2011 IEEE International Conference on Robotics and Automation*, pages 5683–5688.

Craighead, J., Murphy, R., Burke, J., and Goldiez, B. (2007). A survey of commercial open source unmanned vehicle simulators. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 852–857.

Crassidis, J. L., Cheng, Y., Nebelecky, C. K., and Fosbury, A. M. (2009). Decentralized attitude estimation using a quaternion covariance intersection approach. *The Journal of the Astronautical Sciences*, 57(1-2):113–128.

Crassidis, J. L. and Markley, F. L. (2003). Unscented filtering for spacecraft attitude estimation. *Journal of guidance, control, and dynamics*, 26(4):536–542.

Cruz, L., Lucio, D., and Velho, L. (2012). Kinect and rgbd images: Challenges and applications. In *2012 25th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials*, pages 36–49.

Cuerno Rejado, C., Garcia Hernandez, L., Sanchez Carmona, A., Carrio Fernandez, A., Sanchez-Lopez, J. L., and Campoy Cervera, P. (2015). Evolution of the unmanned aerial vehicles until present. *DYNA DYNA-ACELERADO*.

Cummins, M. and Newman, P. (2008). Fab-map: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 27(6):647–665.

Cummins, M. and Newman, P. (2011). Appearance-only slam at large scale with fab-map 2.0. *The International Journal of Robotics Research*, 30(9):1100–1123.

Davis, D. N. (2002). Computational architectures for intelligence and motivation. In *Intelligent Control, 2002. Proceedings of the 2002 IEEE International Symposium on*, pages 520–525. IEEE.

Dechter, R. and Pearl, J. (1985). Generalized best-first search strategies and the optimality of a*. *J. ACM*, 32(3):505–536.

del Valle, T. M., del Blanco Adán, C. R., Núñez, F. J., and Santos, N. G. (2014). New generation of human machine interfaces for controlling uav through depth based gesture recognition. In *Proceedings of SPIE Defense, Security and Sensing Conference*, volume 9084.

Dequaire, J., Tong, C. H., Churchill, W., and Posner, I. (2016). Off the beaten track: Predicting localisation performance in visual teach and repeat. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 795–800.

Doitsidis, L., Weiss, S., Renzaglia, A., Achtelik, M. W., Kosmatopoulos, E., Siegwart, R., and Scaramuzza, D. (2012). Optimal surveillance coverage for teams of micro aerial vehicles in gps-denied environments using onboard vision. *Autonomous Robots*, 33(1):173–188.

Dudek, G., Sattar, J., and Xu, A. (2007). A visual language for robot control and programming: A human-interface study. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2507–2513.

Duffy, B. R., Dragone, M., and O'Hare, G. M. (2005). Social robot architecture: A framework for explicit social interaction. In *Android Science: Towards Social Mechanisms, CogSci 2005 Workshop, Stresa, Italy*.

Eberli, D., Scaramuzza, D., Weiss, S., and Siegwart, R. (2011). Vision based position control for mavs using one single circular landmark. *Journal of Intelligent & Robotic Systems*, 61(1):495–512.

Echeverria, G., Lassabe, N., Degroote, A., and Lemaignan, S. (2011). Modular open robots simulation engine: Morse. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 46–51.

Einhorn, E., Langner, T., Stricker, R., Martin, C., and Gross, H. M. (2012). Mira - middleware for robotic applications. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2591–2598.

Elkady, A. and Sobh, T. (2012). Robotics middleware: A comprehensive literature survey and attribute-based bibliography. *Journal of Robotics*, 2012.

Eustice, R., Walter, M., and Leonard, J. (2005). Sparse extended information filters: insights into sparsification. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3281–3288.

Fan, Y., Haiqing, S., and Hong, W. (2008). A vision-based algorithm for landing unmanned aerial vehicles. In *2008 International Conference on Computer Science and Software Engineering*, volume 1, pages 993–996.

Fels, S. and Hinton, G. (1993). Glove-talk: a neural network interface between a data-glove and a speech synthesizer. *Neural Networks, IEEE Transactions on*, 4(1):2–8.

Fiala, M. (2005). Artag, a fiducial marker system using digital techniques. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 590–596 vol. 2.

Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*.

Forster, C., Pizzoli, M., and Scaramuzza, D. (2014). Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22.

Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., and Marín-Jiménez, M. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280 – 2292.

Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., and Medina-Carnicer, R. (2016). Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51:481 – 491.

Gat, E. (1998). On three-layer architectures. In Kortenkamp, D., Bonnasso, R. P., and Murphy, R., editors, *Artificial Intelligence and Mobile Robots*. AAAI Press.

Geraerts, R. and Overmars, M. H. (2004). *A Comparative Study of Probabilistic Roadmap Planners*, pages 43–57. Springer Berlin Heidelberg, Berlin, Heidelberg.

Gillula, J. H., Huang, H., Vitus, M. P., and Tomlin, C. J. (2010). Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice. In *2010 IEEE International Conference on Robotics and Automation*, pages 1649–1654.

Gonzalez, R. C. and Woods, R. E. (2002). *Digital image processing*. Prentice Hall, Pearson Education International.

Grabe, V., Riedel, M., Bulthoff, H., Giordano, P., and Franchi, A. (2013). The telekyb framework for a modular and extendible ros-based quadrotor control. In *Mobile Robots (ECMR), 2013 European Conference on*, pages 19–25.

Graether, E. and Mueller, F. (2012). Joggobot: A flying robot as jogging companion. In *CHI '12 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '12, pages 1063–1066, New York, NY, USA. ACM.

Grisetti, G., Kummerle, R., Stachniss, C., and Burgard, W. (2010). A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43.

Gupte, S., Mohandas, P. I. T., and Conrad, J. M. (2012). A survey of quadrotor unmanned aerial vehicles. In *2012 Proceedings of IEEE Southeastcon*, pages 1–6.

Gustafsson, F. (2010). Particle filter theory and practice with positioning applications. *IEEE Aerospace and Electronic Systems Magazine*, 25(7):53–82.

Hamel, T. and Mahony, R. (2002). Visual servoing of an under-actuated dynamic rigid-body system: an image-based approach. *IEEE Transactions on Robotics and Automation*, 18(2):187–198.

Han, J. and Gold, N. (2014). Lessons learned in exploring the leap motion(tm) sensor for gesture-based instrument design. In Caramiaux, B., Tahiroglu, K., Fiebrink, R., and Tanaka, A., editors, *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 371–374, London, United Kingdom. Goldsmiths, University of London.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Hesch, J. A. and Roumeliotis, S. I. (2011). A direct least-squares (dls) method for pnp. In *2011 International Conference on Computer Vision*, pages 383–390.

Hidalgo-Paniagua, A., Vega-RodrĂguez, M. A., and Ferruz, J. (2016). Applying the {MOVNS} (multi-objective variable neighborhood search) algorithm to solve the path planning problem in mobile robotics. *Expert Systems with Applications*, 58:20 – 35.

Hoffmann, G. M., Waslander, S. L., and Tomlin, C. J. (2008). Quadrotor helicopter trajectory tracking control. In *AIAA guidance, navigation and control conference and exhibit*, pages 1–14.

Honegger, D., Meier, L., Tanskanen, P., and Pollefeys, M. (2013). An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1736–1741.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). Octomap: an efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206.

Hu, C., h. Meng, M. Q., Mandal, M., and Liu, P. X. (2006). Robot rotation decomposition using quaternions. In *2006 International Conference on Mechatronics and Automation*, pages 1158–1163.

Hu, C., Meng, M., Liu, P., and Wang, X. (2003). Visual gesture recognition for human-machine interface of robot teleoperation. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1560–1565 vol.2.

Huang, H. (2008a). Autonomy levels for unmanned systems (alfus) framework volume i: Terminology version 2.0. *National Institute of Standards and Technology (NIST). Special Publication 1011-I-2.0*.

Huang, H., Hoffmann, G. M., Waslander, S. L., and Tomlin, C. J. (2009). Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering. In *2009 IEEE International Conference on Robotics and Automation*, pages 3277–3282.

Huang, H.-M. (2007). Autonomy levels for unmanned systems (alfus) framework: safety and application issues. In *Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems*, pages 48–53. ACM.

Huang, H.-M., editor (2008b). *Autonomy levels for unmanned systems (ALFUS) framework*, volume I: Terminology. NIST Special Publication 1011-I. Contributed by the Ad Hoc Autonomy Levels for Unmanned Systems Working Group Participants.

Huang, H.-M., Pavek, K., Albus, J., and Messina, E. (2005a). Autonomy levels for unmanned systems (alfus) framework: An update. In *Defense and Security*, pages 439–448. International Society for Optics and Photonics.

Huang, H.-M., Pavek, K., Novak, B., Albus, J., and Messin, E. (2005b). A framework for autonomy levels for unmanned systems (alfus). *Proceedings of AUVSI Unmanned Systems 2005*.

Huang, H.-P. and Chung, S.-Y. (2004). Dynamic visibility graph for path planning. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2813–2818 vol.3.

Hwang, Y. K. and Ahuja, N. (1992). A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, 8(1):23–32.

Hwangbo, M., Kim, J. S., and Kanade, T. (2013). Imu self-calibration using factorization. *IEEE Transactions on Robotics*, 29(2):493–507.

Ishii, K., Zhao, S., Inami, M., Igarashi, T., and Imai, M. (2009). *Designing Laser Gesture Interface for Robot Control*, pages 479–492. Springer Berlin Heidelberg, Berlin, Heidelberg.

Jones, E. S. and Soatto, S. (2011). Visual-inertial navigation, mapping and localization: A scalable real-time causal approach. *The International Journal of Robotics Research*, 30(4):407–430.

Jones, G., Berthouze, N., Bielski, R., and Julier, S. (2010). Towards a situated, multimodal interface for multiple uav control. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1739–1744.

Ju, M.-H. and Kang, H.-B. (2007). Human robot interaction using face pose recognition. In *Consumer Electronics, 2007. ICCE 2007. Digest of Technical Papers. International Conference on*, pages 1–2.

Julier, S. J. and Uhlmann, J. K. (1997). New extension of the kalman filter to nonlinear systems.

Kalal, Z. (2011). *Tracking Learning Detection*. PhD thesis, University of Surrey.

Kalal, Z., Mikolajczyk, K., and Matas, J. (2012a). Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1409–1422.

Kalal, Z., Mikolajczyk, K., and Matas, J. (2012b). Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1409–1422.

Kaltenbrunner, M. and Bencina, R. (2007). reactivision: A computer-vision framework for table-based tangible interaction. In *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, TEI '07, pages 69–74, New York, NY, USA. ACM.

Kandepu, R., Foss, B., and Imsland, L. (2008). Applying the unscented kalman filter for nonlinear state estimation. *Journal of Process Control*, 18(7-8):753 – 768.

Kaushik, M. and Jain, R. (2014). Gesture based interaction NUI: an overview. *CoRR*, abs/1404.2364.

Kavraki, L. E., Svestka, P., Latombe, J. C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580.

Kelly, J. and Sukhatme, G. S. (2011). Visual-inertial sensor fusion: Localization, mapping and sensor-to-sensor self-calibration. *The International Journal of Robotics Research*, 30(1):56–79.

Kendoul, F. (2012). A survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotics*, 29(2):315–378.

Kim, A. and Golnaraghi, M. F. (2004). Initial calibration of an inertial measurement unit using an optical position tracking system. In *Position Location and Navigation Symposium, 2004. PLANS 2004*, pages 96–101.

Kingston, D. B. and Beard, R. W. (2004). Real-time attitude and position estimation for small uavs using low-cost sensors. In *AIAA 3rd unmanned unlimited technical conference, Workshop and exhibit*, pages 2004–6488. sn.

Klein, G. and Murray, D. (2009). Parallel tracking and mapping on a camera phone. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, pages 83–86.

Kmiec, M. (2011). New optimal character recognition method based on hu invariant moments and weighted voting. *Journal of Applied Computer Science*, 19(1):33–5.

Kneip, L., Martinelli, A., Weiss, S., Scaramuzza, D., and Siegwart, R. (2011). Closed-form solution for absolute scale velocity determination combining inertial measurements and a single feature correspondence. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4546–4553.

Kohlbrecher, S., Meyer, J., Graber, T., Petersen, K., von Stryk, O., and Klingauf, U. (2014). Robocuprescue 2014-robot league team hector darmstadt (germany). *RoboCupRescue 2014*.

Kong, W., Zhou, D., Zhang, D., and Zhang, J. (2014). Vision-based autonomous landing system for unmanned aerial vehicle: A survey. In *2014 International Conference on Multisensor Fusion and Information Integration for Intelligent Systems (MFI)*, pages 1–8.

Krajník, T., Nitsche, M., Faigl, J., Vaněk, P., Saska, M., Přeučil, L., Duckett, T., and Mejail, M. (2014). A practical multirobot localization system. *Journal of Intelligent & Robotic Systems*, 76(3):539–562.

Krajnık, T., Santos, J., Seemann, B., and Duckett, T. (2014). Froctomap: An efficient spatio-temporal environment representation. *Advances in Autonomous Robotics Systems*, page 269.

Kumar, V. and Michael, N. (2012). Opportunities and challenges with autonomous micro aerial vehicles. *The International Journal of Robotics Research*, 31(11):1279–1291.

Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). G2o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613.

Kushleyev, A., Mellinger, D., Powers, C., and Kumar, V. (2013). Towards a swarm of agile micro quadrotors. *Autonomous Robots*, 35(4):287–300.

Lavalle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning. Technical report.

LaViola, J. J. (2003). A comparison of unscented and extended kalman filtering for estimating quaternion motion. In *American Control Conference, 2003. Proceedings of the 2003*, volume 3, pages 2435–2440 vol.3.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Lee, D., Ryan, T., and Kim, H. J. (2012). Autonomous landing of a vtol uav on a moving platform using image-based visual servoing. In *2012 IEEE International Conference on Robotics and Automation*, pages 971–976.

Lee, K.-F., Hon, H.-W., and Reddy, R. (1990). An overview of the sphinx speech recognition system. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 38(1):35–45.

Lee, T., Leok, M., and McClamroch, N. H. (2011). Geometric tracking control of a quadrotor {UAV} for extreme maneuverability. {*IFAC*} *Proceedings Volumes*, 44(1):6337 – 6342. 18th {IFAC} World Congress.

Lefferts, E. J., Markley, F. L., and Shuster, M. D. (1982). Kalman filtering for spacecraft attitude estimation. *Journal of Guidance, Control, and Dynamics*, 5(5):417–429.

Leng, D. and Sun, W. (2009). Finding all the solutions of pnp problem. In *2009 IEEE International Workshop on Imaging Systems and Techniques*, pages 348–352.

Lepetit, V., Moreno-Noguer, F., and Fua, P. (2008). Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision*, 81(2):155.

Li, M., Kim, B. H., and Mourikis, A. I. (2013). Real-time motion tracking on a cellphone using inertial sensing and a rolling-shutter camera. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4712–4719.

Li, S., Xu, C., and Xie, M. (2012). A robust o(n) solution to the perspective-n-point problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1444–1450.

Lightbody, P., Hanheide, M., Krajnik, T., et al. (2017). A versatile high-performance visual fiducial marker detection system with scalable identity encoding.

Lim, H. and Lee, Y. S. (2009). Real-time single camera slam using fiducial markers. In *ICCAS-SICE, 2009*, pages 177–182.

Lim, H., Park, J., Lee, D., and Kim, H. (2012). Build your own quadrotor: Open-source projects on unmanned aerial vehicles. *Robotics Automation Magazine, IEEE*, 19(3):33–45.

Linegar, C., Churchill, W., and Newman, P. (2015). Work Smart, Not Hard: Recalling Relevant Experiences for Vast-Scale but Time-Constrained Localisation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, USA.

Linegar, C., Churchill, W., and Newman, P. (2016). Made to measure: Bespoke landmarks for 24-hour, all-weather localisation with a camera. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 787–794.

Ling-juan, M., Xiao-guang, L., Chun-yun, L., and Ya-ping, D. (2002). Derivation of nonlinear error equations of strapdown inertial navigation system using quaternion. In *SICE 2002. Proceedings of the 41st SICE Annual Conference*, volume 1, pages 636–640 vol.1.

Lippiello, V., Loianno, G., and Siciliano, B. (2011). Mav indoor navigation based on a closed-form solution for absolute scale velocity estimation using optical flow and inertial data. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 3566–3571.

Liu, C., Prior, S. D., Teacy, W. L., and Warner, M. (2016). Computationally efficient visual–inertial sensor fusion for global positioning system–denied navigation on a small quadrotor. *Advances in Mechanical Engineering*, 8(3):1687814016640996.

Loianno, G., Mulgaonkar, Y., Brunner, C., Ahuja, D., Ramanandan, A., Chari, M., Diaz, S., and Kumar, V. (2015). Smartphones power flying robots. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 1256–1263.

Lupashin, S., Hehn, M., Mueller, M. W., Schoellig, A. P., Sherback, M., and D'Andrea, R. (2014). A platform for aerial robotics research and demonstration: The flying machine arena. *Mechatronics*, 24(1):41 – 54.

Lupashin, S., Schollig, A., Sherback, M., and D'Andrea, R. (2010). A simple learning strategy for high-speed quadrocopter multi-flips. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1642–1648.

Lynen, S., Achtelik, M. W., Weiss, S., Chli, M., and Siegwart, R. (2013). A robust and modular multi-sensor fusion approach applied to mav navigation. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3923–3929.

Mahony, R., v. Brasch, A., Corke, P., and Hamel, T. (2005). Adaptive depth estimation in image based visual servo control of dynamic systems. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 5372–5378.

Markley, F. L. (2003). Attitude error representations for kalman filtering. *Journal of guidance, control, and dynamics*, 26(2):311–317.

Markley, F. L. (2004a). Attitude estimation or quaternion estimation? *Journal of Astronautical Sciences*, 52(1):221–238.

Markley, F. L. (2004b). Multiplicative vs. additive filtering for spacecraft attitude determination. In *Proceedings of the 6th Conference on Dynamics and Control of Systems and Structures in Space (DCSSS)*, volume 22.

Martínez, C., Mondragón, I. F., Campoy, P., Sánchez-López, J. L., and Olivares-Méndez, M. A. (2013). A hierarchical tracking strategy for vision-based applications on-board uavs. *Journal of Intelligent & Robotic Systems*, 72(3):517–539.

Mashood, A., Noura, H., Jawhar, I., and Mohamed, N. (2015). A gesture based kinect for quadrotor control. In *Information and Communication Technology Research (ICTRC), 2015 International Conference on*, pages 298–301.

Matsebe, O., Namoshe, M., Tlale, N., Pretoria, M., and Africa, S. (2010). *Basic Extended Kalman Filter-Simultaneous Localization and Mapping*. INTECH Open Access Publisher.

Medioni, G. and Kang, S. B. (2004). *Emerging Topics in Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Meier, L., Honegger, D., and Pollefeys, M. (2015). Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6235–6240.

Meier, L., Tanskanen, P., Fraundorfer, F., and Pollefeys, M. (2011). Pixhawk: A system for autonomous flight using onboard computer vision. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2992–2997.

Meier, L., Tanskanen, P., Heng, L., Lee, G. H., Fraundorfer, F., and Pollefeys, M. (2012). Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, 33(1):21–39.

Mellado-Bataller, I., Campoy, P., Olivares-Mendez, M. A., and Mejias, L. (2013a). *Rapid Prototyping Framework for Visual Control of Autonomous Micro Aerial Vehicles*, pages 487–499. Springer Berlin Heidelberg, Berlin, Heidelberg.

Mellado-Bataller, I., Pestana, J., Olivares-Mendez, M. A., Campoy, P., and Mejias, L. (2013b). *MAVwork: A Framework for Unified Interfacing between Micro Aerial Vehicles and Visual Controllers*, pages 165–179. Springer Berlin Heidelberg, Berlin, Heidelberg.

Mellinger, D., Michael, N., and Kumar, V. (2014). *Trajectory Generation and Control for Precise Aggressive Maneuvers with Quadrotors*, pages 361–373. Springer Berlin Heidelberg, Berlin, Heidelberg.

Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., and von Stryk, O. (2012). *Comprehensive Simulation of Quadrotor UAVs Using ROS and Gazebo*, pages 400–411. Springer Berlin Heidelberg, Berlin, Heidelberg.

Michael, N., Mellinger, D., Lindsey, Q., and Kumar, V. (2010). The grasp multiple micro-uav testbed. *IEEE Robotics Automation Magazine*, 17(3):56–65.

Milford, M. J. and Wyeth, G. F. (2012). Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1643–1649.

Mohammed, F., Idries, A., Mohamed, N., Al-Jaroodi, J., and Jawhar, I. (2014). Uavs for smart cities: Opportunities and challenges. In *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, pages 267–273.

Molina, M., Diaz-Moreno, A., Palacios, D., Suarez-Fernandez, R. A., Sanchez-Lopez, J. L., Sampedro, C., Bavle, H., and Campoy, P. (2016). Specifying complex missions for aerial robotics in dynamic environments. In *International Micro Air Vehicle Conference and Competition, IMAV 2016*, Beijing, China.

Monajjemi, V., Wawerla, J., Vaughan, R., and Mori, G. (2013). Hri in the sky: Creating and commanding teams of uavs with a vision-mediated gestural interface. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 617–623.

Mondragón, I. F., Campoy, P., Olivares-Mendez, M. A., and Martinez, C. (2011). 3d object following based on visual information for unmanned aerial vehicles. In *IX Latin American Robotics Symposium and IEEE Colombian Conference on Automatic Control, 2011 IEEE*, pages 1–7.

Mueller, F. F. and Muirhead, M. (2015). Jogging with a quadcopter. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 2023–2032, New York, NY, USA. ACM.

Murphy, R. (2000). *Introduction to AI robotics*. MIT press.

Nagi, J., Giusti, A., Di Caro, G. A., and Gambardella, L. M. (2014). Human control of uavs using face pose estimates and hand gestures. In *Proceedings of the 2014 ACM/IEEE International Conference on Human-robot Interaction*, HRI '14, pages 252–253, New York, NY, USA. ACM.

Neunert, M., Bloesch, M., and Buchli, J. (2016). An open source, fiducial based, visual-inertial motion capture system. In *2016 19th International Conference on Information Fusion (FUSION)*, pages 1523–1530.

Newcombe, R. A., Lovegrove, S. J., and Davison, A. J. (2011). Dtam: Dense tracking and mapping in real-time. In *2011 International Conference on Computer Vision*, pages 2320–2327.

Nielsen, J. (1993). Noncommand user interfaces. *Commun. ACM*, 36(4):83–99.

Nützi, G., Weiss, S., Scaramuzza, D., and Siegwart, R. (2011). Fusion of imu and vision for absolute scale estimation in monocular slam. *Journal of Intelligent & Robotic Systems*, 61(1):287–299.

Olson, E. (2010). A passive solution to the sensor synchronization problem. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1059–1064.

Olson, E. (2011). Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE International Conference on Robotics and Automation*, pages 3400–3407.

Panich, S. (2010). Indirect kalman filter in mobile robot application. *J. Math. Stat*, 6(381384.43).

Park, M. G., Jeon, J. H., and Lee, M. C. (2001). Obstacle avoidance for mobile robots using artificial potential field approach with simulated annealing. In *ISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No.01TH8570)*, volume 3, pages 1530–1535 vol.3.

Parker, L. E. (2008). Distributed intelligence: Overview of the field and its application in multi-robot systems. *Journal of Physical Agents*, 2(1):5–14.

Paul, R. and Newman, P. (2010). Fab-map 3d: Topological mapping with spatial and visual appearance. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2649–2656.

Pavlovic, V., Sharma, R., and Huang, T. (1997). Visual interpretation of hand gestures for human-computer interaction: a review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):677–695.

Pestana, J. (2012). On-board control algorithms for quadrotors and indoors navigation. Master's thesis, Universidad Politécnica de Madrid, Spain.

Pestana, J., Mellado-Bataller, I., Fu, C., Sanchez-Lopez, J. L., MondragĂłn, I. F., and Campoy, P. (2013a). A general purpose configurable navigation controller for micro aerial multirotor vehicles. In *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 557–564.

Pestana, J., Mellado-Bataller, I., Sanchez-Lopez, J. L., Fu, C., Mondragón, I. F., and Campoy, P. (2014a). *Floor Optical Flow Based Navigation Controller for Multirotor Aerial Vehicles*, pages 91–106. Springer International Publishing, Cham.

Pestana, J., Mellado-Bataller, I., Sanchez-Lopez, J. L., Fu, C., Mondragón, I. F., and Campoy, P. (2014b). A general purpose configurable controller for indoors and outdoors gps-denied navigation for multirotor unmanned aerial vehicles. *Journal of Intelligent & Robotic Systems*, 73(1):387–400.

Pestana, J., Sanchez-Lopez, J., Saripalli, S., and Campoy, P. (2014c). Computer vision based general object following for gps-denied multirotor unmanned vehicles. In *2014 American Control Conference (ACC)*, pages 1886–1891.

Pestana, J., Sanchez-Lopez, J., Suarez-Fernandez, R., Collumeau, J., Campoy, P., Martin-Cristobal, J., Molina, M., De Lope, J., and Maravall, D. (2014d). A vision based aerial robot solution for the iarc 2014 by the technical university of madrid. In *2014 International Aerial Robotics Competition*.

Pestana, J., Sanchez-Lopez, J. L., Campoy, P., and Saripalli, S. (2013b). Vision based gps-denied object tracking and following for unmanned aerial vehicles. In *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–6.

Pestana, J., Sanchez-Lopez, J. L., de la Puente, P., Carrio, A., and Campoy, P. (2014e). A vision-based quadrotor swarm for the participation in the 2013 international micro air vehicle competition. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 617–622.

Pestana, J., Sanchez-Lopez, J. L., de la Puente, P., Carrio, A., and Campoy, P. (2016). A vision-based quadrotor multi-robot solution for the indoor autonomy challenge of the 2013 international micro air vehicle competition. *Journal of Intelligent & Robotic Systems*, 84(1):601–620.

Polok, L., Lui, V., Ila, V., Drummond, T., and Mahony, R. (2015). The effect of different parameterisations in incremental structure from motion. In *Australasian Conference on Robotics and Automation (Robert Mahon 02 December 2015 to 04 December 2015)*, pages 1–9. The Australian National University.

Posner, M. I., Nissen, M. J., and Klein, R. M. (1976). Visual dominance: an information-processing account of its origins and significance. *Psychological review*, 83(2):157.

Preece, J., Rogers, Y., and Sharp, H. (2001). *Beyond Interaction Design: Beyond Human-Computer Interaction*. John Wiley & Sons, Inc., New York, NY, USA.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5.

Quigley, M., Goodrich, M., and Beard, R. (2004). Semi-autonomous human-uav interfaces for fixed-wing mini-uavs. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2457–2462 vol.3.

Roberge, V., Tarbouchi, M., and Labonte, G. (2013). Comparison of parallel genetic algorithm and particle swarm optimization for real-time uav path planning. *IEEE Transactions on Industrial Informatics*, 9(1):132–141.

Rothenstein, A. L. (2002). *A Mission plan specification language for behaviour-based robots*. PhD thesis, University of Toronto.

Rudol, P., Wzorek, M., Conte, G., and Doherty, P. (2008). Micro unmanned aerial vehicle visual servoing for cooperative indoor exploration. In *2008 IEEE Aerospace Conference*, pages 1–10.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition.

Rusu, R. B. and Cousins, S. (2011). 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4.

Sahingoz, O. K. (2013). Mobile networking with uavs: Opportunities and challenges. In *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, pages 933–941.

Salih, D. (2015). *Natural User Interfaces*. Research Topics in HCI, School of Computer Science, University of Birmingham, Birmingham.

Sampedro, C., Bavle, H., Rodriguez-Ramos, A., Carrio, A., Suárez Fernández, R. A., Sanchez-Lopez, J. L., and Campoy, P. (2017). [accepted] a fully-autonomous aerial robotic solution for the 2016 international micro air vehicle competition. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, page 0.

Sampedro, C., Bavle, H., Sanchez-Lopez, J. L., Suárez Fernández, R. A., Rodríguez-Ramos, A., Molina, M., and Campoy, P. (2016). A flexible and dynamic mission planning architecture for uav swarm coordination. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 355–363.

Sanchez-Lopez, J., Pestana, J., Collumeau, J.-F., Suarez-Fernandez, R., Campoy, P., and Molina, M. (2015). A vision based aerial robot solution for the mission 7 of the international aerial robotics competition. In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pages 1391–1400.

Sanchez-Lopez, J., Pestana, J., de la Puente, P., Suarez-Fernandez, R., and Campoy, P. (2014a). A system for the design and development of vision-based multi-robot quadrotor swarms. In *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, pages 640–648.

Sanchez-Lopez, J. L. (2012). Path following control system for car-like unmanned ground systems. Master's thesis, Universidad Politécnica de Madrid, Spain.

Sanchez-Lopez, J. L., Arellano-Quintana, V., Tognon, M., Campoy, P., and Franchi, A. (2017a). [accepted] visual marker based multi-sensor fusion state estimation. In *2017 IFAC World Congress*, page 0.

Sanchez-Lopez, J. L., Fu, C., and Campoy, P. (2016a). *FuSeOn: A Low-Cost Portable Multi Sensor Fusion Research Testbed for Robotics*, pages 57–68. Springer International Publishing, Cham.

Sanchez-Lopez, J. L., Molina, M., Bavle, H., Sampedro, C., Suarez-Fernandez, R. A., and Campoy, P. (2017b). [accepted] a multi-layered component-based approach for the development of aerial robotic systems: The aerostack framework. *Journal of Intelligent & Robotic Systems*, page 0.

Sanchez-Lopez, J. L., Pestana, J., and Campoy, P. (2017c). [accepted] a robust real time path planner for the collision free navigation of multirotor aerial robots in dynamic environments. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, page 0.

Sanchez-Lopez, J. L., Pestana, J., de la Puente, P., and Campoy, P. (2016b). A reliable open-source system architecture for the fast designing and prototyping of autonomous multi-uav systems: Simulation and experimentation. *Journal of Intelligent & Robotic Systems*, 84(1):779–797.

Sanchez-Lopez, J. L., Pestana, J., de la Puente, P., Carrio, A., and Campoy, P. (2013a). Visual quadrotor swarm for the imav 2013 indoor competition. In Armada, M. A., Sanfeliu, A., and Ferre, M., editors, *ROBOT2013: First Iberian Robotics Conference*, volume 253 of *Advances in Intelligent Systems and Computing*, pages 55–63. Springer.

Sanchez-Lopez, J. L., Pestana, J., Saripalli, S., and Campoy, P. (2014b). An approach toward visual autonomous ship board landing of a vtol uav. *Journal of Intelligent & Robotic Systems*, 74(1):113–127.

Sanchez-Lopez, J. L., Saripalli, S., Campoy, P., Pestana, J., and Fu, C. (2013b). Toward visual autonomous ship board landing of a vtol uav. In *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 779–788.

Sanchez-Lopez, J. L., Suárez Fernández, R. A., Bavle, H., Sampedro, C., Molina, M., Pestana, J., and Campoy, P. (2016c). Aerostack: An architecture and open-source software framework for aerial robotics. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 332–341.

Sanjeev Kunte, R. and Sudhaker Samuel, R. D. (2008). A simple and efficient optical character recognition system for basic symbols in printed kannada text. *Sadhana*, 32(5):521.

Sanna, A., Lamberti, F., Paravati, G., and Manuri, F. (2013). A kinect-based natural interface for quadrotor control. *Entertainment Computing*, 4(3):179 – 186.

Santamaria, E., Royo, P., Barrado, C., Pastor, E., LÃłpez, J., and Prats, X. (2008). Mission aware flight planning for unmanned aerial systems. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Honolulu (HI).

Saripalli, S. (2009). Vision-based autonomous landing of an helicopter on a moving target. In *Proceedings of AIAA Guidance, Navigation, and Control Conference, Chicago, USA*.

Saripalli, S., Montgomery, J. F., and Sukhatme, G. S. (2003). Visually guided landing of an unmanned aerial vehicle. *IEEE Transactions on Robotics and Automation*, 19(3):371–380.

Sarras, I. and Siguerdidjane, H. (2014). On the guidance of a uav under unknown wind disturbances. In *2014 IEEE Conference on Control Applications (CCA)*, pages 820–825.

Scaramuzza, D., Achtelik, M. C., Doitsidis, L., Friedrich, F., Kosmatopoulos, E., Martinelli, A., Achtelik, M. W., Chli, M., Chatzichristofis, S., Kneip, L., Gurdan, D., Heng, L., Lee, G. H., Lynen, S., Pollefeys, M., Renzaglia, A., Siegwart, R., Stumpf, J. C., Tanskanen, P., Troiani, C., Weiss, S., and Meier, L. (2014). Vision-controlled micro flying robots: From system design to autonomous navigation and mapping in gps-denied environments. *IEEE Robotics Automation Magazine*, 21(3):26–40.

Sch ollig, A., Augugliaro, F., Lupashin, S., and D'Andrea, R. (2010). Synchronizing the motion of a quadrocopter to music. In *2010 IEEE International Conference on Robotics and Automation*, pages 3355–3360.

Schwartz, B., Nagele, L., Angerer, A., and MacDonald, B. A. (2014). Towards a graphical language for quadrotor missions. In *Workshop on Domain-Specific Languages and models for Robotic systems*.

Sharp, C. S., Shakernia, O., and Sastry, S. S. (2001). A vision system for landing an unmanned aerial vehicle. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 2, pages 1720–1727 vol.2.

Shen, S., Mulgaonkar, Y., Michael, N., and Kumar, V. (2014). Multi-sensor fusion for robust autonomous flight in indoor and outdoor environments with a rotorcraft mav. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4974–4981.

Shojaie, K., Ahmadi, K., and Shahri, A. M. (2007). Effects of iteration in kalman filters family for improvement of estimation accuracy in simultaneous localization and mapping. In *2007 IEEE/ASME international conference on advanced intelligent mechatronics*, pages 1–6.

Shuster, M. D. (1993). A survey of attitude representations. *Navigation*, 8(9):439–517.

Singh, P. and Minsky, M. (2005). An architecture for cognitive diversity. *Visions of mind: architectures for cognition and affect*, 312:166.

Sloman, A. (1999). What sort of architecture is required for a human-like agent? In Wooldridge, M. and Rao, A., editors, *Foundations of Rational Agency*. Kluwer Academic Publishers.

Sola, J. (2013). Simulataneous localization and mapping with the extended kalman filter. *unpublished. Available: `http://www.joansola.eu/JoanSola/eng/JoanSola.html`*.

Sola, J. (2016). Quaternion kinematics for the error-state kf. *Laboratoire d'Analyse et d'Architecture des Systemes-Centre national de la recherche scientifique (LAAS-CNRS), Toulouse, France, Tech. Rep*.

Song, G., Miller, S., and Amato, N. M. (2001). Customizing prm roadmaps at query time. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 2, pages 1500–1505 vol.2.

Spica, R., Giordano, P. R., Ryll, M., Bülthoff, H. H., and Franchi, A. (2013). An open-source hardware/software architecture for quadrotor uavs. In *2nd Workshop on Research, Education and Development of Unmanned Aerial System*.

Stentz, A. (1994). Optimal and efficient path planning for partially-known environments. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 3310–3317 vol.4.

Stumm, E., Mei, C., Lacroix, S., and Chli, M. (2015). Location graphs for visual place recognition. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5475–5480.

Suárez Fernández, R. A., Sanchez-Lopez, J. L., Sampedro, C., Bavle, H., Molina, M., and Campoy, P. (2016). Natural user interfaces for human-drone multi-modal interaction. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1013–1022.

Tahri, O. and Chaumette, F. (2005). Point-based and region-based image moments for visual servoing of planar objects. *IEEE Transactions on Robotics*, 21(6):1116–1127.

Tahri, O., Mezouar, Y., Chaumette, F., and Corke, P. (2010). Decoupled image-based visual servoing for cameras obeying the unified projection model. *IEEE Transactions on Robotics*, 26(4):684–697.

Tedaldi, D., Pretto, A., and Menegatti, E. (2014). A robust and easy to implement method for imu calibration without external equipments. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3042–3049.

Terejanu, G. A. (2008). Extended kalman filter tutorial. *Online]. Disponible: http://users. ices. utexas. edu/˜ terejanu/files/tutorialEKF. pdf*.

Thrun, S., Koller, D., Ghahramani, Z., Durrant-Whyte, H., and Ng, A. Y. (2004). *Simultaneous Mapping and Localization with Sparse Extended Information Filters: Theory and Initial Results*, pages 363–380. Springer Berlin Heidelberg, Berlin, Heidelberg.

Trawny, N. and Roumeliotis, S. I. (2005). Indirect kalman filter for 3d attitude estimation. *University of Minnesota, Dept. of Comp. Sci. & Eng., Tech. Rep*, 2:2005.

Turk, M. and Robertson, G. (2000). Perceptual user interfaces (introduction). *Commun. ACM*, 43(3):32–34.

Van de Loosdrecht, J., Dijkstra, K., Postma, J., Keuning, W., and Bruin, D. (2014). Twirre: Architecture for autonomous mini-uavs using interchangeable commodity components. In *IMAV 2014: International Micro Air Vehicle Conference and Competition 2014, Delft, The Netherlands, August 12-15, 2014*. Delft University of Technology.

Walter, M. R., Eustice, R. M., and Leonard, J. J. (2007). Exactly sparse extended information filters for feature-based slam. *The International Journal of Robotics Research*, 26(4):335–359.

Wan, E. A. and Merwe, R. V. D. (2000). The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, pages 153–158.

Wang, J. and Olson, E. (2016). Apriltag 2: Efficient and robust fiducial detection. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4193–4198.

Weiss, S., Achtelik, M., Kneip, L., Scaramuzza, D., and Siegwart, R. (2011). Intuitive 3d maps for mav terrain exploration and obstacle avoidance. *Journal of Intelligent & Robotic Systems*, 61(1):473–493.

Weiss, S., Achtelik, M. W., Chli, M., and Siegwart, R. (2012a). Versatile distributed pose estimation and sensor self-calibration for an autonomous mav. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 31–38.

Weiss, S., Achtelik, M. W., Lynen, S., Chli, M., and Siegwart, R. (2012b). Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 957–964.

Weiss, S. and Siegwart, R. (2011). Real-time metric state estimation for modular vision-inertial systems. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4531–4537.

Wigdor, D. and Wixon, D. (2011). *Brave NUI World: Designing Natural User Interfaces for Touch and Gesture*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.

Wu, Y. and Hu, Z. (2006). Pnp problem revisited. *Journal of Mathematical Imaging and Vision*, 24(1):131–141.

Wurm, K. M., Hornung, A., Bennewitz, M., Stachniss, C., and Burgard, W. (2010). Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems. In *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, volume 2.

Xu, C., Qiu, L., Liu, M., Kong, B., and Ge, Y. (2006a). Stereo vision based relative pose and motion estimation for unmanned helicopter landing. In *2006 IEEE International Conference on Information Acquisition*, pages 31–36.

Xu, X., Xie, J., and Xie, K. (2006b). Path planning and obstacle-avoidance for soccer robot based on artificial potential field and genetic algorithm. In *2006 6th World Congress on Intelligent Control and Automation*, volume 1, pages 3494–3498.

Xu, Y., Chen, X., and Li, Q. (2014). Adaptive iterated extended kalman filter and its application to autonomous integrated navigation for indoor robot. *The Scientific World Journal*, 2014.

Yang, K. and Sukkarieh, S. (2012). Model predictive unified planning and control of rotary-wing unmanned aerial vehicle. In *2012 12th International Conference on Control, Automation and Systems*, pages 1974–1979.

Yang, S., Scherer, S. A., and Zell, A. (2013). An onboard monocular vision system for autonomous takeoff, hovering and landing of a micro aerial vehicle. *Journal of Intelligent & Robotic Systems*, 69(1):499–515.

Yao, K., Li, J., Sun, B., and Zhang, J. (2016). An adaptive grid model based on mobility constraints for uav path planning. In *2016 2nd International Conference on Control Science and Systems Engineering (ICCSSE)*, pages 207–211.

Yu, X. and Zhang, Y. (2015). Sense and avoid technologies with applications to unmanned aircraft systems: Review and prospects. *Progress in Aerospace Sciences*, 74:152 – 166.

Zhang, H. and Ostrowski, J. P. (1999). Visual servoing with dynamics: control of an unmanned blimp. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 1, pages 618–623 vol.1.

Zhang, J. and Singh, S. (2015). Visual-lidar odometry and mapping: low-drift, robust, and fast. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2174–2181.

Zhang, P., Gu, J., Milios, E. E., and Huynh, P. (2005). Navigation with imu/gps/digital compass with unscented kalman filter. In *IEEE International Conference Mechatronics and Automation, 2005*, volume 3, pages 1497–1502 Vol. 3.

Zhang, T., Kang, Y., Achtelik, M., Kuhnlenz, K., and Buss, M. (2009). Autonomous hovering of a vision/imu guided quadrotor. In *2009 International Conference on Mechatronics and Automation*, pages 2870–2875.

Zhang, X., Fronz, S., and Navab, N. (2002). Visual marker detection and decoding in ar systems: A comparative study. In *Proceedings of the 1st International Symposium on Mixed and Augmented Reality*, ISMAR '02, pages 97–, Washington, DC, USA. IEEE Computer Society.

Zhang, Y., wei Gong, D., and hua Zhang, J. (2013). Robot path planning in uncertain environment using multi-objective particle swarm optimization. *Neurocomputing*, 103:172 – 185.

Zhang, Z. (2012). Microsoft kinect sensor and its effect. *IEEE MultiMedia*, 19(2):4–10.

Zingg, S., Scaramuzza, D., Weiss, S., and Siegwart, R. (2010). Mav navigation through indoor corridors using optical flow. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3361–3368.

Zufferey, J. C., Beyeler, A., and Floreano, D. (2010). Autonomous flight at low altitude with vision-based collision avoidance and gps-based path following. In *2010 IEEE International Conference on Robotics and Automation*, pages 3329–3334.