UNIVERSITÉ DU LUXEMBOURG

PhD-FSTM-2020-56
The Faculty of Sciences, Technology and Medicine

DISSERTATION

Defence held on 01/10/2020 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Anush MANUKYAN

Born on 14 July 1988 in Yerevan (Armenia)

LEARNING OF CONTROL BEHAVIOURS IN FLYING MANIPULATION

Dissertation defence committee

Dr-Ing Holger VOOS, dissertation supervisor
Professor, Université du Luxembourg

Dr Miguel Angel OLIVARES MENDEZ
Assistant Professor, Université du Luxembourg

Dr Radu STATE, Chairman
Associate Professor, Université du Luxembourg

Dr Nico HOCHGESCHWENDER
Professor, Hochschule Bonn-Rhein-Sieg

Dr Matthieu GEIST, Vice Chairman
Professor, Université du Lorraine

UNIVERSITY OF LUXEMBOURG

DOCTORAL THESIS

Learning of Control Behaviours in Flying Manipulation

Author:
Anush MANUKYAN

Supervisors:
Prof. Dr.-Ing. Holger VOOS
A-Prof. Dr. Miguel OLIVARES-MENDEZ
Prof. Dr. Matthieu GEIST

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

in the

Automation and Robotics Research Group
Interdisciplinary Centre for Security, Reliability and Trust

October 2020

Declaration of Authorship

I, Anush MANUKYAN, declare that this thesis titled, Learning of Control Behaviours in Flying Manipulation and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Abstract

Anush MANUKYAN

Learning of Control Behaviours in Flying Manipulation

Machine learning is an ever-expanding field of research with a wide range of potential applications. It has been increasingly used in different robotics tasks enhancing their autonomy and intelligent behaviour. This thesis presents how machine learning techniques can enhance the decision-making ability for control tasks in aerial robots as well as amplify the safety, thus broadly improving their autonomy levels.

The work starts with the development of a lightweight approach for identifying degradations of UAV hardware-related components, using traditional machine learning methods. By analysing the flight data stream from a UAV following a predefined mission, it predicts the level of degradation of components at early stages. In that context, real-world experiments have been conducted, showing that such approach can be used as a safety system during different experiments, where the flight path of the vehicle is defined a priori. The next objective of this thesis is to design intelligent control policies for flying robots with highly nonlinear dynamics, operating in continuous state-action setting, using model-free reinforcement learning methods. To achieve this objective, first, the nuances and potentials of reinforcement learning have been investigated. As a result, numerous insights and strategies have been pointed out for crafting efficient reward functions that lead to successful learning performance. Finally, a learning-based controller is provided for controlling a hexacopter UAV with 6-DoF, to perform stable navigation and hovering actions by directly mapping observations to low-level motor commands. To increase the complexity of the given objective, the degrees of freedom of the robotic platform is upgraded to 7-DoF, using a flying manipulation as learning agent. In this case, the agent learns to perform a mission composed of take-off, navigation, hovering and end-effector positioning tasks. Later, to demonstrate the effectiveness of the proposed controller and its ability to handle higher number of degrees of freedom, the flying manipulation has been extended to a robotic platform with 8-DoF. To overcome several challenges of reinforcement learning, the RotorS Gym experimental framework has been developed, providing a safe and close to real simulated environment for training multirotor systems. To handle the increasingly growing complexity of learning tasks, the Cyber Gym Robotics platform has been designed, which extends the RotorS Gym framework by several core functionalities. For instance, it offers an additional mission controller that allows to decompose complex missions into several subtasks, thus accelerating and facilitating the learning process. Yet another advantage of the Cyber Gym Robotics platform is its modularity which allows to seamlessly switch both, learning algorithms as well as agents. To validate these claims, real-world experiments have been conducted, demonstrating that the model trained in the simulation can be transferred onto a real physical robot with only minor adaptations.

Keywords: Artificial Intelligence, Machine Learning, Reinforcement Learning, Unmanned Aerial Vehicles, Aerial Manipulation, Concept-based Mission Control, Neural Networks.

Acknowledgements

Over the past four years I have had the opportunity to meet, work with and learn from many professional and inspiring people, who left a big impact on this exciting and challenging journey.

First, I would like to express my sincere gratitude to my supervisor Prof. Dr-Ing Holger Voos, for giving the opportunity to do my PhD at the SnT Automation Robotic Research Group, for his support and guidance dedicated to set my work on the right path to achieve all the objectives. Also, for providing a great and pleasant atmosphere during the entire time. I am extremely grateful to A-Prof. Dr. Miguel Olivares-Mendez for having his door open to discuss any ideas and thoughts. I owe my deepest gratitude to Prof. Dr. Matthieu Geist for his support and constructive comments. Thank you for inspiring me, and for always providing valuable feedback, which have helped me in conducting this research.

I would like as well to say a big thank you to all my friends and colleagues. Serket, Amin Sajadi, Gary, Manuel, Paul, Dario, Claudio, thank you for the immense kindness and support.

A hearty thanks to my husband, Ralph, who was always there for me. Unintentionally you have been a part of this journey, encouraging me during the hardest times and celebrating with me every small victory. I feel myself very lucky to have you. You are the biggest support in my life. Thank you for your unconditional love.

Lastly, I want to thank to my parents and sister for believing in me and teaching that all our dreams can come true, if we have the courage to pursue them. Also, I want to thank my aunt who was always next to me through tough times. And finally, my Armenian friends, although we don't meet or even talk often, you are irreplaceable and have a special place in my heart. I hope I have been able to show you that far away in distance does not mean far away in heart and mind.

To everyone mentioned above – and to those written fondly in memories if not on the page, thank you!

Anush Manukyan,
Grand Duchy of Luxembourg, 2020

Contents

List of Figures

List of Tables

List of Abbreviations

UAV	**U**nmanned **A**erial **V**ehicle
FM	**F**lying **M**anipulation
UAM	**U**nmanned **A**erial **M**anipulation
MAV	**M**icro **A**erial **V**ehicle
AI	**A**rtificial **I**ntelligence
ML	**M**achine **L**earning
RL	**R**einforcement **L**earning
MDP	**M**arkov **D**ecision **P**rocess
DRL	**D**eep **R**einforcement **L**earning
NN	**N**eural **N**etwork
DNN	**D**eep **N**eural **N**etwork
TRPO	**T**rust **R**egion **P**olicy **O**ptimisation
PPO	**P**roximal **P**olicy **O**ptimisation
TD	**T**emporal **D**ifference
CGR	**C**yber **G**ym **R**obotics
ROS	**R**obotic **O**perating **S**ystem
CLI	**C**ommand **L**ine **I**nterface

To my husband, Ralph, for your
love, patience and
encouragement. There are no
words to describe what you
mean to me…

1 Introduction

This thesis presents how machine learning techniques can enhance the decision-making ability for control in aerial robots as well as amplify the safety, thus broadly improving the autonomy levels. Section 1.1 presents the research context, followed by research challenges and the motivations to conduct this study given in Section 1.2. Section 1.3 presents the aim and scope of this study, followed by the scientific and practical contributions, presented in Section 1.4. Finally, the outline of the thesis is provided in Section 1.5.

1.1 Context of the Study

Over the past decades, machine learning and robotics have become two major research areas in computer science and engineering domains, gaining immense interest among worldwide researchers. A report by Research and Markets predicts that the global machine learning market was valued at $1.5B in 2017 and is expected to reach $20.83B in 2024, with a compound annual growth rate of 44.06% between 2017 and 2024 [168, 169]. There are several key factors for such successful growth. For instance, the recent technological advancement impacted not only the development of machine learning but also had a strong influence on the progress in the field of robotics. A survey studying the ML-based applications reported that, nowadays, robotics is among the areas that ML algorithms are most frequently applied [26], with a market value of 2.8B, and expected to reach $12.36B by 2023 with a compound annual growth rate of 28.78% [72].

From other side robots have been around for decades. The first programmable robot has been developed as early as in the $1950's$, which was a model of a human arm. Although it was able to perform only primitive and limited actions, it became an inspiration for further developments of industrial robots. Following their success in the industry, these days robots have been deployed in many civilian applications outside the factories, in areas such as military, search and rescue, underwater, space and so on.

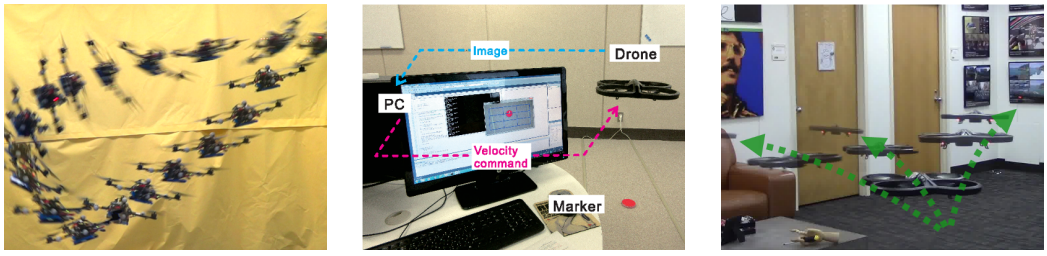
However, building robots that can be seamlessly integrated into the scarcely structured real-world environment is very challenging. To co-exist with humans, robots need to be able to perceive and process information about uncertain environments, be capable of making decisions about their future actions and operate following a see-think-act cycle [181].

While classic control approaches have shown promising results in the past years, the implementation of control architectures in a dynamic and unknown environment remains one of the main problems in robotic research. In recent years many researchers have focused their attention on studying control strategies enhanced by machine learning. On one hand, machine learning relies on algorithms that learn from data. On the other hand, modern robots are equipped with a wide variety of advanced sensors that let robots sense their environments and gather an enormous quantity of data. Thus, researchers have shown that the combination of these fields has the potential to solve the biggest challenges that the area of robotics faces [110, 204, 85, 159].

Despite the remarkable results achieved in the field of robotics over the past decades, there are remaining shortcomings that limit their usage in complex missions. Hence, the main objective of this work is to address these issues and provide practical solutions. Considering a large number of existing robotic systems, we limit our scope to aerial robotics, particularly, the multirotor Unmanned Aerial Vehicles and Flying Manipulators. All other robotic systems are outside of the scope of this work.

1.2 Problem Statement and Motivation

Unmanned Aerial Vehicles (UAVs) are one of the most commonly used robotic systems that are being increasingly deployed in many civil applications, ranging from military missions to videography and goods delivery [221]. Robotic manipulators are yet another widely used robotic system that have been involved in everyday



(A) (B) (C)

FIGURE 1.1: Motor skill learning of UAVs using reinforcement learning, a machine learning subfield. (A) A stabilization task from a hand throw [68]. (B) An acquisition of a stable hover behaviour above a marker [39]. (C) UAV learns to navigate in an indoor cluttered environment. The learning is performed through more than 10.000 crashes. This dataset gathered through crashes further acts as a negative instruction and teaches the drone how NOT to crash [51].

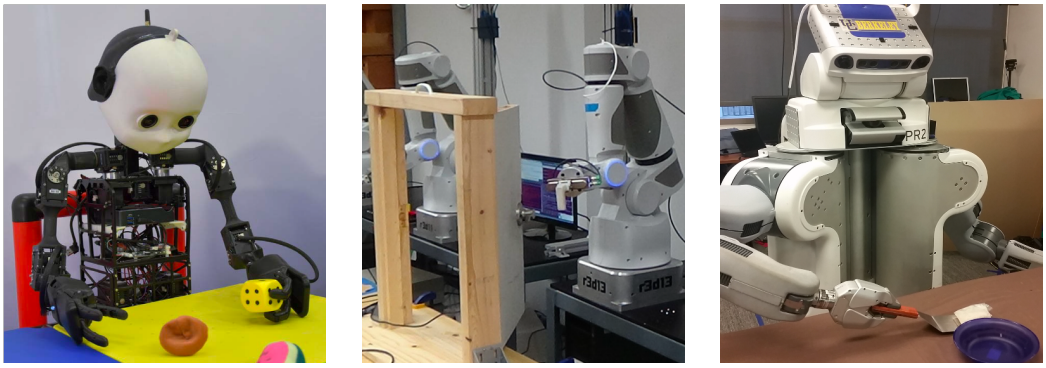
tasks in industry already for decades [90, 150, 145, 148]. Recently, a novel robotic platform, known as Unmanned Aerial Manipulators (UAM), or Flying Manipulators (FM), became a hot research topic in the field of robotics [144, 45]. Flying Manipulators bring immense potential for solving numerous problems, difficult and/or dangerous to perform by humans or other mobile robots. However, being a combination of two complex systems, FMs have highly nonlinear dynamics, which create many difficulties when designing control policies. Although there are many applications of these systems [109, 212], that have shown remarkable results, the tasks specifications, system dynamics and environmental details are usually hardcoded by the engineer. These constraints lead to many limitations in terms of model design, non-linear approximations and computation efficiency.

In order to address these complexities several advanced adaptive control techniques have been proposed, which are used to adjust the parameters of an agent in real-time for maintain a desired level of performance when the parameters of the system are unknown and/or change with time. For instance, Model Reference Adaptive Control [179], Incremental Nonlinear dynamic Inversion (INDI) [185], Backstepping [35], Dynamic inversion and NNs [107] are promising methods that implement control systems for handling system nonlinearities and model inaccuracies. Another noteworthy approach often used in Unmanned Aerial Vehicle [171] control is known as nonlinear Model Predictive Control [5, 65], offers robust control solutions for robots with nonlinear system dynamics [111]. While these methods demonstrated great successes in different complex missions, there are still remaining some practical issues and major challenges that control engineers are facing.

Below we define three main problems that we aim to address in this work:

- The first challenge is the difficulty of obtaining a detailed high accuracy model of the non-linear systems due to their complicated aerodynamic characteristics. Thus, it is crucial to have more generalized controllers that allow robots to operate in unconstrained and less structured environments without having any prior knowledge about the model of system dynamics.

- On another side, operating in unknown and dynamic environments may negatively affect the safety of unmanned aerial robots. For instance, environmental disturbances or slight collisions may deliver damages on different hardware components of a UAV or flying manipulators, leading to catastrophic crashes.



(A) (B) (C)

FIGURE 1.2: Solving grasping and other visuomotor problems using machine learning techniques. (A) NICO robot performs pick-up and grasping tasks by first performing an object detection, and after generating the grasp trajectory using deep neural networks [39]. (B) Robots learn a door opening task. The study uses a method that allows multiple robots to cooperatively learn a single policy with deep reinforcement learning [58]. (C) PR2 learning to scoop a bag of rice into a bowl with a spatula [46].

Likewise, these components can be impacted by changes over time in properties, such as propeller degradations, material loss of elasticity or shorten battery life, resulting to anomalous behaviours. Therefore, having a real-world technique that can continually monitor the UAV's behaviour and identify any arise failures, pointing out their risk factor can be highly advantageous. Thus, the second objective is to tackle safety challenge.

- Finally, the third challenge is the highly nonlinear nature of UAVs and flying manipulators, which often complicates the experiments leading to hazardous and non-desirable situations. Hence, having a convenient and easily accessible platform, that provides close-to-real-world like environment is an essential tool for robotic research. Such platform can be beneficial for carrying out initial experiments in an inexpensive way, and only after the desired behaviour is achieved transfer the control model on a physical robot to perform the fine-tuning in real-world settings.

Following the notable successes of machine learning in different robotic applications, the primary research question of this thesis is formulated as follows:

"How machine learning can enhance the decision-making ability for control in aerial robots as well as amplify the safety broadly improving the autonomy levels."

1.3 Aim and Scope

The aim of this study is to implement control strategies that contribute towards the goal of enhancing the intelligent decision-making property of aerial robots as well as the mastering novel complex control behaviours with ease.

To attain these objectives supervised learning and reinforcement learning, two subfields of machine learning, have been used, both enabling novel applications for

solving challenging robotics problems. In supervised learning the system is presented with labeled samples, and the task is to learn mapping from input data to the output data. It is a robust tool, commonly used for solving problems using regression or classification methods. In this work supervised learning is used as an analytical tool for increasing the safety and autonomy of a UAV.

From other side, reinforcement learning is a powerful mathematical framework that enables agents to learn sophisticated and hard-to-engineer behaviours autonomously. By interacting with an unknown environment an agent discovers a desired control policy based on the feedback received from its environment in form of reward or penalty. One of the main advantages of reinforcement learning is that it does not require any prior knowledge of the system and allows model-free learning. Such approach is very beneficial when dealing with sophisticated robotic systems, such as UAVs and flying manipulators, with large degrees of freedom and highly nonlinear dynamics. It endows the robots with the ability of learning different complex tasks from a simple reward structure. Ultimately, reinforcement learning is applicable to large span of robotics problems because of its convenient framework. In this thesis reinforcement learning has been used for improving the decision-making of aerial robots.

Finally, we decompose the primary research question into several sub-questions, formulated as follows:

- Could machine learning algorithms be used to solve safety problem of a UAV and enhance its autonomy using its flight data. (Chapter 3)
- What are the best machine learning approaches demonstrating notable results in robotics applications. (Chapter 2)
- Can reinforcement learning be used as an optimal control approach for designing basic control policies for robots with nonlinear, complex dynamics, such as UAVs and Flying manipulators. (Chapter 5, 6)
- Could the same approach be extended to solve more sophisticated problems. (Chapter 6)
- How the learning procedure could be improved, more particularly the learning of more sophisticated missions. (Chapter 4, 5, 6)
- What reward functions produce the best results. (Chapter 5)
- How to provide safe and inexpensive learning for mobile robots. (Chapter 4)
- Limitations of learning-based methods. (Chapter 7)

1.4 Significance of the Study

The contributions resulting from extensive research and series of experiments with respect to the research sub-questions stated above are detailed below:

- The first contribution of this study is the development of a lightweight safety system for a commercially available UAV. Using machine learning algorithms, the proposed approach identifies occurred degradation of hardware components of a UAV in real time and estimates the severity level based on its flight data. Such an approach may improve the autonomy of the mobile robot and

help to avoid forthcoming accidents by anticipating degradations before they become catastrophic and beyond the point of recovery. Moreover, the method has been tested on Ar.Drone 2.0, a commercially available UAV.

- The second contribution of this work is the design, implementation and application of reinforcement learning based methods for solving complex motor skill learning tasks. The developed controller has been applied on different learning tasks in a continuous state-action space domain and has been validated through the successfuly performance of three missions, each increasingly complex: During the first mission, a hexacopter UAV, such as an AscTec Firefly with 6-DoF, has been tasked to take off and hover at a given position. For the second mission, a robotic manipulation with 1-DoF has been attached to the UAV's base. The mission has been adapted to consist of a take off, navigation, hovering and end-effector positioning. Here, the complexity comes not only from the added degree of freedom, but also from the introduced disturbances created by the robotic arm, thus increasing the difficulty for the UAV to achieve its goal. Finally, the third mission is a logical extension of the second mission, where we increase the complexity further by upgrading the robotic arm from 1-DoF to a 2-DoF configuration, meaning a robotic arm with 2 joints. The tasks are identical to the second mission, but the mission proved the validity, modularity and extensability of the approach.

- The third contribution is the development of a modular platform, called Cyber Gym Robotics, an experimental framework for benchmarking reinforcement learning methods on different models of multirotor systems, or any other custom robotic platform. Despite the remarkable results that reinforcement learning methods achieve, they require a large number of training samples, obtained through tens of thousands of trials. Performing such way of training in real world settings is very expensive and dangerous for aerial robots. From the other side, reinforcement learning methods are highly sensitive to the underlying model of the environment where the training of the robot has proceeded. Hence, we designed a generic framework that provides a close-to-real world setting and an abstraction layer to seamlessly switch between learning algorithms and learning agents, including simulations. Upon achievement of the desired behaviour in a simulated environment, the platform allows to swap the learning agent for a physical robot interface and run the learned model in a real world setting. Additionally, to improve the learning of complex missions, we developed a concept-based controller, providing a system for users to organise a mission into subtasks. This has the added benefit of accelerating the learning process as well as facilitating the writing of the reward function.

- Last but not least, the fourth contribution of this work is the reconstruction of a hexacopter UAV by reusing its stripped-down frame to install new hardware components, such as motors, electronic speed controllers (ESC), power distribution board, on-board computer, including sensors and complete software stack, and radio receiver. Figure 1.3 illustrates the different stages of the build process. The base robot is an AscTec Firefly UAV and has been used for all experiments, both simulated and real.

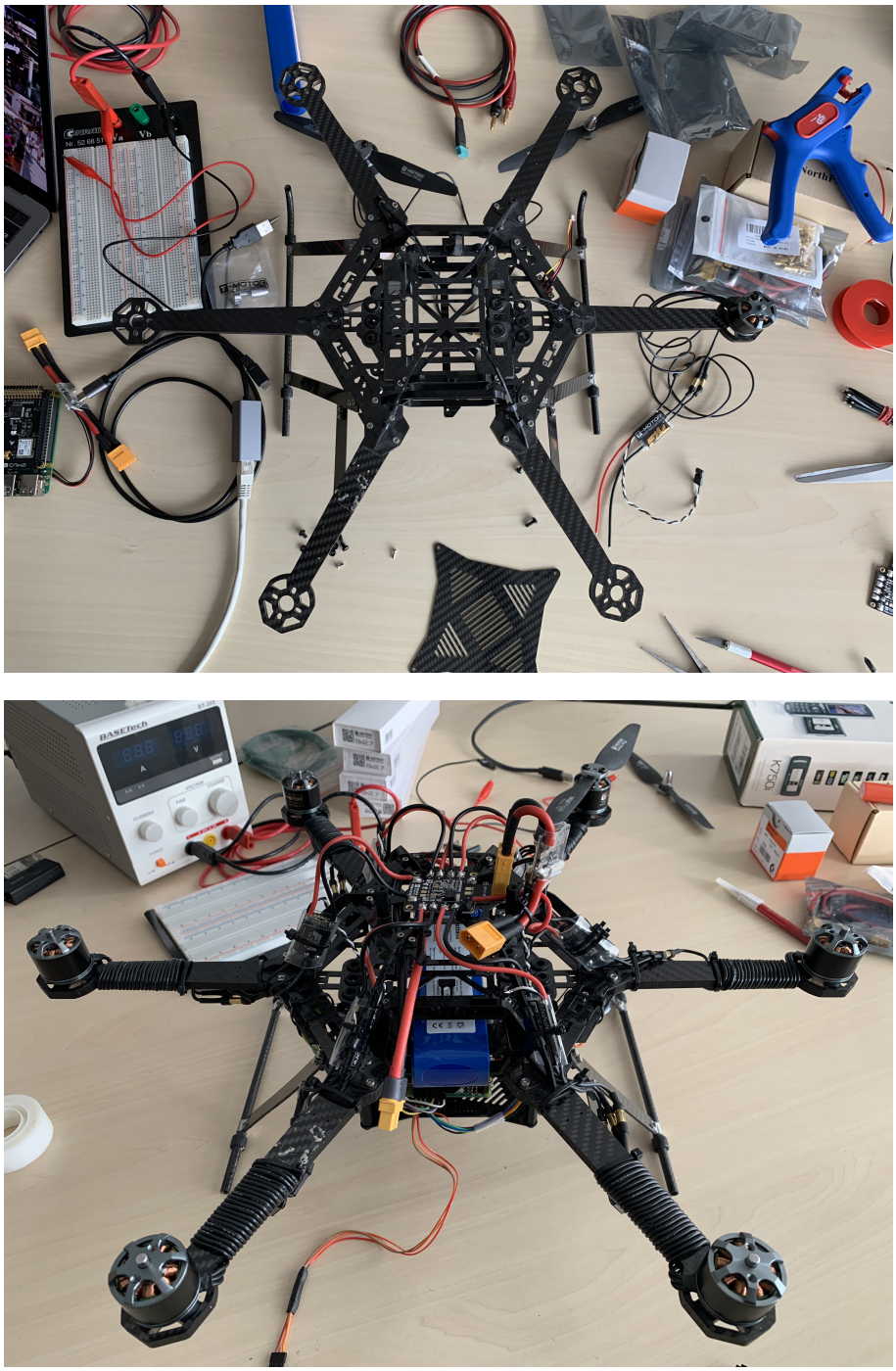


FIGURE 1.3: Reconstruction of the AscTech Firefly UAV.

1.5 Outline of the Thesis

This thesis consists of seven further chapters, which some of these chapters are based on published material.

In Chapter 2 the brief history of the aerial robotic system is presented. Then, the comprehensive review of the theoretical framework and recent research has been carried out. As a result, a detailed overview of essential machine learning methods applied on robotics research, along with the successful robotic application are discussed.

Chapter 3 presents the safety system proposed in this thesis. First, it describes two fundamental steps of the approach. After, it presents the experimental setup and performed experiments. Finally, the it concludes by presenting the obtained results.

In Chapter 4 we present the Cyber Gym Robotics framework designed in this work. First, we discuss the challenges of reinforcement learning method and the importance of a virtual training environment. Then, we present the architecture of the Cyber Gym Robotics platform. Finally, the proposed Concept-based Mission Controller is described in great details along with its structure, software specifications. Finally, the real-world epxriments is demonstrated, which have been performed to validate the effectiveness of the proposed Cyber Gym Robotics platform.

Chapter 5 we tackle the problem of controlling flying robots using machine learning techniques. We demonstrate our design of intelligent controller using model-free reinforcement learning algorithm. Finally, we present the obtained results of the hexacopter UAV while learning a navigation and stable hovering task.

In chapter 6 we increased the complexity of the learning problem demonstrated in chapter 5 by adding a flying manipulation, giving the agent a total of 7 and afterwards 8 degrees of freedom. First, we present the general workflow of the Cyber Gym Robotics, used for training the flying manipulations in this chapter. After a detailed description of the obtained results during the entire learning process is given.

Finally, the thesis concludes in Chapter 7 with the summary of findings and contributions, followed by an outlook of future projects.

2

Theoretical Framework and Related Approaches

Achieving autonomous flight of Unmanned Aerial Vehicles within unknown or uncertain environments involves many guidance and control challenges. Over the past years, many machine learning-based methods have shown remarkable breakthroughs in different robotic applications, such as path planning, obstacle avoidance, perception or safety, to mention a few.

In that context, a comprehensive review of the theoretical framework and recent research has been carried out. We first present a brief history of unmanned aerial vehicles, then introduce the notable successes that machine learning methods achieved in solving anomaly detection and control-related problems in the field of robotics. Then, we discuss about its essential theoretical background along with the baseline methods that have been extensively used in recent literature.

The basic algorithmic principles and the resources set a foundation for the work in the rest of this thesis.

2.1 A Brief History of Unmanned Aerial Vehicles

Since the early civilisation, one of the greatest dreams of mankind was to develop skilled and intelligent machines that are able to imitate their behaviours.

The history of robotics stretches from Egyptian water clocks used human figurines to strike the hour bells (3000 B.C.) to Greek philosopher Aristotle's ideas about automated tools (350 B.C.), all the way through to Henry Ford's Model T assembly line (20th century) and beyond [180]. Thanks to humans' creative mind and ambitious characteristics, this dream continued to flourish over the past centuries, resulting in significant advancements in the field of robotics. Nowadays, there exist different fully autonomous robotic systems which are widely used for facilitating our day-to-day life as well, as for performing tasks that are hazardous or inaccessible for humans or require stable and highly accurate operations.

In general, robotic systems can be classified based on different characteristics, for instance, based on their degree of freedom, kinematic structure, the power source, the workspace geometry, applications or movements. In this section, we will categorise the robotic systems based on movement [180].

- **Fixed robots:** these robots are mounted on stable bases and are more prominent in the industries working in well-defined environments.
- **Mobile robots:** these robots are the ones who are capable of freely move over the surface. Mobile robots themselves can be split into several categories, such as legged robots, wheeled robots, swimming robots, and flying robots.
  - **Wheeled robots:** these robots are able to navigate over the surface using their wheels.
  - **Legged robots:** these are the robots that are capable of walking by using libs for motion. These robots are aimed for navigating in highly unstructured environments with a different kind of path and terrain.
  - **Swimming robots:** these robots are the ones that are able to perform tasks underwater.
  - **Flying robots:** the range of these robots vary from micro-sized robots that mimic the morphology of an insect and reach to large drones and helicopters known as well as Unmanned Aerial Systems.

2.1.1 Unmanned Aerial Vehicles

The significance of Unmanned Aircraft Systems (UASs), also known as Unmanned Aerial Vehicles (UAVs) or drones, have been increasing over the last decades becoming one of the most appealing subfields of robotics. Until recently, use cases of drones have been limited mainly to military purposes. The technological advancements have played an essential role in the rapid development of UAV research. Today, UAVs are being deployed in a wide variety of applications, ranging from leisure entertainment to surveillance and videography.

The UAV platforms can be classified based on their endurance and manoeuvrability properties as follows:

- **Fixed-wing UAVs** are referred to unmanned airplanes with wings attached on their body-frame. In general, they can fly at high speed and have long endurance [14].

- **Rotary-wing UAVs**, also called rotorcraft UAVs, are the aircrafts capable of hovering and have high manoeuvrability. These vehicles are one of the most commonly used UAVs in different robotic missions, especially in civilian applications.
- **Bioinspired UAVs** are micro aerial vehicles that are designed to imitate the exceptional flying behaviours of biological species, for example, birds and insects.
- **Blimps** are lighter-than-air systems, are generally large-sized, have long endurance and very low speed. Such system examples are balloons and airships.

Nowadays, the aerial robots have advanced to a state in which sophisticated sensors, lightweight microcomputers and several communication interfaces are carried onboard. Based on the purpose of sensors, we can classify them as proprioceptive/exteroceptive and passive/active [202]:

- Proprioceptive sensors are used to measure the internal values of the system. For instance, motor speed, robot arm joint or battery voltage.
- Exteroceptive sensors provide information about the robot's environment, such as distance measurements, sound amplitude and even light intensity.
- Passive sensors are the ones that measure ambient environment energy entering the sensor, e.g. thermometers, microphones and cameras.
- Active sensors are, for instance, ultrasonic sensors and laser rangefinders that emit energy into the environment for measuring the environmental reactions.

With the advancements of UAV systems, the research interests in developing novel control solutions increased dramatically. Some of the widely used control policies for autonomous navigation of UAVs are proportional-integral-derivative (PID) [210], Model Predictive Control (MPC) [25], Simultaneous localization and mapping (SLAM) [188], linear quadratic regulator (LQR) [32], etc.

Learning-based control techniques are yet other practical control approaches that have gained the attention of many researchers over the past years. Some of these methods are known as reinforcement learning, deep learning and deep reinforcement learning. These methods have the advantage of being less dependent of the mathematical model of the robot, requiring only minor knowledge about the environment where the agent operates. More detailed introduction to these methods is given later in this chapter.

2.1.2 Robotic Manipulators

Robotic manipulators are one of the most commonly used fixed robots that have been commonly used in many areas of manufacturing for many decades.

In general, a robotic manipulator is a tool that is used to manipulate objects without having direct physical contact by a human operator. The ability to interact with and modifying its environment opens many possibilities for a vast number of applications.

Initially, this kind of robots were designed for dealing with radioactive or biohazardous materials, but in recent developments, the applications are ranging from welding automation to robotic surgeries and are deployed in space missions.

There exist many different types of robotic manipulators for performing from very basic to very complex manipulation tasks. A typical robotic manipulator consists of a series of joints and links. The joint provides a certain degree-of-freedom (DoF) motion between two links of the robot. The complexity of the manipulator is defined based on the number of degrees-of-freedom they process [212].

The manipulator systems themselves consist of a body, also known as a robotic arm, and an end-effector. Depending on the functionalities of the manipulator, the end-effector may differ. The most commonly used end-effects are grippers. However, the end-effectors of industrial robotic manipulators often consist of different tools, such as suction cups, magnetic grippers, or painting cells [52].

Based on the axes of movements and degrees-of-freedom, the robotic manipulators can be classified as a linear, planner, cartesian, cylindrical and articulated. Linear manipulators are the most basic manipulators having only one prismatic joint. Their movements are limited to only a single direction. Planner manipulators can be described as two linear manipulators attached to each other. In general, they have two degrees-of-freedom. The end-effector can move along two mutually perpendicular axes, covering all the points in the plane defined by the range of motion along the two axes. Cartesian manipulators provide movements along the three Cartesian axes, which are three orthogonal axes. They consist of only prismatic joints and can reach any position in its rectangular workspace by Cartesian motions of the links. Cylindrical manipulators are similar to Cartesians, but they can reach any point in the cylindrical workspace. Finally, articulated manipulators are one of the most sophisticated manipulators with rotary joints, also called revolute. The number of joints can range from two to ten or more. These manipulators are able to provide relatively large movement freedom in a compact space.

2.1.3 Flying Manipulators

Unlike unmanned aerial vehicles and robotic manipulators, flying manipulators are a relatively new type of aerial robotic systems.

In general, the most UAV applications involve either monitoring/filming and/or sensing tasks, while robotic manipulators are able to physically interact with and modify their environments.

The combination of these multifunctional systems results in a novel robotic platform, known as aerial or flying manipulators.

Flying manipulators are sophisticated robotic platforms that extend the usage of UAVs, providing them with the ability to interact with their environments. This opens the immense potential for solving numerous problems and has been already deployed in many different real-world applications, ranging from bridge inspection [30], the opening of valves [218] and canopy sampling of the environment [79].

Despite the potential advantages of flying manipulators, they face many challenges, such as limited payload that limits the manipulator size and payload, the complexity of control because of its highly non-linear dynamics and precise localisation, to mention a few. Because of these issues, the number of commercially available flying manipulators is minimal. However, an increasing number of studies are being carried out around this topic, and different approaches are being explored and developed [99, 141].

2.2 Related Approaches

Keeping robots safe, enhancing their autonomy and endowing them with intelligent decision-making abilities remain the primary mission of this work. However, before introducing our proposed solutions to these problems, we first provide a broad survey of the current developments of machine learning applications in the domain of robotics and present technical details of some representative baseline algorithms.

Taking into account that a tremendous number of researches have been made in both robotics and machine learning domains, it would not be possible to cover all the aspects in one work. Thus, we set constraints on the scope of this thesis, by mainly focusing on the research related to the design of control policies for multirotor systems based on the sensorimotor data.

More specifically, our points of interests are robotic systems, such as UAVs and flying manipulators. Since flying manipulators are composed of UAVs and robotic arms, we also cover the existing applications related to these robotic manipulators. Finally, we present the methods and algorithms that have been commonly used in many aerial robotics applications in great details.

We split this chapter into two main parts: 1) the review of the state of the art work covering the applications of anomaly detection in UAVs, and the notable achievements of learning control behaviours in UAVs and Flying Manipulators, 2) detailed theoretical background on machine learning along with the baselines methods widely used for solving robotic problems.

2.2.1 Anomaly detection in Unmanned Aerial Vehicles

Nowadays, UAVs play an increasingly important role in different daily activities. However, often operating in a dynamic and unknown environment may cause hard-to-detect hardware-related degradations. Over time these degradations can worsen and lead to dangerous situations or even catastrophic crashes. Therefore, it is widely desirable to provide a safety system that can automatically detect incipient failures and pinpoint the root cause of the evolving anomalies. In general, robots are equipped with vast array of sensors that can be used as an effective source for fault detection systems. For instance, by constantly monitoring and analysing time series data streamed by different sensors of a UAV, it is possible to recognize indicative fault data that do not conform to some explicit laws or historical patterns.

Due to its proven ability of learning from data, and observe abnormal patterns, machine learning-based methods gained extensive attention for anomaly detection-related tasks. For instance, in applications where the desired outcome is known, and a labelled data is possible to construct, supervised learning-based methods have shown attractive performances. Such a scenario can be met in environments where robots perform repetitive tasks, such as in manufacturing environments, factories, museums, etc.

One of the commonly used method is the Mahanalobis distance when it is used to compute the difference between two datasets. For instance, Khalastchi et al. [86] use the Mahanalobis distance as an anomaly detection identifier for light-weight robots. In order to work with high-dimensional sensor data, they first find correlations between the data attributes [108]. Then they apply the Mahanalobis distance measure function, which finds the degree of difference between a predefined threshold and a set of unlabelled data. Another method based on Mahanalobis distance have been presented by Brotherton et al. [19] for detecting anomalies in military aircraft subsystem data. By computing the Mahanalobis distance between the existing

dataset and new received data, they try to predict whether or not it has an abnormal behaviour.

Anomalies due to Wind Gusts (AWG) is another supervised learning-based anomaly detection tool proposed by Afridi et al. [2]. It is used to detect accurately the wind gust anomalies in the altitude control unit of an Aerosonde UAV. They apply J48 [15] decision tree algorithm, which achieves competitive results based on their experiments. In their work Duan et al. [36] address the issue when the training data is sparse, e.g. when dealing with too small size of data. They apply Relevance Vector Machine (RVM) [200] for mining the correlations between the training and test data. Their experiments in a simulated environment show that because of its computing simplicity their approach can be applied in online scenario, obtaining an accuracy rate of 87%. Another study uses Micro-Electro-Mechanical (MEMS) light weight sensor [112] for anomaly detection in UAV flights. They classify the raw data of MEMS sensor based on the flight mode. Then, they apply Kernel Principal Component Analysis (KPCA) [89] for mining the correlations between test and training data and perform anomaly detection. In their experiments they prove that MEMS sensor can be very effective for performing anomaly detection of UAV flights, because of its light weight and easy-to-deploy features. Chen et al. [28] implement a non-invasive anomaly detection method for UAVs. They apply last squares support vector machine (LS-SVM) regression method for predicting vertical speed anomalies. Various other approaches have been proposed for solving such a safety issue [22, 47, 23]. However, they typically focus on detecting the existence of anomalies. Thus, in this work, we present a degradation identification technique that not only intends to detect any occurred abnormalities in the robot's behaviour but also evaluates the severity level of the damage. The proposed approach is presented in chapter 3 in great details.

2.2.2 Learning-based control for Unmanned Aerial Vehicles

Reinforcement learning is another subfield of machine learning that gained broad interest among researchers, because of its power of solving a remarkable variety of problems, ranging from computer vision and natural language processing to robotics and medicine [178, 93].

Below we present a state of the art work of reinforcement learning methods applied on different different aerial robotics application, mainly focusing on UAVs, robotic manipulations and flying manipulations.

Unmanned Aerial Vehicles

Over the past decade, many researchers deployed reinforcement learning methods in different control applications, ranging from multicopter hovering to navigating in an unknown environments. For instance, Imanberdiyev et al. [174] applied TEXPLORE [156] , a model-based reinforcement learning algorithm on a UAV navigation task. The main objective of the quadcopter UAV was to learn to reach a goal position while monitoring its battery levels and modify the flight path if necessary. As an experimental environment, the authors use the Gazebo simulator [208] combined with Robotic Operating System (ROS) [170], where they demonstrate that the UAV was able to successfully perform such a task using only learning methods. Pham et al. [158] applied reinforcement learning on a similar navigation task, where the UAV had to learn to arrive at a desired position from an arbitrary starting point by the shortest way possible. To accomplish this mission, they combined a PID approach

with the Q-learning algorithm [206], in which the agent estimates an optimal value function and records it into a tabular database. As input, the algorithm gets the linear and angular velocities of the UAV together with its relative position, received from a motion capture system. For the initial training the authors use the MATLAB simulator [128]. After the desired behaviour is learned, the training is being continued in real-world setting. Using a Parrot AR Drone 2.0 quadrotor, they compare the results of both experiments performed in simulated and real-world settings. As a result, they show that the UAV successfully learns to navigate in an unknown environment, without the need of having any knowledge of its mathematical model.

Another work presented by Faust et al. [43] addresses the task of aerial cargo delivery. The primary mission is split into two subtasks: (1) learn a flying policy for an autonomous navigation in an obstacle-free environment, (2) the manipulation of a suspended load. For the first subtask, the authors use the approximate value iteration algorithm (AVI) [40], which allows the quadcopter to learn a policy using its location data. For solving the second subtask, the authors apply a model predictive control [97]. They evaluate their results both in a simulation and on a physical experimental drone. They showed that the proposed method attains proper tracking and load-displacement characteristics for a reasonable amount of training time.

Pham et al. [40] proposed a method for generalising an autonomous navigation task. This makes their learning approach applicable for any commercially available drone. As input, they use the data received from the drone's standard sensors, such as a GPS and camera. They apply DQN [131] as a learning algorithm that outputs the best next action. After experimenting in the AirSim open source simulator [177], they show that their approach can learn an autonomous navigation task in any unknown environmental settings.

Sugimoto et al. [190] and Yijing et al. [215] offered approaches for hovering, path learning and obstacle avoidance tasks. In the first two studies, Q-learning has been used as a learning approach. When learning a hovering control, the authors carry out the experiments on an AR.Drone, where the drone learns a stable hovering task using its onboard controller. Using the camera of the UAV, and a marker placed on the floor, the drone learns to hover above the marker. The second study focuses on a path planning task while avoiding obstacles. They adopt trap-escape strategy, which is based on an Adaptive and Random Exploration (ARE) approach [4], which guides the UAV flight to a proper path. They perform experiments in a simulator using different scenarios and show that their approach can effectively guide a UAV to reach a goal position by following an optimal path. Another pioneering research in this field is the work presented by Bou-Ammar et al. [16] addressing the problem of precise and fast stabilization of a quadrotor UAV. They propose a nonlinear autopilot based on a feedback linearization. Using value iteration method, after each generated control action, the agent receives a feedback regarding the taken action. Performing experiments in the MATLAB simulated environment they show that the agent learned the given objective in a short amount of training time. A novel method for dynamic stabilization of a UAV, from upside-down throws, is introduced by Hwangbo et al. [69]. They use a deterministic policy optimization approach based on a natural gradient descent that maps the state to a rotor thrust. To demonstrate the performance of the trained policy, they first use a simulated environment and later perform experiments on a physical drone. Eslamiat et al. [41] and Li et al. [104] presented a two-level optimisation approach for a trajectory generation problem in a simulated environment for multirotor UAVs. They as well use DQN as their policy learning approach, defining the learning task as a waypoint planning for avoiding obstacles with minimum control trust. At the end they conduct experiments in a

simulation and present that their approach achieves better performance compared to a baseline PID controllers.

Lastly, for solving one of the major issues in this domain, which remains to be the safety, Zhang et al. [220] combined model predictive control (MPC) [25] with RL. Their approach helps to reduce the catastrophic failures at the training time. They propose an off-policy guided policy search algorithm, where the MPC is used for the initial training of the control policies. Furthermore, authors apply reinforcement learning for the final training of the agent. Such approach transforms the RL to a supervised learning task. In a simulated environment, the authors demonstrate the effectiveness of their method.

Robotic Manipulations

Robotic manipulators are another robotic systems that are widely used in many real-world applications. Hence, applying deep reinforcement learning on the such complex systems became another area of interest for many researchers in recent years. The applications of robotic manipulations are ranging from object grasping to motion control of underactuated manipulation. For instance, Kim et al. [90] applied the SARSA on-policy RL algorithm [149] on a 3-DoF manipulator for learning a motion control policy. For experiments, they implemented a three-link planner arm and tested for positioning the end-effector on a given static point and after randomizing the goal points. The experimental results showed that the robotic manipulation is able to complete both missions successfully. Mahmood et al. [115] aimed to find out if an off-the-shelf implementation of a standard RL algorithm can produce effective and reliable results when applying on a real robotic manipulator. Hence, they designed a Reacher task with UR5 lightweight robotic manipulator, which is a flexible industrial robotic arm consisting of six joints. They aimed to design the learning task to be as similar to the OpenAI Gym Reacher's [142] task as possible, which mainly aims to learn a motion policy by directly controlling the torques of the manipulator. Based on their experiments, they made the following hypotheses that are crucial when applying learning on real-world robots:

- The computational delays can be detrimental to the learning.
- The network communication may have a significant impact on the robot's performance.
- Too small and too long action cycles can obstruct the performance.

A cooperative learning approach has been presented by Gu et al. [59]. The authors use multiple robots for learning a single policy with a novel asynchronous DRL algorithm. They show that by parallelizing the learning process across multiple robots a complex manipulation task, such as a door opening, is possible to learn with only a few hours of training. Also, it is possible to decrease by adding more learners. Some other works have been done by Peters et al. [150] and Park et al. [145]. The authors presented an effective learning of different motion policies of two link robotic arm by applying the natural actor-critic method.

Inspired by the advances and successes of DRL approaches using raw image data [131, 132], a significant amount of studies applied similar methods on learning visual-based control policies of robotic manipulations. However, these works remain out of scope of this paper, therefore, we provide only a few references related to these researches [78, 162, 9, 33, 164].

Flying Manipulator

The investigations and above discussions pointed out that DRL approaches have gained vast popularity among different domains of robotic applications because of their easy-to-design control techniques. The next point of interest of this work is to explore another, relatively new robotic systems, known as flying or aerial manipulations. Flying manipulations are the combination of two complex robotic platforms, such as UAVs and robotic manipulations, with highly nonlinear dynamics. Such hybrid aerial robotic platforms have a potential of being deployed in a broad spectrum of real-life applications ranging from videography to bridge inspection. However, designing control policies for flying manipulations are very challenging. Although, many researchers have proposed solutions to this problem, the majority of solutions are based on the traditional classic control algorithms [75, 24, 109, 12, 213, 80, 81]. Our investigations have shown that at this point of time there are no studies that applied deep reinforcement learning on a flying manipulator. However, below we will discus about a few works that tackled similar research area.

In [45] Figueroa et al. propose a reinforcement learning approach for balancing an inverted pendulum attached on a UAV. The authors split the main task into two subtasks, referring as (1) Initial Balancing and (2) Balance and Hover. In the first phase, the pendulum learns to stay closer to the upright position, when in the second phase the UAV learns to hover while maintaining an inverted pendulum at same place. Their experiments performed in a simulated environment have shown that the UAV learns to reduce its velocity rapidly to zero for maintaining the pendulum at its desired position. Another study, presented by Palunko et al. [144], addresses the task of suspended load manipulation attached on an aerial robot. Using the least squares policy iteration (LSPI) reinforcement learning algorithm [100] the agent learns to follow different trajectories while having a suspend load attached on its center of mass. It is worth noting that no other work has been found in the literature aiming to apply (deep) reinforcement learning-based control policy to a UAV integrated with a manipulator.

2.3 Theoretical Framework

As mentioned before, in this work, machine learning has been used as a key tool in different robotic applications, ranging from hardware-related degradation identification to the development of flexible and straightforward controllers for aerial robotic tasks. Thus, before presenting the proposed solutions, we first provide a detailed background on the fundamentals of the core concepts and algorithms used in the development of these techniques.

Machine learning (ML) is one of the primary subfields of Artificial Intelligence (AI). The term "Artificial Intelligence" first has been mentioned back in 1956, by computer scientist John McCarthy [209]. He defined it as the "science and engineering of making intelligent machines, especially intelligent computer programs" [209]. On the other hand, machine learning provides machines the ability of learning from data, making predictions and decisions. The ML methods can be categorized as supervised learning, unsupervised learning and reinforcement learning, as represented on figure 2.1.

The algorithms that use labelled data for performing a classification or regression task are referred to as supervised learning methods. When no labelled data is available, and the algorithm outputs results based on correlations of patterns in

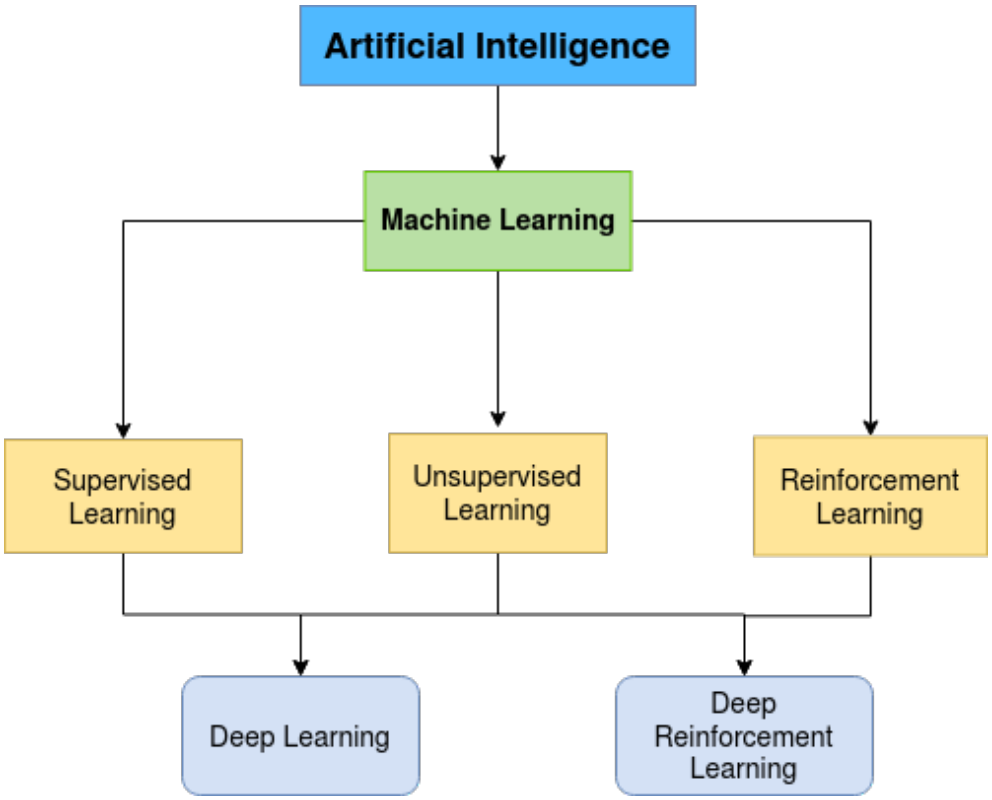


FIGURE 2.1: The categorisation of AI subfields.

the input, then it is defined as unsupervised learning. Reinforcement learning differs from these methods. Essentially, it is a black-box, goal-driven learning where the agent learns to perform a task by interacting with its environment, receiving a numerical reward signal as feedback. The agent learns to take actions that lead to greater cumulative rewards. The representation of a classic reinforcement learning workflow is shown on figure 2.2.

Lastly, deep learning and deep reinforcement learning are relatively novel methods that introduced neural networks into the machine learning subfields. For instance, deep learning is about learning multiple levels of representations and abstractions that help to make sense of data, such as images, texts and sounds. Deep learning algorithms can be either supervised or unsupervised, where deep neural networks are used as function approximators. In contrast, deep reinforcement learning combines deep neural networks within RL algorithms providing a powerful framework for software-driven agents or physical robots to learn control behaviours by interacting with a black-box environment.

The solution that has been proposed for solving a degradation identification problem of a UAV, is based on a supervised learning techniques. From other side, for designing flexible control solutions for aerial robots independent of the model dynamics, the techniques of model-free deep reinforcement learning have been used. Thus, below we mainly focus on introducing the underlying concepts of these methods.

2.3.1 Supervised and Unsupervised Learning

Supervised, or predictive, learning attempts to fit input data to previously learned labelled data [96]. Here, the algorithm is trained with a carefully chosen and crafted

data set that is correctly labelled. The algorithm uses this knowledge as a unique basis to analyse new data and compute the inferred label [6]. Supervised learning itself consists of two main techniques: *regression* and *classification*. Regression predicts a continuous target variable and classification predicts which category or class it belongs to [6]. Both techniques can be used together to perform complex predictions.

Unsupervised learning is more challenging than supervised learning as there is no training data available. Unsupervised learning tries to find hidden structures and discover patterns in unlabelled data by analysing regularities and/or frequencies in the input data. The processed data is summarized and key features are presented using, for example, clustering methods [96] such as k-means, hierarchical clustering or other density based algorithms. Most often, this type of learning can be applied to image compression, bioinformatics, and so on.

2.3.2 Reinforcement Learning

The underlying concept of RL has been adopted from behavioural psychology [194], where the agent learns an optimal control policy throughout interactions with its environment. The reinforcement learning framework consists of a few main elements:

- **An agent:** the learner and decision-maker is called agent.
- **An environment:** the space where the learning procedure takes place is the environment.
- **Policy:** the policy defines the way that the learning agent behaves at a given time. It is a mapping from perceived states of the environment to actions to be taken when in those states [194].
- **Reward function:** it defines the objective that the learning agent aims to achieve. At each step the agent receives a reward indicating the intrinsic desirability of the state.
- **Value function:** it quantifies the quality of a given policy. The value of a state is the total amount of rewards an agent can expect to receive over the future, starting from that state and following policy.

At each state, the agent takes an action and transitions to a new state receiving a feedback from the environment in form of a reward. The main objective of the agent is to learn an optimal policy that maximizes the cumulative reward.

The classic reinforcement learning workflow represented on figure 2.2 can be described as a Markov decision process (MDP) consisting of following elements:

- a finite set of states S,
- a finite set of actions A,
- a transition probability model $p(s_{t+1}|s_t, a_t)$,
- a reward function $r(s_t, a_t, s_{t+1})$, which maps the state-action pair to a set of real numbers,
- ρ_0, a distribution of the starting states s_0,
- a discount factor $\gamma \in (0, 1)$, where the lower value emphasises an immediate rewards.

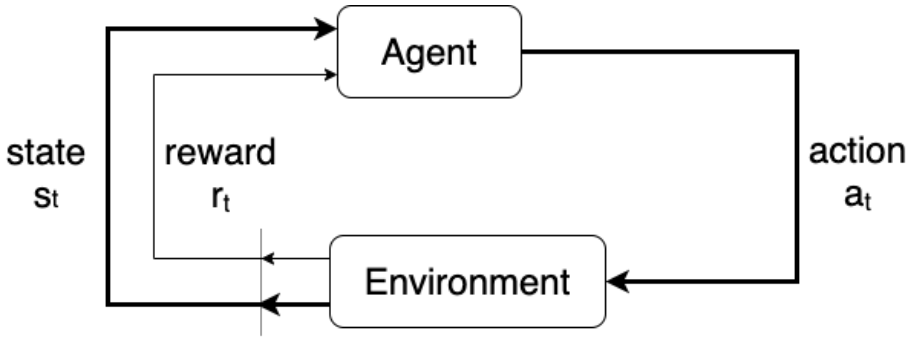


FIGURE 2.2: The workflow of a classic RL setup. It consists of an agent being in a state $s_t \in S$, performing an action $a_t \in A$ and receiving a reward $r_t = R(s_t, a_t)$ at timestep t. Based on the current state and chosen action the agent transients to a new state s_{t+1} [194].

The main idea behind the procedure of an MDP is as follows: an agent starts its interactions at an initial state $s_0 \in S$, takes an action $a_0 \in A$. Based on the taken action and the transition probability $p(s_1|s_0, a_0)$ the agent transitions to a new state s_1 and receives a reward of $r(s_0, a_0, s_1) \in R$.

If the MDP is episodic, the state is reset after each episode of lenght T and process repeats. The main objective of the agent is to discover an optimal policy π^* that maps states to actions by maximizing the expectation of the accumulated reward $R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$, which is the discounted sum of the rewards in one whole episode, and T is the horizon, representing the length of one episode.

The policy can be either deterministic $\pi : S \to A$, or stochastic $\pi : S \times A \to [0, 1]$, where $\pi(a|s)$ is the probability of choosing an action a being at state s. To find the optimal policy π, a value function $V_\pi(s)$ or an action-value function $Q_\pi(s, a)$ is used.

A value function is defined as the excepted sum of rewards that the agent will receive while following a particular policy π from a particular state s. The value function $V_\pi(s)$ for policy π is as follows:

$$V_\pi(s) = \mathbb{E}[R_t|s_t = s]. \tag{2.1}$$

An action-value function, also known as Q-function is the expected sum of rewards for taking action a being in a state s when following policy π [105]. The action-value function $Q_\pi(s, a)$ for policy π can be written as follows:

$$Q_\pi(s, a) = \mathbb{E}[R_t|s_t = s, a_t = a] \tag{2.2}$$

The optimal value function is the one that obtains the maximum state value for any state s under any policy, defined as

$$V^*(s) = \max_\pi V_\pi(s).$$

Similarly, an optimal action-value function is the one that has the maximum value over all policies:

$$Q^*(s, a) = \max_\pi Q_\pi(s, a).$$

2.3.3 Deep Reinforcement Learning

Although numerous reinforcement learning algorithms made significant breakthroughs in various challenging decision-making problems, often, these proposed

solutions are limited to discrete state and action spaces, when most of the real-world robotic applications require a continuous state and/or action space setup. Thus, in order to eliminate these limitations, function approximators have been introduced into reinforcement learning. One of the most commonly used function approximators are deep neural networks. Their combination with reinforcement learning resulted in a powerful tool called deep reinforcement learning (deep RL), that increased the usage of the reinforcement learning-based methods in different real-life scenarios.

Below we describe a few of the most fundamental concepts of reinforcement learning.

Deep Neural Networks

Deep Neural Networks (DNNs) are the driving force behind many successes of reinforcement learning. DNNs provide a mathematical framework which represents the process that is loosely based on how the human brain learns [18]. As the human brain consists of millions of interconnected neurons, NNs as well contain large amount of interconnected artificial neurons, called units, arranged in layers.

Each neural network consists of at least two, input-output layers, but often more layers are added between these layers, known as hidden layers. As an input data is received, it is processed using weights and biases, resulting in an output. In case of multi-layer neural networks, the output of one unit is further fed as an input to another neuron. Figure 2.3 shows a simple example of a two-layer neural network [31].

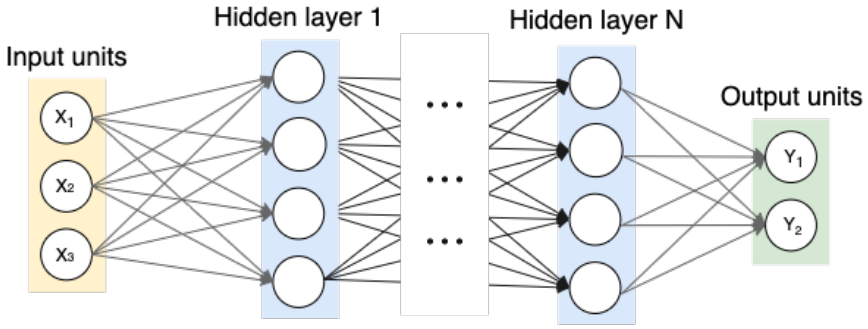


FIGURE 2.3: A general structure of a deep neural network. The training observations are fed through X_1, X_2, X_3 input units. Then they pass through the intermediate layers between the input and output layers. In general the intermediate/hidden layers help to learn more complicated relationships involved in data. Finally, the final output, Y_1, Y_2, is extracted from previous two layers. The output units may varies from task to task [31].

The connections between neurons are called synapses. A weight in form of a real number is assigned to each synapse in the network, representing their significance. Once the data is fed into the input layer, each input unit is being multiplied with a weight assigned to its synapses. Afterward, a bias is added to the sum of the input vector and fed into an activation function, from which the value of an output unit is computed [62]. As mentioned above, when the neural network consists of more than two layers, this output becomes an input to another layer of units. Figure 2.4 represents the operation of a very basic neural network, known as perceptron.

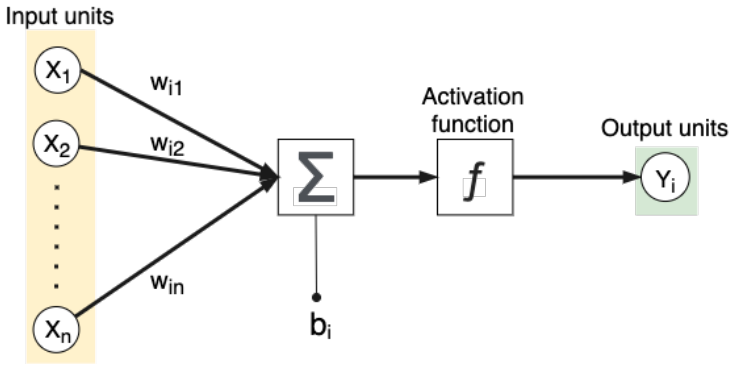


FIGURE 2.4: The workflow of a single unit, where $X_1, X_2, \ldots, X_n$ are the input units, $w_{i1}, w_{i2}, \ldots, w_{in}$ are the weights on each input unit, b_i is a bias, $f(\cdot)$ is the activation function, Y_i is the output.

Finally, the complete equation for a single unit can be written as follows:

$$Y_i = f(\sum_{i=1}^{n} w_{ij} \times X_i + b)\,, \tag{2.3}$$

where j is the number of inputs to the units.

In general, the activation functions are used to introducing non-linearity into the deep neural networks. These include:

- Sigmoid:
$$f(x) = \frac{1}{1+e^{-x}}\,.$$
- Hyperbolic Tangent:
$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}\,.$$
- Rectified Linear Unit (ReLU):
$$f(x) = max(0, x)\,.$$

Because of its ability to enable fast and robust training ReLU has been the most frequent choice between many researchers when training deep neural networks [3].

There exist a few other mathematical operations that are often applied for improving the computations and the performance of the network. For instance, pooling operation is used for reducing the dimensionality of a feature map, helping the network to be robust and invariant to small shifts and directions. To speed up the training and improve the accuracy of the neural network, a normalization of the data is often performed across the input layers.

As was mentioned above, in the beginning of training, the weights are randomly initialized. Then, they are being adjusted throughout whole training using a gradient decent algorithm in a way that minimizes the error rate. This is called backpropagation, since the weights of the output layer is first updated, and then backpropagated into the network [27].

Finally, to prevent the over-fitting problem in large neural networks, a dropout technique can be applied. Dropout is a regularization method, which randomly drops neurons along with their connections from the neural network during the

training. Dropping out neurons means temporarily removing them from the network. Thus, at the time of training, dropout samples from a certain number of "thinned" networks which consist of the neurons that survived the dropout [187].

In general, there exist wide variety of architectures of deep neural networks. Some of the most commonly used ones are Feed-Forward Neural Networks [197], Recurrent Neural Networks (RNN) [192], Long Short-Term Memory networks (LSTM) [198], and Convolutional Neural Networks (CNNs) [91]. The choice of the DNN type depends on the application and the input data that needs to be analysed by the network, which may range from pixels of an image to a numerical representation of the states of some system.

Reward function

The reward function is one of the most fundamental and challenging components in reinforcement learning and has a decisive impact on the learning process. In any reinforcement learning task, the agent learns the desired policy by interacting with its environment and observing the feedback signals that it receives in form of rewards and/or punishments. Hence, a proper design of the reward function yields better performance and faster convergence of the algorithm.

In general, the reward function is the means by which the engineer describes the desired behaviour that the agent should learn. Thus, poorly designed reward functions cannot only hinder the convergence of the model but also guide the agent to learn an unexpected sequence of actions, if it leads to maximizing the cumulative reward. Such performance of an agent is called reward hacking [57], which describes the process when the agent finds shortcuts to achieve its only goal of obtaining the highest reward possible. Therefore, careful crafting of the reward function that adequately represents the desired behaviour is crucial.

In a sparse and underspecified reward settings, the agent receives no or limited feedback until it enters a small subset of the environment space, known as goal state. However, discovering trajectories that attain accidental success is very challenging, it may either increase the exploration time dramatically or even result in divergence of the algorithm. One of the possible ways to tackle this problem is through reward shaping [20], where the reward function is designed in a way that guides the agent towards finding its goal state [140]. Other reward shaping-inspired methods are potential-based reward shaping [34] or inverse reinforcement learning [139, 48]. Although these methods have the promise of providing faster convergence to an optimal policy, they as well have some problems as traditional reward functions, that is the difficulties to capture complex behaviours. Fortunately, in recent years many researchers have focused their attention on solving this issue. For instance, [166] proposes a novel divide-and-conquer approach that enables the designer to specify a reward separately for each environment. A hierarchical learning approach [60] presents whereby using the Concept Network Reinforcement Learning, the authors use multi-level hierarchies and decompose the learning problem into different subproblems. Hierarchical reinforcement learning [136] is another promising approach that enables engineers to solve more complex decision-making tasks.

Exploration vs Exploitation

One of the well-known tradeoffs in reinforcement learning is the exploration-exploitation dilemma. In order to avoid being stuck at local maximum, the agent should find a balance between exploration and exploitation. **Exploit:** the agent

chooses actions that it has already attempted in the past (from the history of trials) that maximized the cumulative reward and proved to be the most efficient. **Explore:** the agent performs explorations to gather more information by selecting new actions. This helps to find new states that may improve the policy in the future. Aiming to solve this dilemma, different approaches have been proposed [29]. Upper confidence bound (UCB) [7], Softmax [193] and ϵ-greedy [208] are some of the most commonly used strategies.

On-policy vs Off-policy learning

In reinforcement learning the optimal policy can be found either by off-policy of on-policy learning. *On-policy* methods use the current policy to generate actions and use it to update the current policy while *off-policy* methods use a different exploratory policy to generate actions as compared to the policy which is being updated. SARSA [149] and Q-Learning [207] are two well known on-policy and off-policy algorithms.

Model-free vs Model-based RL

In general, reinforcement learning can be split into two main categories: model-based and model-free. Model-free RL tends to emphasize the learning, meaning that the agent directly learns a value function, policy function or both, by interacting with its environment. On the contrary, in mode-based RL, the model of the environment is either given or learned from experience.

Both methods have their advantages and disadvantages. For instance, applying model-free RL methods for solving problems in robotics are more convenient, since these methods are capable of learning desired behaviours without requiring prior knowledge about the environment. However, often the high sample complexity obstructs the usage of model-free methods in real-world settings. In contrast, the model-based methods reduce the sample complexity due to the availability of the model of the environment. Therefore, when it comes at applying reinforcement learning approaches in real-world problems, model-based methods have shown to be more efficient. However, the need of having greater knowledge about the environment limits its usage in many robotics applications, since often the agent has partial or no knowledge about its settings. Moreover, learning a policy is often easier than learning a model of the environment, which makes model-free RL more amenable to large scale problems.

In this work, our primary intention was to design a flexible control solutions for aerial robots operating in a continuous state-action domain using model-free deep reinforcement learning techniques. We also aim at understand their capabilities as well as limitations. Moreover, we aim to provide further insights for eliminating these limitations.

2.3.4 Model-free Reinforcement Learning

In the literature, model-free RL methods are split into three main classes: (1) **value function-based** (actor-only), (2) **policy search** (critic-only) and (3) **actor-critic**, which is the combination of the previous two methods. Below we describe each method individually.

Value function-based methods

The value function is one of the key concepts in model-free reinforcement learning. The agent seeks to find an optimal value-function $V^*(s)$ or action-value function $Q^*(s,a)$ from which an optimal policy π^* is derived. For estimating the value function the Temporal-Difference (TD) Learning method is often used. TD learning learns value function $V(s)$ by calculating the TD error, which is the difference between the new and old estimates of the value function. The update rule for the value function is given by

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]\,, \tag{2.4}$$

where α is a learning rate, r is the reward received at the current timestep, s' is the new state, s is the old state and $r + \gamma V(s') - V(s)$ is the TD error.

There are two temporal difference methods, that have been widely used to solve different RL problems. SARSA (State-Action-Reward-State-Action) [149] on-policy and Q-Learning [206] off-policy temporal difference algorithms that aim to derive an optimal policy by estimating the action value function. The main difference between these two algorithms is the update rule used to update the action-value function. In SARSA at each timestep the action-value function is updated using the following rule

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]\,, \tag{2.5}$$

when in Q-Learning algorithm, the action-value function is updated as follows

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]\,. \tag{2.6}$$

Finally, from an optimal action-value function $Q^*(s,a)$ an optimal policy $\pi^*(s,a)$ is derived, $\pi^*(s,a) = \operatorname{argmax} Q^*(s,a)$. Because of its simplicity and the ability of successfully obtaining an optimal policy, the Q-learning algorithm has been used in many different applications. However, its applicability is limited to domains with low-dimensional state spaces.

To reduce some of the limitations of Q-learning method, different variations of the algorithm have been proposed in [54, 201, 63, 146]. For instance, Mnih et al. [132, 131] proposed Deep Q-Networks (DQN) algorithm, which is an extended version of Q-Learning method. By introducing deep neural networks as function approximation in the Q-learning algorithm they demonstrated that the novel approach can successfully learn to control policies directly from high-dimensional sensory input. They apply DQN on number of Atari 2600 games and show that the algorithm can achieve human-level control measured by human normalized scores [11].

Policy search methods

Policy search methods tend to directly search for an optimal policy using either gradient-free or gradient-based methods. The gradient-free policy search [203, 167] is a black-box optimization which is used as an alternative approach for solving RL problems. However, gradient-free policy search methods are outside of the scope of this work, thus, only gradient-based methods will be further discussed.

Over the past few years, gradient-based policy search methods, also known as policy gradient algorithms, gained increased popularity and are perhaps one of the most broadly used approaches for solving reinforcement learning tasks in continuous action-space settings.

As it was stated above, in reinforcement learning, an agent being at a state s_t at time t, takes action a_t and transitions to a new state s_{t+1} according to policy $\pi(a_t|s_t)$ and receives a reward r_t. The main objective of the agent is to maximize the cumulative discounted reward, denoted as $J(\pi) = \mathbb{E}[r_t|\pi]$. In policy gradient methods, the main objective remains the same, but they parametrise the policy $\pi_\theta(a|s)$ and aim to find values of the parameter θ that maximize the cumulative discounted reward $J(\pi_\theta)$. Thus, at each step-size, the policy parameters are updated following the REINFORCE [152] rule:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\pi_{\theta_t}) \tag{2.7}$$

where α is the step size, often called as learning rate, and $\nabla J(\pi_{\theta_t})$ is the gradient of the performance with respect to the parameter vector θ. The policy-gradient theorem given in [196] provides the gradient of $J(\pi_\theta)$ with respect to the parameters θ of policy π_θ as follows:

$$\nabla J(\pi_\theta) = \int_S \rho_\pi(s) \int_A \nabla_\theta \pi_\theta(a|s) Q_\pi(s,a) da ds = \mathbb{E}_{s\sim\rho_\pi, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_\pi(s,a)] \tag{2.8}$$

where $\rho_\pi(s)$ is the state distribution induced by the policy π. For estimating the action-value function, often the actual cumulative rewards are used as an approximation for each $Q_\pi(s,a)$ [182].

Based on policy gradient theorem, later, Silver et al. [182] proposed a deterministic policy-gradient (DPG) algorithm for estimating gradient. They demonstrate that their approach is more efficient than the usual stochastic policy-gradient method. Another gradient estimate is proposed by Kober et al. [94] called Reward-Weighted Regression algorithm. It uses the Expectation-Maximization problem and formulates the policy optimization without the need of manually choosing the learning rate.

Actor-Critic methods

Up till now, all the discussions were around either value-based or policy-based methods. In the literature, those methods are often referred to critic-only and actor-only. In the last few years, the algorithms that combine the advantages of both techniques gained significant popularity in deep reinforcement learning domain, especially in the continuous action-space settings.

The actor-critic methods consist of two main objects: an "actor" which represents the policy, and a "critic", representing the value-function, respectively. The main difference of this method is that the action-value function is estimated by another network, called "critic".

Numerous actor-critic methods have been proposed over the past few years that have been widely used in many robotics applications. For instance, Lilicrap et al. [106] presented the Deep deterministic policy gradient (Deep DPG) off-policy algorithm, which combines the actor-critic approach with insights from Deep Q-Networks [132], to solve simulated physics tasks and it has been widely used in many robotic control tasks. It uses two neural networks, the actor network learns a deterministic policy and the critic network approximates the action-value function of the current policy [103]. In some policy gradient algorithms a trust region technique has been incorporated, which bounds the optimization step of the policy

update and helps to prevent extensive changes between the previous and current policies, ensuring monotonic improvements in the policy performance. Trust region policy optimization (TRPO), and Proximal policy optimization (PPO), proposed by Schulman et al. [176, 175], are relatively robust on-policy algorithms, for high-dimensional, continues action-space domains. TRPO aims to optimize a surrogate function, which is the lower bound of the expected cumulative reward. It measures the KL-divergence between the current and previous policies restricting the updates of the policy to be inside the predefined bounds. By applying constraints on the size of the policy updates, it guarantees that the changes of the state distribution to be in the range of the trust-region, preventing the policy from drastic alterations. In contrast, PPO introduces a clipping mechanism which reduces the complexity of the implementation of TRPO algorithm. PPO pushes away the old policy if the probability ratio between the current and old policies is out of the clipping range, which helps to improve the stability for learning an optimal policy.

Because of their effectiveness of searching for optimal policies using low-variance gradient estimates, actor-critic algorithms have been widely used in many real-life applications [55].

Selecting the learning method

Nowadays, the variety of approaches often create great confusion when it comes to applying them to real applications. For instance, in ML-related forums, one of the most commonly asked question is "How to know which class of algorithms will yield the best learning of an optimal policy?". Although there is no best answer to this question, considering the following facts may facilitate the selection process. First of all, we should answer the following questions: "What type of behaviour the agent tries to learn? Is the state-action space discrete or continues? Is it possible to model the task by an MDP?". If the agent's task is to learn an optimal policy in a discrete space-action environment, then the value-based (critic-only) methods are the best suited. On the contrary, for continuous state-action spaces, the trade-off is between policy-based (actor-only) and actor-critic methods [151]. Here, the choice depends on the MDP model. If the environment can be described as a stationary MDP then actor-critic methods will provide the best results.

2.4 Conclusion

The main focus of this chapter was to present the potential of machine learning in robotics applications. As well, investigate and provide in-depth insights on the successful studies over the past years, mainly targeting applications, such as anomaly detection and learning complex motor skills for unmanned aerial robots, robotic manipulators and flying manipulators.

Although traditional control methods have shown remarkable performance in many robotic applications, and have been predominately applied over the past decades, they often suffer from numerous limitations. For instance, due to their highly parametric models, the development of more general behaviours can be challenging, since a significant amount parameter need to be hand-tuned. Therefore, they often fail when exposed to unknown dynamics or unstructured environments. From the other side, machine learning provides a framework for developing optimal control policies without making any assumption about the robot's dynamics or the model of the environment. Additionally, machine learning can be very effective

for improving a safety system of robots because of its ability to learn from data and detecting abnormal patterns in high dimensional data.

In the upcoming chapters, we present the research conducted in this thesis, and the implemented approaches that apply machine learning for solving the problems defined on chapter 1.

3

Online Degradation Identification of UAV using Machine Learning techniques

Heretofore, we presented a brief history of robotics and introduced the kinematics and dynamics model of a flying manipulator. Then, we have discussed a set of machine learning methods addressing problems within the scope of anomaly detection of UAVs hardware-related components, along with learning complex motor skills.

This chapter presents a lightweight supervised learning-based approach for monitoring the behaviour of a UAV during its missions. It observes patterns in the flight data identifying anomalous behaviours and provides an estimate of the level of severity of the damage. One of the main objectives of this approach is to improve the autonomy of a robot, and help to prevent unexpected and undesirable incidents.

Parts of this chapter have been published in:

- Manukyan, Anush, et al. "UAV degradation identification for pilot notification using machine learning techniques." 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE, 2016. APA
- *Manukyan, Anush, et al. "Real time degradation identification of UAV using machine learning techniques." 2017 International Conference on Unmanned Aircraft Systems (ICUAS). IEEE, 2017.*

3.1 Introduction

Navigating autonomously in an unknown and dynamic environment can be challenging for mobile robots, especially for unmanned aerial vehicles, often leading to unexpected consequences, such as damages on hardware-components or even sudden catastrophic accidents. Especially in these days, when robots are equipped with agile controllers, that quickly adapt themselves to occurred subtle behavioural changes of the system, makes it even harder to foresee the possible software or hardware degradations in the early stages. Hence, it is greatly desired to have a real-world technique that can continually monitor behaviour of the robot and automatically detect incipient failures. Methods of such kind may improve the autonomy of the robot, provide more reliable and safe accomplishment of the task and reduces the risk of unexpected breakdowns.

As we discussed in chapter 2, there were many efforts devoted to addressing this problem. However, the majority of the proposed approaches mainly focus on finding anomalies in the data gathered by different sensors of the robot. But, finding only abnormal patters in the flight data may not be enough, since it might result in wrong assumptions. Hence, in this work, we present a lightweight approach that aims to identify hardware-related degradation and point out the severity level of the damage. A typical scenario consists of a drone flying autonomously in an arbitrary environment following a pre-determined or job-specific path. During this mission, the flight data is being recorded by internal and/or external sensors or by any other movement tracking system. The captured data is in a raw format and, thus, must be pre-processed by performing various feature extraction techniques, such as reducing the dimension or even running format conversions methods in order to facilitate the next computational steps in the model. After these first steps, the processed data should be stored in a convenient file format or database, easily accessible for further analysis. Thus, the proposed approach consists of two phases: 1) data gathering and offline model learning, 2) online degradation identification process using the sliding time window technique.

In this work, the term online refers to the data being evaluated as soon as it is being received. On the opposite, an offline process uses data that has been gathered in advance.

3.1.1 Offline Model Learning

The first phase starts with a UAV following a predefined flight path. During the entire flight, three-dimensional coordinate data captured by a high frequency and precision movement tracking system, along with the data streamed by internal sensors of the drone is being gathered and stored.

Having a robust training dataset is essential for obtaining accurate model. Thus, the flight scenario is being entirely run for several times, while the hardware-components of the UAV are manually degraded. The degradations are applied gradually, to instigate different behaviours of the drone.

Finally, all collected datasets are split into time windows of predefined length and labelled semi-automatically. More details on the labelling process is given in section 3.3. Once the data is labelled, the next step is to train and validate the model offline, and find the best parameters used by the classification algorithm for achieving its best performance.

3.1.2 Online Degradation Identification Process

Now, when the model has been trained offline, it can be used online to identify degradations of the UAV. During the mission of the UAV, flight data is being recorded by internal and/or external sensors and the data is streamed to the degradation identification algorithm.

As data comes in, the current time window is filled. In this approach, we use a window-based on duration and not based on the amount of data. This means that after a fixed time interval (*e.g.,* 10 seconds) the current window is closed and sent asynchronously to the machine learning algorithm, which then estimates the behavior of the UAV for the given window. Since we are working on a continuous stream of data, the algorithm predicts the level of degradation in the background while the server already records the incoming stream of data for the next time window. Figure 3.1 presents each step of online degradation identification process, using offline trained model. The online prediction capabilities depend on the amount of data to

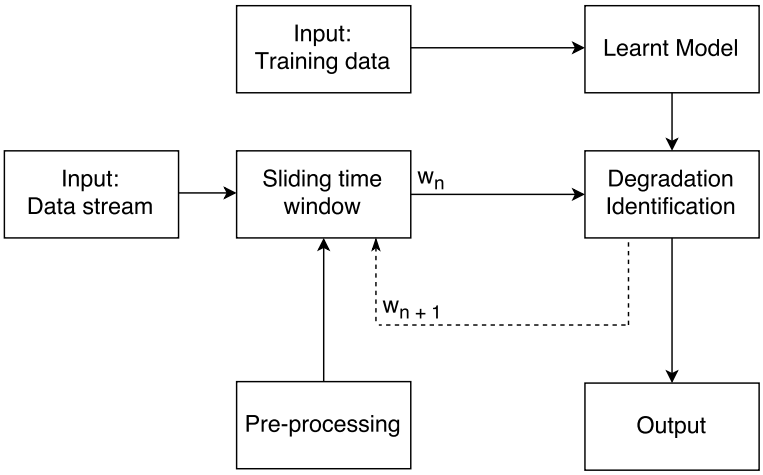


FIGURE 3.1: General flow of online degradation identification process.

process for each window and the speed of the prediction algorithm. Because the prediction is run once on each time window, the estimated level of degradation can only be updated every x seconds, x being the size of the time window. Ideally, the computation of the prediction should be fast enough that it yields an estimation before the next time window is reached.

It is worth mentioning, that the streamed data arrives in a raw, unprocessed format, and can contain many unnecessary feature attributes and/or many repetitive data points. This is especially the case when the data is being streamed by high-performance sensors. Therefore, the stream of data should go through some kind of pre-processing. One approach is to pre-process the outgoing stream form the perspective of the UAV. This can be achieved by reducing the frequency of the streamed data. Another way consists of recording the entire stream and only pre-processing the current time window before running the prediction algorithm. The aim of pre-processing data streams is to apply several feature extraction techniques in order to reduce the dimensions of the data and facilitate the next computational steps.

3.1.3 Theoretical Framework

Following the general model description and learning, here we present the core algorithms and techniques that are used and which have been implemented as a proof

of concept on real UAV flights.

The main objective of the degradation identification process is to categorize the output based on the learned samples, which is known as a classification job.

Although there exist a large number of machine learning algorithms, in this research, as classification algorithm the k Nearest Neighbours (kNN) [83] has been used. The choice of the algorithm was motivated by its ability of efficiently handling time series data as well as its simplicity and high-performance. Moreover, kNN allows the distance measure function to be easily adapted to the data specifications, in this case to the different lengths of time series. Thus, the Dynamic Time Warping [214, 165] has been selected, which computes the best possible alignment warp between two unequal length time series by selecting the one with the minimum value.

k Nearest Neighbours classification algorithm

Implemented in 1968 by Cover and Hart [83], k Nearest Neighbour remains among the most important and intuitive supervised learning algorithms for pattern classification [84]. At the core, kNN takes a set of labelled training samples and classifies test samples by the use of a similarity or distance metric. In other words, it predicts the label of new objects based on the k most similar training samples, better referred to as k nearest neighbours [84, 211].

Given a training set X, of size N with Y class labels and an unknown test sample x, the computational steps of kNN are as follows:

- **Step1:** Choose a number of nearest neighbours k.
- **Step2:** Compute the distance d between the test sample and each training samples, noted $d(X_i, x), i \in [1, N]$.
- **Step3:** Sort the distances and extract the class labels y_j, for the smallest j^{th} distances $(j \in [1, k])$.
- **Step4:** Compute a majority vote on the extracted labels and return the resulting winner, y'.

The majority voting rule assigns the class label to an unknown data object based on the most frequent labels among the k candidates extracted from the training set. The majority voting formula is the following:

$$y' = \underset{v}{argmax} \sum I\left(v = y_j\right) , \tag{3.1}$$

where v is a class label, y_j is the class label for the j^{th} nearest neighbour, and $I\left(\cdot\right)$ is an indicator function that returns the value of 1 if its argument is true and 0 otherwise [211].

As a similarity measure the Euclidean distance is often used. However, in some cases, depending on the data specifications it can be substituted by other distance measures, such as Manhattan, Minkowski, Hamming, Mahalanobis or DTW, to mention a few. In general, the Euclidean distance is used when the distance between two points needs to be measured. Given that our inputs are time series of different sizes, DTW has been chosen as a distance measure function which can compute the distance between two unequal datasets.

Dynamic Time Warping

Dynamic Time Warping is an algorithm to measure the distance of two time series, or temporal sequences, of different length. Given two time series $X = (x_1, x_2, ...x_N)$, $N \in \mathbb{N}$ and $Y = (y_1, y_2, ...y_M)$, $M \in \mathbb{N}$, sampled at unequal points in time.

The more similar X and Y are, the smaller the distance function $D(X, Y)$ is. Furthermore, the value of distance measure function increases as the time series differ more from each other [129]. In other words, the $D(X, Y)$ is a distance measure function which computes the best possible alignment or the minimum mapping distance between two time series, using a dynamic programming approach. It builds the warping path between time series and returns the distance value by following these tree main conditions:

Boundary condition. The first and last elements of X and Y must map exactly to the starting, respectively ending points of the warping path: $p_1 = (1, 1)$ and $p_K = (N, M)$.

Monotonicity condition. Elements of X and Y should stay to time-ordered: $n_1 \leq n_2 \leq ... \leq n_K$ *and* $m_1 \leq m_2 \leq ... \leq m_K$.

Step size condition. While mapping sequences, this condition is used for limiting the warping path from long jumps: $p_{l+1} - p_l \in \{(1, 1), (1, 0), (0, 1)\}$.

We note p^*, the best warping path between X and Y with the minimal total cost: $p^* = \text{argmin}(D(X, Y))$. This minimization problem is computationally difficult, since the algorithm has to calculate all possible warping paths, which number grows exponentially as the size of the time series grows linearly. Therefore, DTW uses a Dynamic Programming algorithm to find this minimum-distance warp path, which evaluates the recurrence of the cumulative distance that is found in the adjacent elements:

$$D(i, j) = d(x_i, y_j) + \min \begin{cases} D(i, j-1) \\ D(i-1, j) \\ D(i-1, j-1) \end{cases} . \tag{3.2}$$

Once the cumulative distance matrix is computed, we can use backtracking to find the minimum warping path. The last element of the warping path is the distance between two time series [130]. This matrix has complexity of $O(MN)$, thus in our implementation we decreased the complexity by limiting the warping window size.

3.2 Experimental setup

The full scenario, or mission, has taken place indoor, where for the entire experiments a motion capture system (OptiTrack [13]) has been installed. A commercially available drone (Ar.Drone [147]), shown in figure 3.2, has been used, and it has been programmed to follow a pre-defined path autonomously.

The total length of the scenario is $1m40s$, during which time the UAV performs different fast and slow basic manoeuvres such as take off, landing, forward/backward moves, right/left turns and hovering.

Since the interest of this work is to identify the degradations of the UAV's hardware components, the propellers have been chosen as the main target for being altered and damaged. Several incremental damages, such as bending, cutting and/or scraping has been made on the propellers. At each level of damage, the scenario has been fully run to observe and collect the flight data, which is later labelled and used

FIGURE 3.2: A commercially available drone (Ar.Drone 2.0).

as a training set. Figure 3.3 presents the different kind of damages that have been made on the UAV's propellers.

Although this work mainly targets on monitoring and predicting the level of degradation of propellers, this approach can be applied to any other hardware-component of the UAV, such as motors, batteries or even the frame.

Experiments have been video recorded, and one of the experiments is available online [127], which showcases the desired flight against the flight where all propellers were in their worst condition.

It is noteworthy that an informative training set is crucial for further predictions. Hence, applying many different levels of degradations on the chosen component and recording the flight data is a necessary step in the first phase, before the offline learning of the model can be performed.

3.3 Evaluation and Results

In this subsection, we first introduce the dataset used for the evaluation. Then, we describe the labelling process, and we conclude by presenting the obtained results along with the assessment of the model performance of the proposed approach.

Data

The data used in this work have been gathered from 50 flights performed during experiments. Because the scenario was played in a small environment, it has been possible to record the flight data using an external high-performance motion capture system, which allows recording the precise 3-dimensional location coordinates of the UAV at a rate of 240 readings per second. For all further analysis and prediction, this data has been used, due to its preciseness.

Eventually, the data gathered by such systems are high in volume and can decelerate computations. Thus, once the data is recorded, a pre-processing is directly applied by removing consecutive location points that are less than $0.01m$ apart from

FIGURE 3.3: Different form of damages made on propellers.

each other. This recuses the size of the dataset by 60%, without any important information loss.

Labelling

The labelling process is one of the essential steps before the degradation identification can be performed. The main steps of the labelling process are the following:

- choose the data of a flight where the UAV had close to ideal performance,
- compute the distance between the data of chosen best flight and all other flights using any distance measure function,
- manually analyse the distance values and set thresholds for further labelling.

In this work, the flight where the UAV still had four new propellers in perfect conditions has been chosen as a reference flight, often called desired flight. As a distance measure function, the DTW algorithm has been used [214], because of its ability to compute the distance between two time series of different lengths.

Before the data can be labelled, the entire training dataset is split into multiple time windows of an equal time interval. Then, for a given time interval, the DTW distance between the desired flight and each time window of the other flights is computed. After, manual analysis of DTW values is performed, and different thresholds are defined. In general, a smaller DTW value is an indicator of the hardware's good condition. In contrast, bigger value is the result of greater degradation. Finally, based on these thresholds all time windows are automatically labelled using a proof of concept software, implemented for validating the propose of this work [116].

It should be noted that the length of the time window can vary based on the duration of the mission. For instance, for the scenario presented in this work, the window size has been chosen to be 10 seconds. A larger time window size can help the model achieve higher accuracy since more data points will be available for the computations. However, a smaller time window length allows more frequent predictions.

To provide a better and clearer understanding of the manual analysis of the results, three random flights has been selected for the visualization of the UAV's behavior using DTW values. Figure 3.4 is a 2D representation of the z axis position (in meters) of each flight (shown as red, green and purple), along with the best desired flight (shown in blue), over time (in seconds) for time interval $[10, 20]$. We see that flight 3 is close to the desired flight with some small differences. Flight 1 and 2 are similar with each other, but are very different from the desired one. After computing the DTW values between each flight and the desired flight, we have the following distances for this example: Flight 1: 281, flight 2: 316, flight 3: 26. Those values confirm the similarities observed previously.

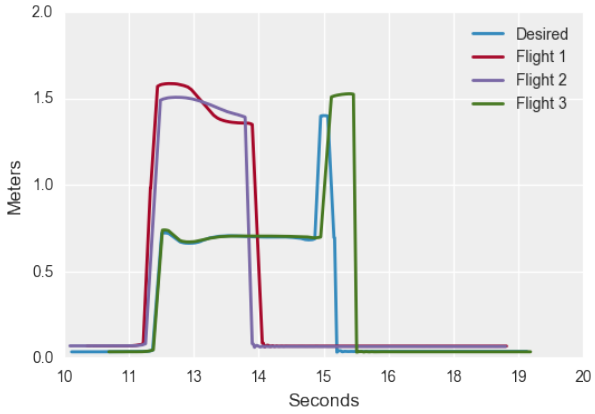
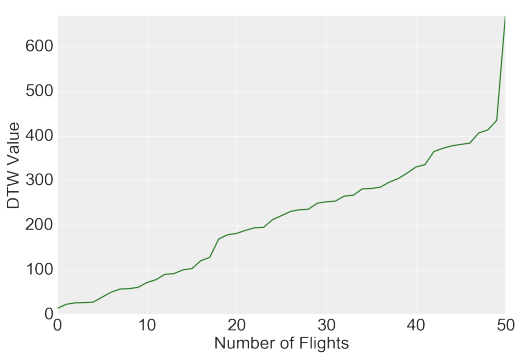
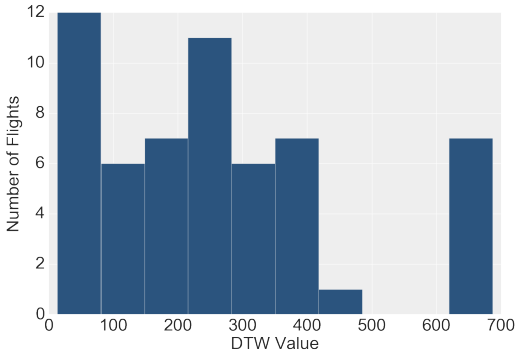


FIGURE 3.4: 2D representation of z axis position for time interval [10,20].

Figure 3.5 presents the distribution of the DTW values for all flights for only one time window $[10, 20]$. We can observe that most flights do have a DTW distance between 150 and 350. As a starting point, we chose to classify all flight with DTW value between 0 and 99 as *Good*, meaning that no degradation is identified. Flights with DTW values between 100 and 299 are assigned the label *Bad*, which could describe a possible defect on the UAV's propellers, however the mission can be continued. Finally, all flights with a DTW value higher than 300 are labelled as *Worst*. This last label warns that the propellers of the UAV is probably damaged and could lead to a crash.



(A) DTW value between first show and each other show.



(B) Distribution of experiments grouped by DTW value.

FIGURE 3.5: DTW values representation for time window of $[10, 20]$.

Classification

For estimating the performance of the model, the dataset is split into 90% training and 10% testing. The training dataset is then divided again such that 70% is used for training and the remaining 30% for validation. The training and validation datasets are used to find the best parameters and tune the performance of the model. In the end, the final model is fitted on the entire 90% training dataset, and the 10% testing dataset is used to measure the actual accuracy of the model.

There exist different metrics used for assessment of machine learning algorithms. For instance, precision, recall, F1-score and accuracy are the ones that are commonly used for this purpose. In general, precision measures how correctly the model predicted the true positives (3.3). On the other hand, recall estimates how many of true positives were actually found (3.4). Finally, F1-score is the harmonic mean of precision and recall, estimating how many false positive and false negatives were predicted (3.5). Higher F1-score an indication of a lower number of false positives and false negatives. Lastly, accuracy shows how often the model predicted correctly. It is one of the frequently used metrics for estimating the performance of the algorithms (3.6).

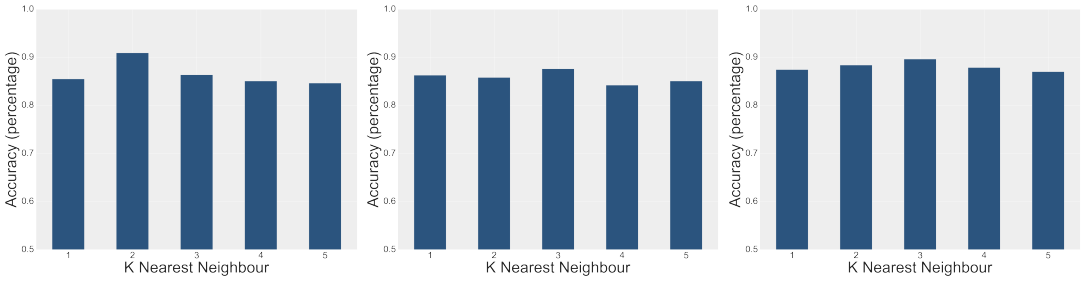
$$Precision = \frac{TP}{TP + FP}, \tag{3.3}$$

$$Recall = \frac{TP}{TP + FN}, \tag{3.4}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}, \tag{3.5}$$

$$Accuracy = \frac{TP + TN}{T + N}. \tag{3.6}$$

where TP is the true positive predictions, TN is the true negative, FP is the false positives, FN is false negatives, P is all positives and finally N is all negative predictions.



(A) (B) (C)

FIGURE 3.6: Model comparison for different k values. (A) represents x-axis, (B) represents y-axis, (C) z-axis respectively.

Before we can assess the performance of the algorithm, we need to find convenient values for k and w parameters. The strategy for finding these values is, first, to run the algorithm for different k values, such as $1, 2, 3, 4, 5$ for a given warping window w of 200, and estimate the accuracy of the model. The model achieving the highest accuracy is chosen as an convenient parameter value. Based on the results shown in figure 3.6, we can see that the model achieves its best performance for the z-axis position when k is 3 for a warping window of 200.

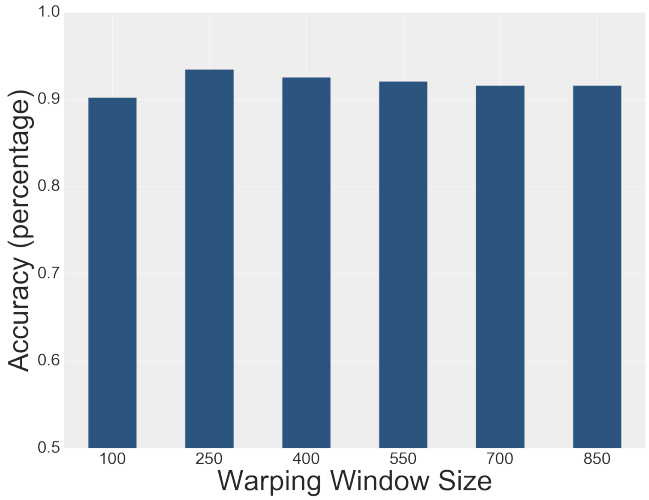


FIGURE 3.7: Model comparison for position z with different w values.

The next step is to use the best k and run the algorithm for different warping window sizes. Figure 3.7 presents the results of 6 new models that have been trained and tested for different ws of 100, 250, 400, 550, 700, 850, and $k = 3$. It is obvious that the algorithm has its best performance when the warping window size is 250.

Finally, these parameters, $k = 3$, $w = 250$, have been used and the performance of the algorithm have been assessed using f1-score and accuracy metrics.

Table 3.1 presents the final results of the classifier's performance for x, y, z positions. For each 10 seconds long sliding time window it takes about 6 seconds for the algorithm to perform its computations and return the severity of the damage.

TABLE 3.1: The final results of the performance of the algorithm.

Feature	Accuracy	F1-score	k	Max Warping Window	Comp. Time
X	92.5%	90.7%	3	250	$\approx$ 6 sec
Y	88.5%	93.1%	3	250	$\approx$ 6 sec
Z	90.6%	85.2%	3	250	$\approx$ 6 sec

All presented results have been obtained by the proof of concept [REF]. The results prove that kNN, along with DTW, can successfully be used for classifying streams of time series in online bases.

3.4 Conclusion

In this chapter, we presented an approach which predicts the level of degradation of the UAV's propellers during its mission, based on the flight coordinates. Such a lightweight method can be easily deployed in the autopilot of the robot, improving its autonomy and reducing the risk of unexpected breakdowns by predicting any occurred degradations in early stages.

The model is based on well-proven machine learning techniques, such as kNN along with the DTW, which has been used as a distance measure for computing the difference between time series data of different lengths. The main procedure of the approach consists of two phases. In the first phase, the training data is processed and

prepared such that the model can be trained using a supervised learning approach. The second phase uses the trained model in an online mode and predicts the level of degradation of a UAV's hardware-component using sliding time window technique.

Our main contributions were as follows. (1) An implementation of a lightweight approach that can successfully identify the level of damages affecting UAV's propellers. (2) step-by-step guide of the entire process, that allows the reader to apply this technique on their own missions. (3) giving an open source access to the proof of concept software of the presented approach, which can be reused or/and extended for any other personal usage [116].

Despite the good performance obtained by our model, there are many further research opportunities for extending and improving this technique. For instance, a possible extension is testing the proposed method on different scenarios targeting more than one hardware-component. Or performing a one-class classification for identifying a specific problem, for instance a motor failure identification [155] or even monitoring of helicopter gearboxes [74]. Finally, using the data collected by the internal sensors of the UAV would widen the usage of such method, making it available for outdoor missions.

4
Cyber Gym Robotics Platform

Training aerial robots in real-world settings using model-free reinforcement learning can be expensive due to the fragile nature of robots and their highly nonlinear dynamics. One way to tackle this problem is to start the training of agents in a simulated environment generating a large number of training data inexpensively, and only after the desired behaviour is achieved, transfer the model on a physical system. However, since reinforcement learning algorithms are very sensitive to the underlying model of the environment where the training of the robot has proceeded, a framework that provides close-to-real world settings is crucial. Hence, in this chapter, we present the Cyber Gym Robotics modular platform for training different robotic systems. It not only provides realistic environmental settings but also allows the control engineer to switch between different learning algorithms and /or learning agents with ease. Moreover, it offers two different mission controllers for designing intelligent control strategies of different complexity levels.

Parts of this chapter will be sibmitted for publication.

4.1 Introduction

For decades, robots have been implemented in factories to perform simple repetitive tasks since the environment is highly structured. However, endowing robots with human-like abilities for performing motor skills in a dynamic and unstructured environment remains one of the biggest challenges in robotics. Only recently, robots started to be further involved in different civilian applications. Nowadays, we often meet domestic robots that perform household tasks such as vacuum cleaning, grass cutting or even washing windows. However, for designing intelligent robots able to operate in changing environments, one should implement more generalized controllers allowing robots to adapt and even learn new behaviours without the programmer's intervention. To achieve such performance could be by allowing robots to observe their environments through onboard sensors and learn to map these observations to desired actions. Such decision-making interface provides reinforcement learning, which lets robots to learn new behaviours using very minimal human feedback, without requiring any prior knowledge about their environment.

Although over the last few years reinforcement learning methods have shown some notable successes in robotic applications, the training process often can be challenging and costly. One reason is that for acquiring the desired behaviour, current reinforcement learning algorithms require a large number of training samples, obtained through tens of thousands of trials. On the other hand, considering the fragile nature of robots, such way of training is highly expensive, sometimes even life-threatening, therefore also non-desirable.

One way to tackle this challenge of real-world interactions is to start the training in a simulated environment generating a large number of training samples inexpensively. Moreover, when the desired behaviour is achieved, transfer the learned model on a physical robot to perform the final fine tuning in real-world settings. However, it is worth noting that reinforcement learning methods are highly sensitive to the underlying model of the environment where the training of the robot has proceeded. Hence, having a framework that provides close-to-real world settings is crucial [126].

Given our goals described above, we can identify the following requirements for our software solution:

- Modular software stack enabling to swap individual components, such as the simulator or even the target robotic platform, with minimal changes.
- Ability to train and test multiple reinforcement learning models.
- Support for multi-concept learning models.
- Ability to run different tasks and missions.
- Ability to port code and models into real robots with no or minimal changes.

To satisfy our requirements, we evaluated, analysed and tested existing solutions and open source software.

One of the pillars in robotics applications is ROS, the Robot Operating System, which acts as a middleware and provides a layer of hardware abstraction to develop robotic software. Due to its flexibility and heterogeneity, ROS is used by a wide range of robots. One of its key advantage is its message-passing feature implemented as a publish-subscribe pattern, enabling efficient, scalable and flexible

inter-process communication. It allows for the same software to be run on different target platforms without any changes.

With the growing number of interests in the field of reinforcement learning and robotics, several platforms have been proposed, aiming to facilitate the training and testing processes. For instance, [217, 114] extend the OpenAI Gym, proposing a toolkit for robot training using ROS and Gazebo. Another framework called Gym-Ignition [44] has been proposed for creating reproducible reinforcement learning environments for robotic. They use the new generation of the Gazebo simulator, called Ignition Gazebo, for providing realistic robotic environments. By adding a new abstraction layer, it makes easier to integrate new physics engines in C++ and switch between them during the simulation. Even though, the existing learning platforms offer compatible features [53, 95], they either provide more task and/or robot-specific environments, or they lack the modularity required in this work. Lastly, not every proposed framework supports ROS, which is crucial, especially when the goal is to port simulated policies on real-world robots.

In this study we propose RotorS Gym framework [125, 126], which combines and further extend the following powerful open-source toolkits: OpenAI Gym [142], gym-gazebo [114], RotorS Micro Aerial Vehicle (MAV) framework [49], Gazebo simulator [208] and ROS [170]. Such combination resulted into one complete framework for benchmarking deep reinforcement learning baselines for different multirotor systems in a close-to-real-world like environment. Later we expanded the RorotS Gym framework presenting the Cyber Gym Robotic (CGR) generic cross-platform, designed as a multifunctional research-oriented tool for facilitating the creation of reproducible reinforcement learning environments for robotic applications. The Cyber Gym Robotic inherits the main functionalities of RotorS Gym, and further augments its features, simplifies the configuration and execution.

Below we describe the whole architecture of the Cyber Gym Robotic platform in great details.

4.2 Cyber Gym Robotics Framework

In this work, one of the main objectives to train aerial robots to perform different tasks by applying reinforcement learning-based control. In general, in order to enable robots to solve complex decision-making problems, they need to dynamically interact with their environment, gathering information from their sampling experiences. However, such training in real-world settings is challenging, expensive, time-consuming and often dramatic crashes are unavoidable. Thus, our motivation was to design a multifunctional and practical platform, resembling real life conditions.

The main intent of the first version of the Cyber Gym Robotics platform, called RotorS Gym [125], was to provide safe and realistic environment for training multirotor systems while experimenting with different reinforcement learning baseline algorithms. It combined a few open source toolkits, such as OpenAI Gym, ROS, Gazebo simulator, RotorS Micro Aerial Vehicle Simulator Framework, which has been used for training our agents to perform hovering and manipulation tasks in a continuous action space domain. However, RotorS Gym was limited only to simple robotic missions and suffered from integrated architecture. Hence, we extended RotorS Gym framework and designed the Cyber Gym Robotics modular platform. Because of its generic interface it allows to seamlessly switch between robotic simulators, adapt deep neural network architectures and tune different parameters and

hyper parameters with ease, create multi-concept learning scenarios and finally provides an easy to deploy configuration and execution.

We hope that such platform will be beneficial for the robotic community as well as will contribute in development of different reinforcement learning methods for robotic systems.

4.2.1 Cyber Gym Robotics Architecture

Cyber Gym Robotics platform is built on top of ROS and is composed of three main, fully modular, components: 1) mission controller, 2) communication manager and 3) robotic interface.

In general, the controller component inherits the structure of OpenAI Gym, which allows constructing a fine-grained abstraction of the environment and the learning method. The communication component provides a generic interface between the controller and the robotic platform through ROS plugin using gazebo-gym API.

Finally, the last component is a modular system that consists of either software that replicates an accurate physical model of a robot resembling real-life conditions or a real, physical robot. Figure 4.1 presents the graphical demonstration of the CGR architecture. While designing the architecture of CGR platform our main intention was to provide a modular tool where each attribute is an independent entity and can be easily adjusted and/or modified, without affecting on the functionalities of other components.

Before describing the workflow of the CGR platform, we below give the thorough insights of each component.

The mission controller

It is the core of the entire Cyber Gym Robotics platform. It is responsible to supervise missions by running algorithms and handling communications messages.

This component is modular by design as it contains several modules:

- **Mission definition:** Define all hyperparameters of the mission or the neural networks, modules to use, mission success and failure conditions as well as any additional necessary configuration.
- **Concept:** In this implementation, the concept is comprised of a reinforcement learning algorithm and a mission-specific handler. Both components are decoupled and, thus, allow to easily use different algorithms. The mission-specific handler has to implement three basic methods that should work as follows:
  - **init()** - initializes the robot, the simulated environment if necessary and setup ROS-specific communication channels.
  - **step()** - is used to execute the actions at each step. After an action is executed, the $step()$ must function returns three essential values: **observation**, which is an environment-specific object, containing different information about the agent, for instance, its position and/or orientation, or the rotors velocities. **_reward** is a floating-point number that the agent gets after each performed action. **_done** is a boolean value that indicates whether the robot ended its mission regardless of success or failure. *Done*

is *True* when the maximum number of steps have been accomplished or if the agent overpasses the predefined constraints.

  - **reset()** - this function is called when an episode starts in order to reset the environment and setup the initial environmental conditions.

- **Multi-concept learning handler:** As a mission can be organized into sub-missions, also referred to as concepts, this module is responsible for handling the switch between the concepts. Implemented with flexibility in mind, this handler is fully configurable and easily extendable to support any number of concepts as well as any type of concept activation conditions. We call this module Concept-based controller and give detailed introduction on section 4.2.2.

- **Communication interface:** Making complete use of ROS, this module facilitates the usage of ROS messages and topics. It provides helper methods that can be reused by any concept.

Based on the explanations above, the mission controller can be defined as the orchestrator between each module in use.

Communication Manager

In this project, the communication is being handled by ROS, which provides a publish-subscribe messaging system. As mentioned before, the usage of ROS is key to make this platform modular across simulations and real robots.

Robot interface

Robot interface is the last component of the platform which can either be a real robot or a simulated replica. There are a few requirements that the robot interface should fulfill. It should fully support ROS and provide an adequate sensor readings as well as publishing interfaces to send commands to the appropriate hardware modules, such as motors.

However, as it was mentioned before, reinforcement learning methods are highly sensitive to the underlying model of the environment where the training of the agent is being operated. Thus, if the training is performed in a robotics simulation, then ideally, it should be able to provide close-to-real physical world settings. In other words, it is crucial that simulated robot is a close replica in terms of physical properties (size, weight, aerodynamics, ...) and hardware capabilities (motors, propellers, joints, ...) In this work, our choice of environment where the most of our agents have been trained is Gazebo simulator, which is an open-source 3D dynamic multi-robotic simulator, with a high-quality graphical interface, capable of recreating complex real-world environments with high physical fidelity. Gazebo provides a headless mode (no graphical interface), which helps to scale up the computations and significantly accelerates the training process.

Although Gazebo has all the potential to be used as a fundamental tool for training reinforcement learning agents, one of our main goal was to allow the end-user to switch between different physics engine with ease. We have additionally performed experiments in two different environments, such as MatLab [128] and Panda, a python-based 3D simulator [183]. In both cases, only the ROS node had to be adjusted.

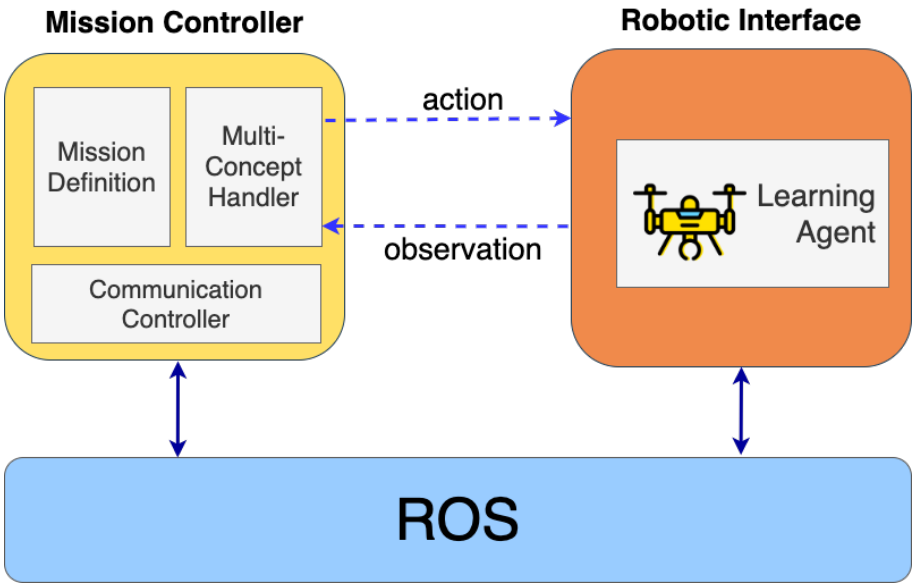


FIGURE 4.1: The architecture of the Cyber Gym Robotics framework.

4.2.2 Concept-based Control System

In general, acquiring robust and agile control policies for robotic tasks using reinforcement learning is not trivial. The challenges arise when we deal with more sophisticated robotic systems with larger number of degrees of freedom operating in high dimensional, continuous state-action spaces. Moreover, writing a reward function that covers all aspects of the learning problem while guiding the robot to obtain an optimal policy is often impracticable, since the agent needs to discover the actions that lead to higher rewards in a large pool of low-reward sequences. And even if the desired behaviour is obtained, it can not be integrated into other missions, and every new problem is being newly retrained. Hence, to achieve a successful accomplishment of the missions as well as reusing already obtained behaviours, we propose a concept-based mission controller that allows to decompose the whole learning task into independent subtasks, allowing the agent to learn one subtask at a time. In the end, when each subtask is learned, the mission controller is used to orchestrate the action to be taken, an example of mission controller workflow is shown on figure 4.2.

Such an approach offers a number of benefits. First, by decomposing the entire problem into independent subtasks, the size of the state-action space for each task reduces dramatically. This has the direct effects of reducing the complexity of the reward function, allowing faster training toward mastering the sub-goal and even enabling parallelization of the training process. Another advantage is the ability to create a database of diverse skills that can be assembled in order to solve a complex behaviour. It is important to note that since each concept is an independent component, it implements and runs its own logic, thus enabling the usage of multiple solutions, such as RL algorithms and classic control algorithms, as part of larger policy ensemble.

The great potential of such an approach, has motivated efforts to adopt different techniques tackling similar ideas by integrating primitive functions into new complex functions and solving more complex learning problems. For instance, Qureshi et al. [163] proposed a policy ensemble composition method that takes basic tasks and transfers them into a new complex problem, using standard or hierarchical reinforcement learning. Gudimella et al. [60] introduced concept networks for

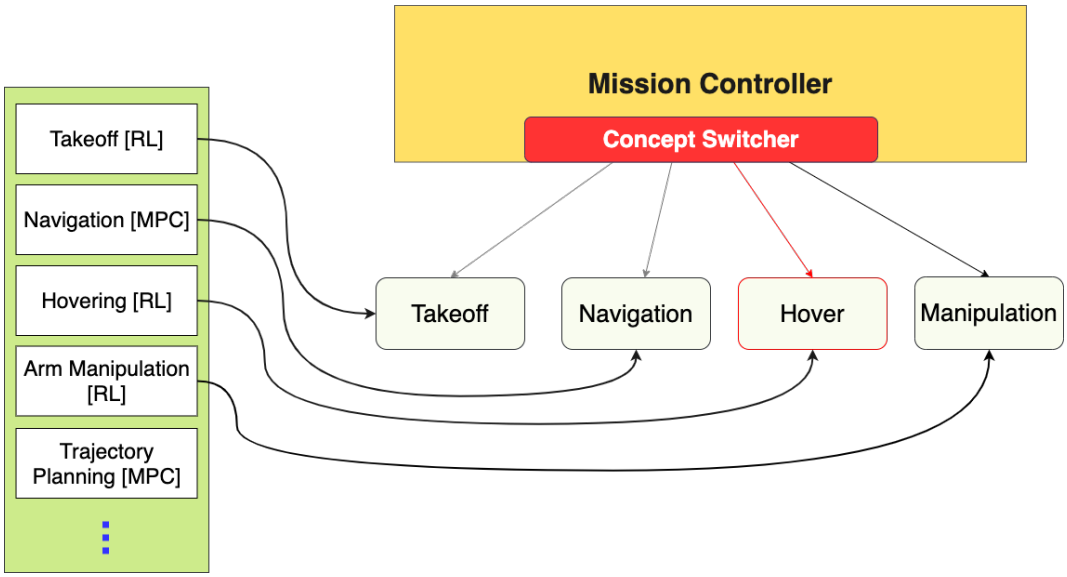


FIGURE 4.2: The workflow of the Mission Controller using a concept-based control.

training a dexterous manipulation, where the authors use reinforcement learning to train "Concept Networks" which are tree-like structures in which leaves are "sub-concepts", representing policies on a subset of state spaces. Some other works have made progress towards delivering on this promise by suggesting techniques based on a modular representation of the learning tasks [195, 161, 61, 160, 163], which could find optimal policies in more than three-times fewer environment interactions, than non-modular based policies [137]. The notion of "Concept-based" mission controller has been inspired by these works mentioned above.

Finally, incorporating the concept-based mission controller into Cyber Gym Robotics platform advances its performance in regard to other state-of-the-art platforms.

4.3 Software Specifications

The Cyber Gym Robotics framework has been implemented in Python by making extensive use of PyTorch, a state-of-the-art and open source machine learning framework, as well as ROS related Python libraries in order to communicate with the robots. The software is used as a Command Line Interface (CLI), allowing to easily be run on desktop, servers or robotic platforms, supporting the basic dependencies (Python).

Cyber Gym Robotics features 2 modes: training and demonstration. The training mode, as the name suggests, is used to train a model in a simulated environment. For each training episode, the model, its performance and graphical representation are saved. By saving each model, the CLI supports resuming training given a specific trained model. On one hand, this allows to select a better candidate model to continue the training. On the other hand, every trained model can be tested in demonstration mode. It is worth to keep in mind that the most recent model is not always the one with the best policy, thus it is important to save all previous models.

The demonstration mode is used to run a specific model in a continuous environment, be it simulated or real. In this mode, the model is used as is and the agent is not being trained anymore. In other words, the model is not being updated.

4.4 Practical Implementation

In this section, we aim to demonstrate the ability of the Cyber Gym Robotics platform to seamlessly switch between simulated and real agents and show that learned behaviours can be ported onto a real robotic platform with only minimal integration and prove that CGR indeed provides close to real-life training possibilities. In other words, the experiments focuses on CGR's software implementation, modularity and stability across different environments and even hardware.

4.4.1 Experimental Setup

To achieve the experiments, we took the fully trained and learned model explained in chapter 5 and implemented the required ROS-based communication interfaces on a real hexacopter platform, which has been custom rebuild as explained below. Concerning the software integration, only three communication interfaces were required to be adapted: a publishing endpoint for engaging motors, another publishing endpoint for sending motor velocities and finally, a topic to subscribe to for receiving the agent's onboard sensor data.

In order to easily reproduce real experiments as safe as possible and focus the tests on the integration between Cyber Gym Robotics software and another hardware agent, a mock-up sensor controller has been implemented on the UAV. This allows to run targeted tests against the hardware and make sure that the learned model results similar behavior as during simulation.

We ran two experiments focusing, first, on the stable hovering, and second, on navigation. During the exeriments, the mock-up sensor controller is configured to lerp between a start and end for the position and orientation.

4.4.2 Experimental Results

During the first scenario, we measure the agent's usage of the learned model and how the motor velocities change over the course of the experiment.

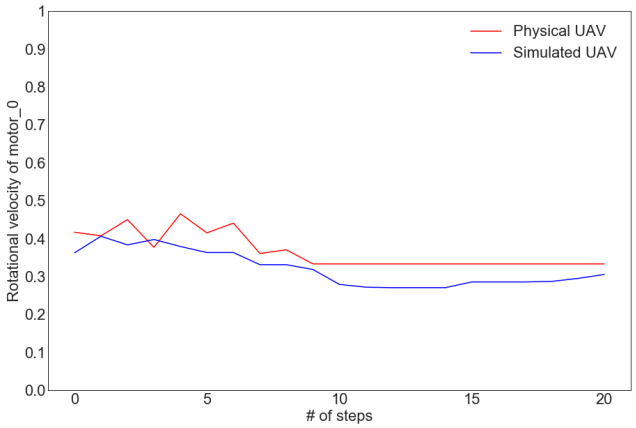
Figures 4.3 and 4.4 compare each real motor velocity changes against each motor velocity changes out of the simulation. The results clearly show that, as expected, the learned model could be ported onto a physical UAV and present similar behaviours. As presented in the simulated training videos in chapter 5, the agent requires a few seconds before fully stabilizing, which is also reflected in the physical velocities.

Next, we run and analyze the second scenario, where the agent should navigate from an initial pose of $x = 0, y = 0, z = 0$ to a goal pose $x = 0, y = 0, z = 1$. As for the first scenario, we compare each real motor velocity changes against each motor velocity changes out of the simulation, respectively, as deminstrated in figures 4.5 and 4.6. Even with this more complex scenario, we can identify clear similarities between the learned model run in simulation and on a real robotic platform. Initially, the agent needs constant high motor velocities to take off and gradually reduces the velocities in order to go into a hovering state at the given goal position.

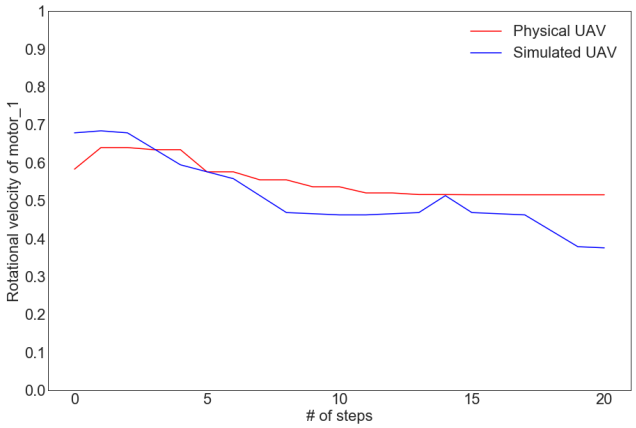
Video cpips demonstrating the performed experiment are available [118] and [119].

4.4.3 Reconstruction of AscTec Firefly

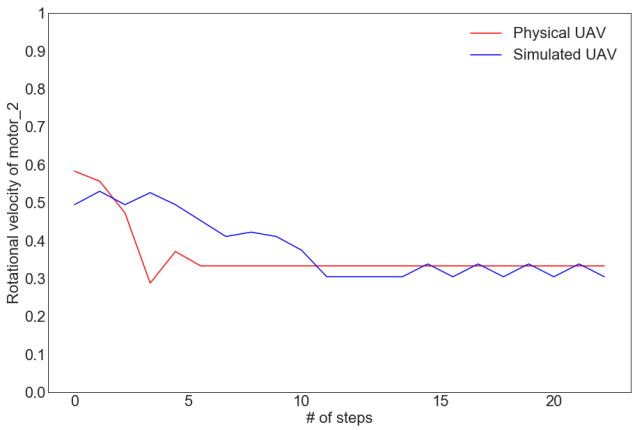
The decision behind the usage of the Asctec Firefly has been driven by several mainly factors. First and foremost, Asctec Firefly has been made with research in



(A)

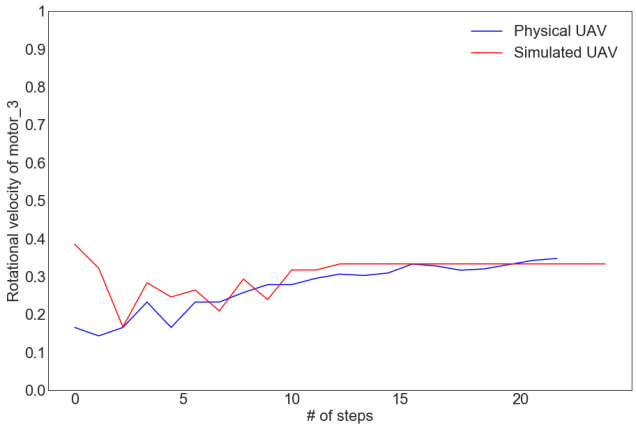


(B)

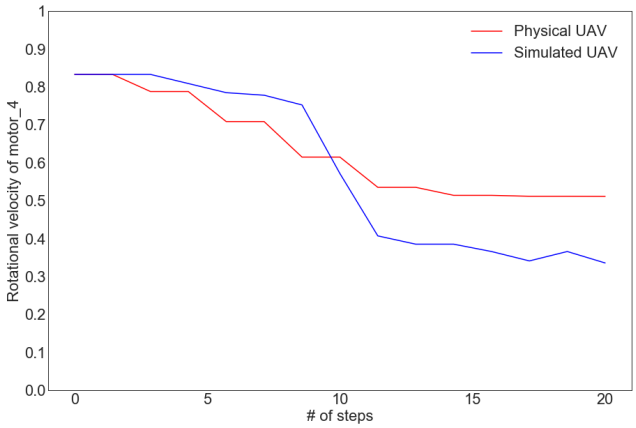


(C)

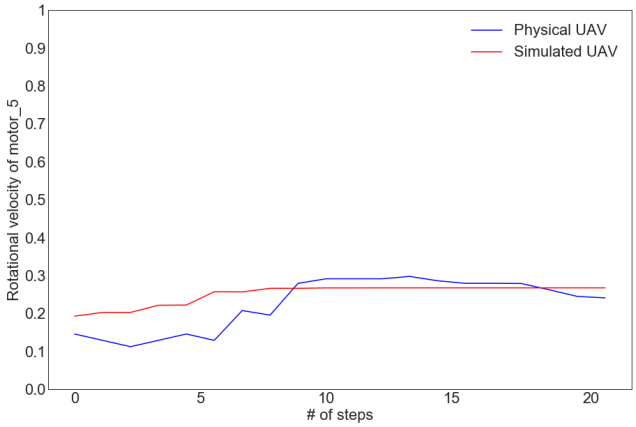
FIGURE 4.3: The comparison of rotational velocities of the simulated and physical hexacopter for each motor during 20 environmental interaction steps.(For a smoother curve the moveing average is used.)



(A)

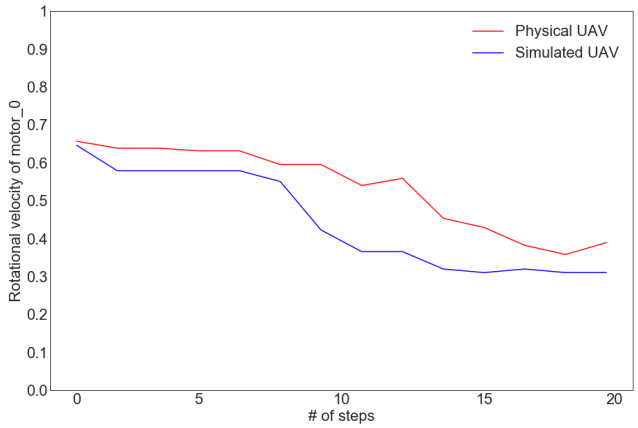


(B)

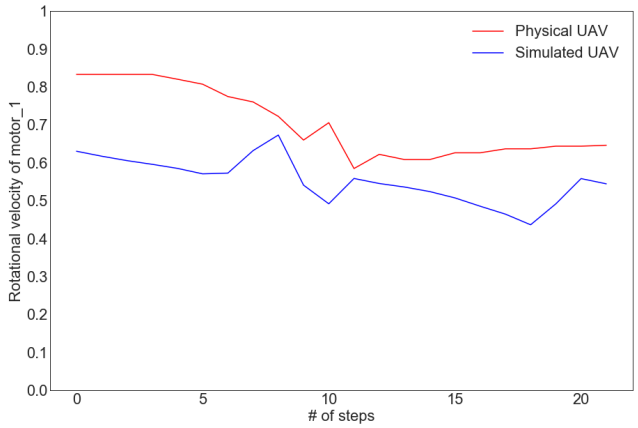


(C)

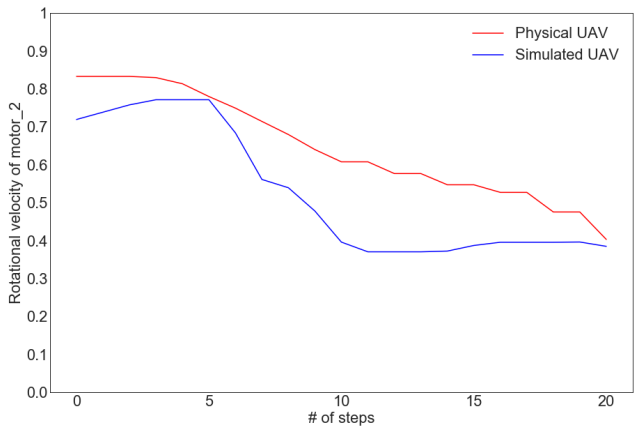
FIGURE 4.4: The comparison of rotational velocities of the simulated and physical hexacopter for each motor during 20 environmental interaction steps.(For a smoother curve the moveing average is used.)



(A)

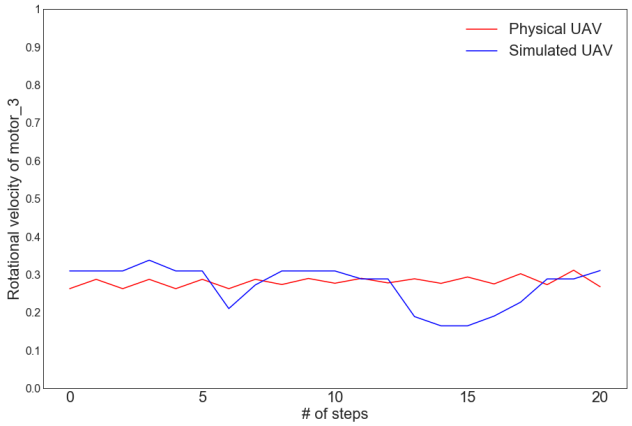


(B)

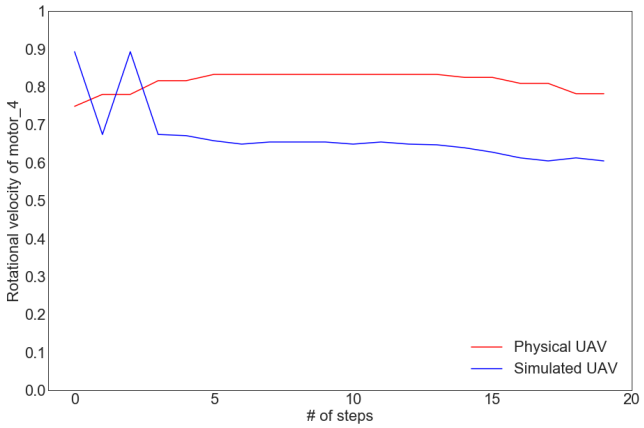


(C)

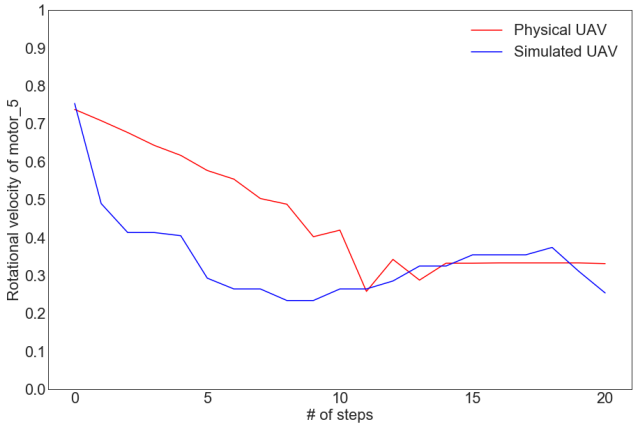
FIGURE 4.5: The comparison of rotational velocities of the simulated and physical hexacopter for each motor during 20 environmental interaction steps.(For a smoother curve the moveing average is used.)



(A)



(B)



(C)

FIGURE 4.6: The comparison of rotational velocities of the simulated and physical hexacopter for each motor during 20 environmental interaction steps.(For a smoother curve the moveing average is used.)

Technical Data – AscTec Firefly	
UAV Type	**Hexacopter**
Onboard computer	Up to 3rd Generation Intel® Core™ i7 Processor
Size	605 x 665 x 165 mm
MTOW	1,6 kg
Max. payload	600 g
Flight time incl. payload	12–14 min.
Range	4,500 m ASL, 1,000 m AGL
Max. airspeed	15 m/s
Max. climb rate	8 m/s
Max. thrust	36 N
Wireless communication	2,4 GHz XBee link, 10–63 mW, WiFi (optional)
Intertial guidance system	AscTec AutoPilot with 1,000 Hz update rate
Flight modes	GPS Mode, Height Mode, Manual Mode
Emergency modes	Direct landing, Comehome straight, Comehome high

mind and thus, it consists of a modular, lightweight and compact frame for a hexacopter. Second, the accurate 3*D* model of the UAV is available and supported in one of the state-of-the-art simulations.

(A) (B)

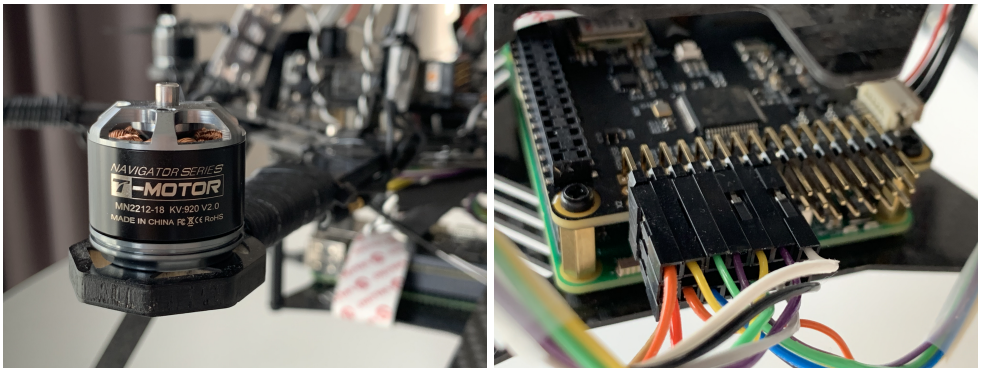
FIGURE 4.7: (A) A carbon fiber propeller. (B) An electronic speed controller.

In January 2016, Ascending Technologies has been acquired by Intel. Several months later AscTec official declared their AscTec Research Line UAVs to reach end of life by 31st March 2018. This means that no further support, maintenance or updates would be provided. Considering the immense success of the AscTec Research UAVs, this news has not only been surprise, but put us in a position to reconsider the usage of AscTec Firefly in this work.

After careful consideration and given the advantages listed above, the main challenge has been related to the usage of the current AscTec hardware, which is mostly proprietary or slightly derived from existing products.

Therefore, we took the decision to continue using the UAV's base but rebuild the hardware as well as the software stack. This allows us to gain full control over every component and create an improved platform for research and experiments.

With the aim of training an algorithm to fly a UAV from scratch, the algorithm must have direct access to low-level controls over the hardware. More specifically,

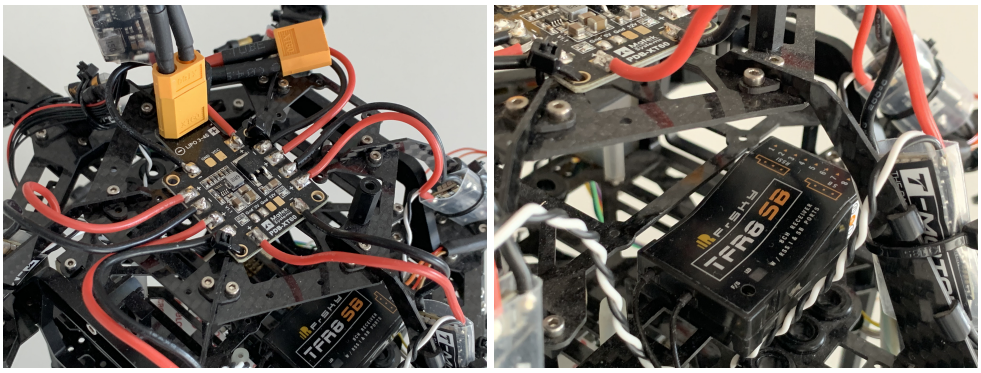


(A) (B)

FIGURE 4.8: (A) A brushless motor. (B) The motors connection to Navio2.

the controller must be able to access the rotational velocities of each individual motor.

The first step toward rebuilding the AscTec FireFly has been to strip and disconnect each component and test them individually in order to eventually reuse them. However, as most hardware components are proprietary or derivations, reusing any previous part has proved challenging.



(A) (B)

FIGURE 4.9: (A) The radio reciver. (B) The power distribution board.

With only the UAV's frame remaining, we bought the necessary hardware components, mainly:

- 1 **Onboard Computer: Raspberry Pi** 3 **Model B extended with Navio**2

  The most important component is the onboard computer as it's the brain and controller of the UAV. We chose a lightweight solution consisting of a Raspberry Pi 3 equipped with Navio2, which is a HAT (Hardware Attached on Top) addon. Navio2 is an Autopilot HAT providing the necessary hardware extensions to build a fully functional autonomous UAV.

- **Navio2** includes:

  - Dual IMO with accelerometers, gyroscopes and magnetometers for orientation and motion sensing

  - High resolution barometer (resolution of 10cm)
  - RC I/O processor accepts PPM/SBUS input and provides 14 PWM output channels for motors and servos. This allows to directly control individual motors via the ESCs.
  - Extension ports exposing ADC, I2C and UART interfaces for sensors and radios
  - Full support for ROS, ArduPilot and MAVROS, allowing to easily create any kind of UAV application.

- 6 **Electronic Speed Controllers: BeliHeli** 32

  The Electronic Speed Controller, or ESC for short, are the link between the motors and the onboard controllers. As motors need a lot of power, ESC's are directly connected to the power distribution board in order to get the right amount of voltage. The onboard computer, in this case Navio2, connects to the ESC via its command cable.

  The ESC we chose, support $3-6S$ batteries and can provide $35A$ continuously, which is more than enough for our purpose and will allow a wider range of future missions. The ESC's performance is chosen in a way that we avoid reaching the electronic limits when the motors are at full throttle.

- 6 **Brushless Motors: T-Motor Navigator** $MN2212\ V2\ 920KV$ **Brushless Motor** The motors has been chosen by taking into account several factors such as the compatibility with the ESC's, the mount specifications and the overall performance. For the latter, it was important that the motors were highly performant and capable of lifting at least the initial AscTec weight, taking into account a possible payload.

- 6 **Propellers: T-MOTOR** $8x2.7\ V2$ **Carbon Fiber Prop Propeller CW/CCW**

  Based on the specifications of the brushless motors and UAV's frame architecture, appropriately sized propellers have been installed. Originating from the same brand as the motors, they offer optimal compatibility. The propellers are made out of carbon fibres, are highly efficient and thus can provide stable flight behaviour and contribute to an improved battery life.

- 1 **Radio Receiver: FrSky** $TFR8\ SB\ 8ch\ 2.4Ghz$ **S.BUS Receiver FASST Compatible** This UAV aiming to be autonomous, we could argue that a radio receiver is optional. However, it is crucial to test the UAV manually and make sure it is properly set up, configured and balanced and is able to fly with manual controls. The radio receiver is connected directly into Navio2 and allows the UAV to be remote controlled when running the auto pilot module.

- 1 **Power Distribution Board: Matek PDB**$-XT60\ w/BEC$ **(**$5V$ **and** $12V$**)**

  Having 6 motors and 1 onboard computer, the build requires an appropriate power distribution board, which, as the name hints, is a simple electronic component responsible for distributing the power from the battery toward each motor and as well as the onboard computer. It is important to note that motors require around $12V$ and can draw up to $13A$ continuously.

- 1 **Battery: Hyperion** $G5\ 50C\ 3S\ 4000mAh$ **LiPo Battery**

  Finally, the battery is bought last based on the required cells of the other components. In this case, a 3 cell $(3S)$ Lithium Ion battery fits. We chose a $4000mAh$ one as this provides a good compromise between weight, size and autonomy.

Once the UAV reconstructed, the software stack could be set up. As a base, the default operating system image provided by emlid [37], the company behind Navio2, has been put on the onboard computer. The image came with Ardupilot and, most importantly for us, ROS. Navio2 also provides a repository for Python and C++ that allows developer to access the sensors and pins directly, thus enabling us to control motors by direct communication to the ESC's [38].

4.5 Conclusion

Training reinforcement learning agents in real world settings can be very challenging. The problem becomes more complex when dealing with unmanned aerial vehicles because of their highly non-linear dynamics. Thus, for training our agents we designed the Cyber Gym Robotics a modular platform providing generic interfaces that can be customized as needed. Since it uses familiar tools, does not put constraints on simulated environment, it allows the user to not only modify the existing robotic models, but also create the new ones, at the same time experiment with different reinforcement learning baselines.

To validate the presented approach, we ran several experiments on the rebuild UAV and demonstrated that CGR indeed provides an usable structure to train in a simulation and test in the physical world with minor adjustments.

5

Motor skills learning of UAVs with deep reinforcement learning

In the previous chapters we have shown the great potential and success of machine learning methods for solving different problems in the field of robotics. We have presented a lightweight approach that can directly improve the safety of a UAV using only its flight data. In the next chapter we aim to present how machine learning can be advantageous for solving control problems without having any prior knowledge about the dynamics model of the robot in a continuous action-state setting.

Parts of this chapter have been published in:

- Manukyan, Anush, et al. "Deep Reinforcement Learning-based Continuous Control for Multicopter Systems." 2019 6th International Conference on Control, Decision and Information Technologies (CoDIT). IEEE, 2019.

5.1 Introduction

Unmanned aerial vehicles (UAVs) are robotic aircrafts that are being increasingly deployed in many civilian applications [143, 133, 42, 216, 113], owing to their high flexibility, possibility to carry a wide range of sensors, inexpensive cost and hovering abilities. However, providing autonomous and stable navigation in unknown and unstructured environments remain a challenging problem, due to the highly nonlinear dynamics of small aircrafts.

Over the past few decades, classical control approaches have been dominantly used in many aerial robotic applications because of their remarkable performance in stable environments. For instance, linear quadratic Gaussian [186], model predictive control (MPC) [184], backstepping [10], gain scheduling [66], and Proportional-Integral-Derivative (PID) [173] control techniques are some of the most popular methods used in autopilots systems of unmanned aerial vehicles. However, such control architectures heavily rely on the accurate mathematical model of the aircraft as well as require significant amounts of hand-tuning to reach the desired behaviour, limiting their usage for more complex scenarios. Thus, designing high-performance flight control systems for autonomous aircrafts that operate in unstructured and unpredictable environments require more robust and adaptive control approaches.

Lately, machine learning-based techniques, known as reinforcement learning, have become an active area of research addressing the limitations of classic control methods. Reinforcement learning is a powerful framework that allows autonomous agents to learn hard-to-engineer behaviours through interaction with their environment. Thus, it enables us to develop optimal control policies without making any assumption about the aircraft's dynamics or requiring prior knowledge about its environment. Although numerous reinforcement learning algorithms have made significant breakthroughs in various challenging decision-making problems, until recently, the majority of the proposed solutions were limited to discrete state and action-spaces, when most of the real-world robotic applications require a continuous state and/or action space setup. However, recent advances in this field have unveiled a new set of possibilities to solve complex robotics problems by means of deep reinforcement learning strategies. The deep neural networks have been introduced in reinforcement learning as a function approximation. Such combination resulted in powerful tools, increasing the usage of reinforcement learning methods in different continuous action-state scenarios. Another advantage of deep reinforcement learning is the capability of simplifying the otherwise time-consuming process of designing a robot's desired behaviour by defining **what** actions the robot should perform without ever programming **how** it should function.

Generally, in reinforcement learning, the engineer specifies the goal or desired behaviour of the robot through a reward function, which acts as positive reinforcement or negative punishment depending on the performance of the robot with respect to the goal.

Recent works have shown that reinforcement learning-based methods achieve notable performances in different UAV control problems. For instance, Abbeel et al. [1] and Kim et al. [88] trained an autonomous helicopter to perform a hovering in place task and some various aerobatic manoeuvres. Other pioneering researches addressing quadcopter stabilization and control problem have been presented by Bou-Ammar et al. [17] and Waslander et al. [205]. In their work Bou-Ammar et al. proposed a nonlinear autopilot for quadrotor UAVs based on feedback linearization for precise and fast stabilization. In contrast, Waslander et al. used model-based reinforcement learning for altitude control. More recent works presented by Hwangbo

et al. [70], Pham et al. [157], Sugimoto et al. [191] propose different solutions for quadcopter altitude control.

As it was stated before, in this thesis, our main objective is to develop a model-free on-policy control system for more sophisticated aircrafts, such as flying manipulations. However, before dealing with such highly nonlinear robotic systems, we first investigate the nuances and potentials of reinforcement learning methods applied on different control problems of a hexacopter UAV, by directly mapping observations to low-level motor commands.

In contrast to most state-of-the-art works, where the mathematical model of the aircraft and the environment have been used, in this work, we use a simulated robotic system, that replicates an accurate physical model of a robot resembling real-life conditions. This increases the realism, at the same time reduces the complications that often occur while executing the trained model on the real robot, also known as a reality gap [172]. Thus, for this purpose, we have developed RotorS Gym framework, that provides a safe and close to real-world like environment for training multirotor systems [125]. Furthermore, RotorS Gym framework has been extended to a more generic modular platform called Cyber Gym Robotics which allows seamless switching between robotic simulators, adapt deep neural network architectures, tune hyperparameters with ease, and more. A detailed description of RotorS Gym and Cyber Gym Robotics platforms is given in chapter 4.

For training the learning agent, the trust region policy optimization algorithm (TRPO) has been used [176]. TRPO is a model-free on-policy algorithm that allows the agent to learn a policy without having first to generate a model. It employs the actor-critic architecture using two different neural networks: one for policy function approximation and another for value function approximation. To prevent extensive updates between previous and current policies and ensure monotonic improvements, TRPO uses a trust region technique which bounds the optimization step of the policy update.

Before presenting the performance of the learning agent, we first provide details about the flight dynamics of the hexacopter, along with the fundamentals of the learning algorithm.

5.2 Hexacopter Flight Dynamics

In this work, as a learning agent the digital replica of an AscTec Firefly UAV has been used as a learning agent, shown on figure 5.1. Later, the experiments have been carried on a physical AscTec Firefly, which, however, has been fully reconstructed, due to its obsolete hardware and software components. More details on the physical AscTec model are given later in the chapter.

Before exploring the leanring-based controller proposed in this thesis, we first provide a short theoretical background of the flight dynamics of an AscTec Firefly UAV.

5.2.1 Kinematics and Dynamics

AscTec Firefly is a hexacopter UAV with 6-DoF, consisting of a rigid body and six propellers orthogonally aligned along the body frame as shown on figure 5.2:

In order to describe the hexacopter, two coordinate frames are considered 1) the world-fixed inertial frame, F_I , and 2) the body fixed frame, F_B, which is attached to the hexacopter. The origin of the body frame is in the center of mass of the

FIGURE 5.1: a) AscTec Firefly is a hexacopter UAV produced by ASCENDING Technologies [98, 71]. b) A digital replica of the AscTec Firefly in the Gazebo simulator used as a learning agent for both training and testing the reinforcement learning tasks [50].

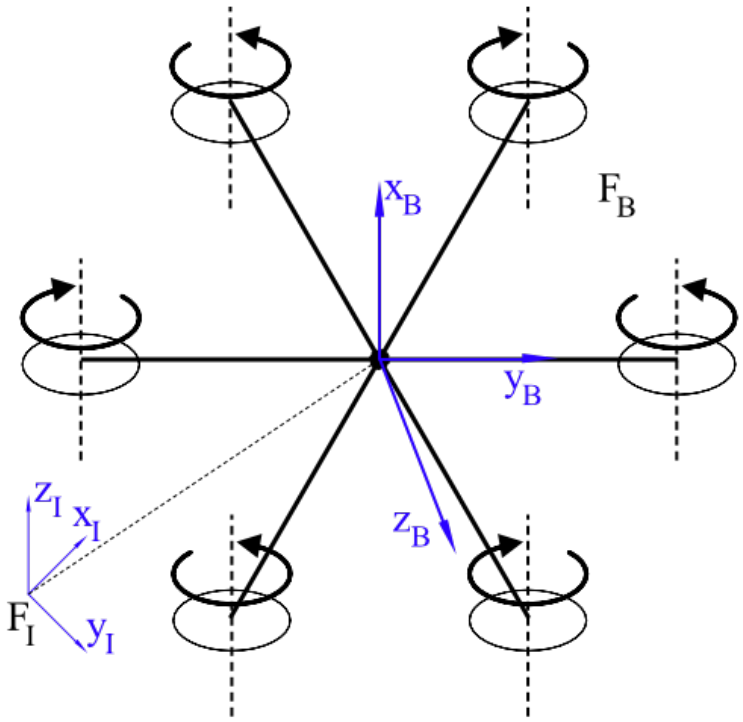


FIGURE 5.2: The illustration of the schematic structure of the hexacopter along with the rotational directions of the rotors.

hexacopter. The orientation of the hexacopter is described by ZYX Euler angles, known as ϕ roll, θ pitch and ψ yaw. These three angles together form the vector $\eta = [\phi, \theta, \psi]^T$, where ϕ and $\theta \in [-\pi/2, \pi/2]$ and $\psi \in [-\pi, \pi]$. The position of the center of mass of the hexacopter, expressed in the inertial frame F_I is represented by $\xi = [x, y, z]^T$. By manipulating the velocities of six rotors the aircraft generates three movements. When the balance of $(1, 2, 3)$ or $(6, 5, 4)$ rotors is changed then a rotation around x axis is obtained changing the ϕ angle. When the balance of the speed of 1 and 6 or 3 and 4 rotors is changed then a lateral acceleration or rotation around y axis is obtained changing the θ angle. Lastly, by the speed of $(1, 3, 5)$ or $(2, 4, 6)$ rotors is simultaneous changed then it results in a longitudinal acceleration or a rotation around the z axis, changing the ψ angle [134].

Finally, the transformation related to the position and the angular velocity from the body with respect to inertial frame is presented by rotation matrix R:

$$R = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\theta \\ s\psi c\theta & s\psi s\theta s\phi - c\psi c\phi & s\psi s\theta c\phi - c\psi s\theta \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \tag{5.1}$$

where $c\psi = \cos\psi$, $c\theta = \cos\theta$, $c\phi = \cos\phi$, $s\psi = \sin\psi$, $s\theta = \sin\theta$, $s\phi = \sin\phi$, and ϕ, θ, ψ are the roll, pitch and yaw angles respectively.

A translation from body to inertial frame is described by the linear transformation matrix T:

$$T = \begin{bmatrix} 1 & 0 & -sin\theta \\ 0 & cos\phi & sin\phi cos\theta \\ 0 & -sin\phi & cos\phi cos\theta \end{bmatrix} \tag{5.2}$$

In order to derive the dynamic model of the hexacopter the Newton-Euler formalisation is used [67]. First, the linear and angular position in inertial frame is defined as vector $[x\ y\ z\ \phi\ \theta\ \psi]^T$ and the linear and angular velocities in the body frame are defined as vector $[u\ v\ w\ p\ q\ r]^T$, where u, v and w are forward, sideways and downward motions of the aircraft. p, q and r are roll, pitch and yaw speeds, respectively.

Finally, using the rotation matrix represented in equation (5.1), the kinematics equations of the aircraft can be written as follows:

$$v = Rv_b\,, \tag{5.3}$$

$$\omega = T\omega_b\,, \tag{5.4}$$

where b is the speeds in body frame. $v = [\dot{x}\ \dot{y}\ \dot{z}]^T, \omega = [\dot{\phi}\ \dot{\theta}\ \dot{\psi}]^T, v_b = [u\ v\ w]^T$ and $\omega_b = [p\ q\ r]^T$.

The full equations of motion from Newton's law, and Euler's equation can be written as follows:

$$F_b = m(\omega_b \times v_b \times \dot{v_b})\,, \tag{5.5}$$

$$M_b = m(I\dot{\omega_b} \times \omega_b \times I\omega_b)\,, \tag{5.6}$$

where, $F_b = [F_x F_y F_z]^T$ represents all forces on the rigid body of the hexacopter, m is the mass of the hexacopter, $M_b = [M_x M_y M_z]^T$ are all the moments acting on the quadcopter and where I is the diagonal inertia matrix:

$$I = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z\,. \end{bmatrix} \tag{5.7}$$

These equations combined (F_b and M_b equations) form the dynamic equations of the quadcopter.

More detailed description of the kinematics and dynamics of a hexacopter can be found in [134, 67].

5.2.2 Control of Hexacopter

Very often autopilot systems of UAVs are based on two loops: an "inner loop", that contains four control laws, such as roll command (ϕ), pitch command (θ), yaw control (ψ) and controlling altitude z and an "outer loop", that includes two control laws positions (x, y) [134]. In general, the inner loop is responsible for the stabilization and control, when the outer loop provides mission level objectives. One of the most commonly used flight control strategy is the classic Proportional, Integral Derivative (PID) linear feedback controller, due to its robustness and simple design. The main objective in the PID controller is to adjust its parameter gains to arrive at an acceptable degree of tracking performance for each Euler angle.

Mathematically it can be represented as follows:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}, \tag{5.8}$$

where $u(t)$ is the control signal and K_p, K_i, K_d are configurable constant gains [134]. Although its predominant use in UAV autopilot systems, PID controllers face several major challenges. The highly non-linear mathematical model of the hexacopter

limits the performance of the aircraft in different complex missions. In contrast, reinforcement learning-based methods solve these problems by allowing the agent to discover an optimal control policy without having any prior knowledge about the mathematical model of its dynamics. In the next section we present in great detail the control algorithm used for training the learning agent for performing hovering and flight navigation tasks.

5.3 Deep Reinforcement Learning-based Control

The control approach presented in this work is based on reinforcement learning methods, where an agent (or a controller) discovers an optimal policy through interactions with its environment. The main process consists of an agent taking actions at an arbitrary state, receives feedback from its environment in the form of a reward indicating the quality of the taken action. The function that maps states to actions is called policy. The main objective of the agent is to discover an optimal policy that maximizes the cumulative reward. By following a given policy and receiving rewards, the agent can estimate the future expected rewards using a value function. Value functions are functions that provide an estimate of how good it is for an agent to be in a given state, or perform the given action in a given state using expected rewards. Over time, a tremendous amount of algorithms have been introduced in the literature. We provide an in-depth overview of the fundamentals of reinforcement learning along with its baseline algorithms in chapter 2.

Generally, in reinforcement learning, the choice of the learning algorithm strongly depends on the problem to solve. In this work, the main objective of the agent is to learn complex motor skills in high dimensions. Even though reinforcement learning provides a general framework for training robots to perform different actions, most methods do not scale beyond discretized environments. In contrast, policy gradient-based actor-critic methods are notable exceptions to this statement [153, 154]. Due to their effectiveness and provably convergent property, over the past decade, they have been used in a variety of robotic applications ranging from simple control tasks to complex motor skills learning [199]. Thus, based on these remarkable features, Trust Region Policy Optimization has been used as a learning algorithm. TRPO uses not only the best properties of policy gradient actor-critic based methods, but also adopts the "trust region" technique ensuring monotonic improvements in the policy performance.

5.3.1 Trust Region Policy Optimization

Trust Region Policy Optimization is a policy gradient-based method that aims to monotonically improve the performance of the agent and find an optimal policy for a given task. To describe the TRPO algorithm we use an infinite-horizon MDP defined by the tuple $(S, A, P, r, \rho_0, \gamma)$. As stated before, S is the state space, A is the action space, $P : S \times A \times S \to R$ is the transition probability distribution, $r : S \to R$ is the reward function, $\rho_0 : S \to R$ is the distribution of the initial state s_0, and $\gamma \in (0, 1)$ is the discount factor [176].

Let π denote a stochastic policy $\pi : S \times A \to [0, 1]$, and let $\eta(\pi)$ denote its expected discounted reward:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) \right], \tag{5.9}$$

where $s_0 \sim \rho_0(s_0), a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t)$. As in the most policy gradient methods, here as well, the main objective is to find the optimal policy by maximising the expected return $\eta(\pi)$. The standard definitions of the value function V_π, state-action value function Q_π and the advantage function A_π is as follows:

$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, \ldots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}) \right], \tag{5.10}$$

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}) \right], \tag{5.11}$$

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s), \tag{5.12}$$

where $a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t)$ for $t \geq 0$.

As was mentioned before, TRPO adopts the architecture of the actor-critic method, but it modifies the way the policy parameters of the actor are being updated. The new policy will be denoted as $\tilde{\pi}$ and $\eta(\tilde{\pi})$ will be the expected return of the new policy $\tilde{\pi}$. One can show that [176]:

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0, a_0, \cdots \sim \tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t) \right] \tag{5.13}$$

In (5.13) the advantage function is used which measures how good the new policy is with regard to the average performance of the old policy. $\eta(\tilde{\pi})$ can be rewriten into the following form, where instead of the sum over timesteps is the sum over states [176]:

$$\begin{aligned} \eta(\tilde{\pi}) &= \eta(\pi) + \sum_{t=0}^{\infty} \sum_{s} P(s_t = s|\tilde{\pi}) \sum_{a} \tilde{\pi}(a|s) \gamma^t A_\pi(s, a) \\ &= \eta(\pi) + \sum_{s} \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\tilde{\pi}) \sum_{a} \tilde{\pi}(a|s) A_\pi(s, a) \\ &= \eta(\pi) + \sum_{s} \rho_{\tilde{\pi}}(s) \sum_{a} \tilde{\pi}(a|s) A_\pi(s, a), \end{aligned} \tag{5.14}$$

where π is the old policy, $\tilde{\pi}$ is the new policy and ρ is the discounted visitation frequencies: $\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \ldots$. However, (5.14) is hard to optimise due to the dependency of $\rho_{\tilde{\pi}}$ to the new policy $\tilde{\pi}$. Therefore, a local approximation to η is used, which is known as a surrogate function or objective:

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_{s} \rho_\pi(s) \sum_{a} \tilde{\pi}(a|s) A_\pi(s, a). \tag{5.15}$$

In (5.15) L_π uses the state visitation frequency ρ_π instead of $\rho_{\tilde{\pi}}$, assuming that the state visitation frequency of the new policy is not too different from the old policy. Kakade and Langford [176] proved that small steps that optimizes the local approximation $L_{\pi_{\theta_{old}}}$ also improves η. Hence, in order to find an optimal step size, Kakade and Langford proposed a conservative policy iteration update scheme:

$$\pi_{new}(a|s) = (1 - \alpha)\pi_{old}(a|s) + \alpha \pi'(a|s), \tag{5.16}$$

where π_{old} is the current policy, $\pi' = argmax_{\pi'} L_{\pi_{old}}(\pi')$ is the greedy policy and π_{new} is the new policy. This conservative policy iteration gives an explicit lower bound

on the improvement of η.

$$\eta(\pi_{new}) \geq L_{\pi_{old}}(\pi_{new}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha^2, \tag{5.17}$$

where $\epsilon = \max_s \left|\mathbb{E}_{a\sim\pi'(a|s)}\left[A_\pi(s,a)\right]\right|$. This mathematical manipulations give the trust region. However, it worth to mention that such bound only applies on mixture policies which is impractical to use. But, by chaining the constant ϵ, and replacing α with a distance measure between π and $\tilde{\pi}$, (5.17) can be extended to general stochastic policies. Now, we can obtain the following bound:

$$\eta(\pi_{new}) \geq L_{\pi_{old}}(\pi_{new}) - \frac{4\epsilon\gamma}{(1-\gamma)^2}\alpha^2, \tag{5.18}$$

where $\epsilon = \max_{s,a} \left|A_\pi(s,a)\right|$, $\alpha = D_{TV}^{max}(\pi_{old}, \pi_{new})$,
$D_{TV}^{max}(\pi,\tilde{\pi}) = \max_s D_{TV}(\pi(\cdot|s) \| \tilde{\pi}(\cdot|s))$.

Now in order to express (5.18) in terms of KL divergence D_{TV}, which is the total variation divergence, will be replaced by KL divergence, (Pollard [12]):

$$\eta(\pi_{new}) \geq L_{\pi_{old}}(\pi_{new}) - CD_{KL}^{max}(\pi,\tilde{\pi}), \tag{5.19}$$

where $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$. For parameterized policies with parameter vector θ, (5.19) assumes that the expected long-term reward η is monotonically improving as long as the right-hand-side term is maximized. Therefore, we are guaranteed to improve η by performing the following maximization:

$$\max_\theta \left[L_{\theta_{old}}(\theta) - CD_{KL}^{max}(\theta_{old},\theta)\right]. \tag{5.20}$$

In practice, when using a penalty coefficient C in the objective function, the step size will be very small, which leads to longer training time. Thus, a constraint on the KL divergence between the new policy and the old policy is used in order to allow to take larger steps, while guarantee robust performance: $\max_\theta L_{\theta_{old}}(\theta)$, subject to $D_{KL}^{max}(\theta_{old},\theta) \leq \delta$. However the constrained is bounded at every state which is not practical, since there are a infinitely large number of states. Thus, a heuristic approximation is used with the expected KL divergence over states, instead of finding the maximum KL divergence: $\overline{D}_{KL}^{\rho}(\theta_1,\theta_2) := \mathbb{E}_{s\sim\rho}\left[D_{KL}(\pi_{\theta_1}(\cdot|s) \parallel \pi_{\theta_2}(\cdot|s))\right]$.

Now the optimization problem becomes $\max_\theta L_{\theta_{old}}(\theta)$, subject to $\overline{D}_{KL}^{\rho_{\theta_{old}}}(\theta_{old},\theta) \leq \delta$. Therefore, by expanding $L_{\theta old}$, we obtain the following optimization problem:

$$\max_\theta \sum_s \rho_{\theta_{old}}(s) \sum_a \pi_\theta(a|s) A_{\theta_{old}}(s,a) \tag{5.21}$$

In equation (5.21), by replacing $\sum_s \rho_{\theta_{old}}(s)[\dots]$ with the expectation $\frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho_{\theta_{old}}}[\dots]$, and the advantage values $A_{\theta old}$ by Q-values $Q_{\theta old}$, where Q-values are estimated by TD-Learning, will change the objective by a constant. After, we replace the sum over the actions by an importance sampling estimator. Finally, we denote the sampling distribution by q, then the unconstrained loss function can be written as follows:

$$\sum_a \pi_\theta(a|s_n) A_{\theta old}(s_n,a) = \mathbb{E}_{a\sim q}\left[\frac{\pi_\theta(a|s_n)}{q(a|s_n)} A_{\theta_{old}}(s_n,a)\right] \tag{5.22}$$

Finally, by taking into account that the sampling distribution $q(a|s)$ is equivalent to $\pi_{\theta old}(a|s)$, the optimization problem at timestep t in terms of expectation will be as follows:

$$\max_{\theta} \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t \right] \tag{5.23}$$

$$subject to \mathbb{E}_t \left[D_{KL}(\pi_{\theta_{old}}(\cdot|s_t) \parallel \pi_\theta(\cdot|s_t)) \right] \leq \delta. \tag{5.24}$$

Figure 1 presents the pseudocode of the described algorithm.

Algorithm 1: TRPO algorithm

for $i = 1, 2, \ldots$ **do**
  run policy $\pi_{\theta_{old}}$ for T timesteps or N trajectories ;
  compute advantage estimates $A_1, \ldots, A_T$;
  optimize surrogate objective $\max_\theta \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t \right]$, subject to
  $\mathbb{E}_t \left[D_{KL}(\pi_{\theta_{old}}(\cdot|s_t) \parallel \pi_\theta(\cdot|s_t)) \right] \leq \delta$;
  $\theta_{old} \leftarrow \theta$
end

Although the learning algorithm is an essential driving force for developing a successful reinforcement learning-based controller, there is another important component that highly impact on the agents' behaviour and performance, known as a reward or objective function.

Reward Function

With the increased interest in deep reinforcement learning methods, the number of research that aims to answer the question of "How to design a reward function properly?" has grown significantly. As it was stated above, in reinforcement learning, an engineer defines the desired outcome by simply crafting a reward function, a real-valued number, that the agent receives at some specific states. The agent's goal is to transition between states and discover the ones that lead to higher rewards while avoiding the states with low or negative rewards. Thus, a proper design of a reward function, that adequately represents the desired behaviour (outcome), plays a critical role when building reinforcement learning models.

For instance, one of the most straightforward ways to design a reward function is to give the agent a reward when it achieves the given goal and a punishment otherwise. However, getting such narrowed feedback from the environment may result in infinite explorations. Thus, such sparse reward functions rarely provide successful learning. Efficient reward functions guide the agent along the way, since they provide continues information about its environment as well as behaviour. This way of defining the reward is known as reward shaping, and has been commonly used in a variety learning tasks [56, 21, 138]. In this work as well the designed reward functions are using the concept of reward shaping providing the agent feedback throughout its entire performance. Although a detailed design of reward functions are crucial for success in any learning task, often, it may not be enough for obtaining the desired outcome. Recently, a research paper has shown anecdotes illustrating artificial intelligent agents behaviours, that were able to discovering unintended loopholes in the reward functions [102] and escape from learning the challenging

task. The agents learned to maximize their reward functions in surprising and creative ways, which, however, resulted in completely different and not desirable behaviours. This issue is also known as reward hacking. In this work, we have also have experienced similar behaviour from our learning agent when instead of learning a flight navigation task, the hexacopter found out that by staying parked on the ground it can maximize the cumulative reward faster than if taking new actions.

Finally, when designing a reward function it is important to remember that the only objective of the agent is to maximize the cumulative reward. Thus, to avoid such incidents, before executing lengthy training, it is important to perform an adequate observation and debugging of the reward function that represents the desired policy. In the next section we demonstrate the formulation of the reward that has been engineered in for training our learning agents. The proposed formulation is a result of extensive experiments.

5.4 Experimental Validation

To validate the effectiveness of the proposed reinforcement learning-based controller, two experiments have been conducted in the RotorS framework [117]. In the first scenario, a hexacopter UAV, with 6-DoF, is trained to perform a stable hovering task in continuous state-action spaces. The second scenario extends the first model by means of learning to navigate towards the given goal position, where the agents learn to continue hovering for a defined number of timesteps.

Before discussing the obtained results, first, we present the RotorS Gym Framework, a multifunctional platform, designed to facilitate the implementation of intelligent flight control systems using reinforcement learning baselines in the continuous observation space domain. Then we detail the workflow of the controller, along with the neural network architecture and the formulation of the reward function. We conclude the chapter with great discussions about the results of both scenarios.

5.4.1 Environment Setup

Training a real robot in a real world, using RL techniques, can be expensive due to large number of explorations that the agent has to perform, many of which result in crashes with high risk of damaging the robot. Therefore, having a framework that provides close to real-world dynamics is necessary. There, the training of the robot can be performed safely and the optimal policy can be learned efficiently, which can later be run on the real robotic system, with only minor adaptation.

To provide such an environment for our learning agent, we designed the RotorS Gym framework, which provides a generic interface between the learning algorithm and agent in a simulated environment, ensuring a seamless workflow. Because of its modular properties it enables switching between different baselines as well as simulated aerial robotic models with ease. Later, we have proposed Cyber Gym Robotics platform, that extends RotorS Gym framework, that provids a novel concept for designing complex missions for robots with highly nonlinear dynamics. The Cyber Gym Robotics have been presented in great details in chapter 4. However, in this chapter, the training and further experiments have been performed in the RotorS Gym Framework.

Learning agent

In reinforcement learning one of main elements is the learning agent. The learning agent interacts with its environment and based on the received feedback signals learns to perform a desired behaviour. The environment where the agent operates can be either simulated or a real-world setting. However, when an agent learns a control behaviour trough trials and errors, this can lead to numerous catastrophic crashes. Moreover, considering the fragile nature of robots, such way of training is highly expensive, sometimes even life threatening, therefore, also non desirable. One way to tackle this challenge with real-world interactions is to start the training in a simulated environment generating samples in an inexpensive way, and only after the desired behaviour is achieved transfer the model on a real physical robotic platform. Although this approach may seem an optimal solution, but to achieve successful performance the model of the learning agent should be analogous in both environments.

As it was stated above, in this work, as a learning agent the simulated model of an AscTec Firefly has been used. It is a digital replica of a real AscTec Firefly hexacopter of 6-DoF, provided by Furrer et al. [49]. The authors designed a Gazebo-based modular Micro Aerial Vehicule framework, called RotorS MAV. Besides the Firefly model RotorS MAV also provides models of other multirotor systems for experimental research. Additionally, all components and sensors that are commonly found on real MAVs, are simulated by Gazebo physics engine and its different plugins. In the framework a position controller and a state estimator is also available, however, they are outside the interest of this work.

The digital replica of the hexacopter consists of a body frame and six motors and is presented in figure 5.1 . The hexacopter is actuated by changing the angular velocities of each motor ω_i ranging between 500 to $600 rad/s$. These values have been derived from the actual model of a real AscTec Firefly aircraft. Below each learning scenario, along with obtained results, are presented.

5.4.2 Hexacopter Hovering Scenario

The learning scenario is composed of an AscTec Firefly UAV, with 6-DoF, operating in a 3-dimensional simulated environment. The agent starts at a predefined position of $x = 0$, $y = 0$, $z = 1.02$, with an objective of learning to perform a continuous stable hovering task, while remaining as close as possible to the initial state. The closer and longer the agent stays to its objective position, the greater reward it receives.

As it was stated before, all operations have been performed in RotorS GYM framework.

RotorS GYM Framework

RotorS Gym framework provides an environment for solving reinforcement learning tasks that observe continuous robotic control problems. It can be represented as a combination of two main blocks: 1) a mission controller and 2) a learning agent. The communication between these blocks is handled by ROS, which provides a publish-subscribe messaging system. The general workflow of the RotorS Gym framework is illustrated on figure 5.3.

The mission controller inherits the architecture of OpenAI Gym and implements the following basic methods:

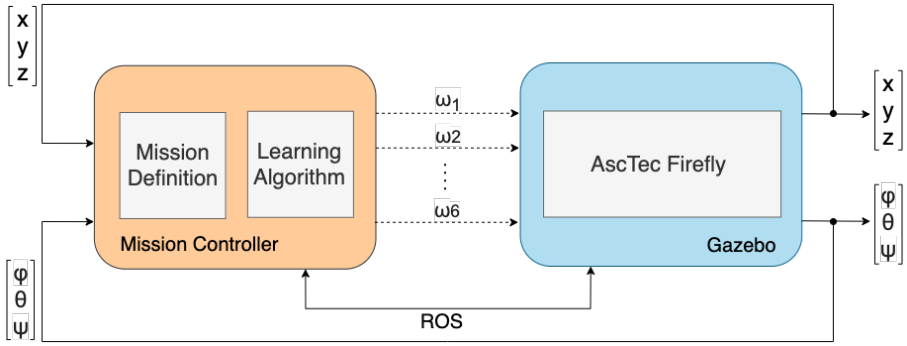


FIGURE 5.3: The general workflow of the RotorS Gym framework consisting of two main blocks: (1) a learning agent, which is the simulated model of the AscTec Firefly and (2) the mission controller.

- **init()** - initializes the robot, the simulated environment if necessary and setup ROS-specific communication channels.
- **step()** - is used to execute the actions at each step. After an action is executed, the $step()$ must function returns three essential values: **observation**, which is an environment-specific object, containing different information about the agent, for instance, its position and/or orientation, or the motors rotational velocities. **_reward** is a floating-point number that the agent gets after each performed action. **_done** is a boolean value that indicates whether the robot ended its mission regardless of success or failure. *Done* is *True* when the maximum number of steps have been accomplished or if the agent overpasses the predefined constraints.
- **reset()** - this function is called when an episode starts in order to reset the environment and setup the initial environmental conditions.

The learning process in RotorS Gym is executed in episodic setting, where the agent learns to perform the given task through a series of episodes. At each episode, the agent interacts with its environment until it reaches a terminal state. There are a few ways to define the terminal states: (1) by defining conditions in the reward function, (2) by setting a timestep. When the agent reaches the terminal state, the environment resets, the agent returns to its initial state, and the process repeats. The agent aims to maximise the expected reward at each episode and achieve the desired performance as fast as possible.

In our application example, the environment is assumed to have 6-dimensional state and action spaces. The continuous action space consists of $\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6 \in [500,\ 600]$ rad/s rotors velocities. The state space is represented by $(x,\ y,\ z)$ position and (ϕ, ψ, θ) orientation of the UAV.

The main workflow of the controller shown in figure 5.3 can be described as follows: when the episode starts at each timestep, the controller receives the position and orientation of the UAV and outputs six actions representing six rotors velocities. The agent transitions to a new state receiving feedback from its environment in the form of a reward. The process continues until the agent reaches the terminal state. After the cumulative reward over the entire episode is computed and a new episode is initialised.

Reward Function

In reinforcement learning, the reward function is one of the most fundamental components, since the policy that the agent aims to discover is expressed through this function. Often, the reward function is called environmental feedback which the agent uses to estimate its performance. If a careful design is not performed, the derived policy may not reflect what was originally intended, or may require a tremendous amount of explorations before the desired behaviour is obtained. Therefore, careful crafting of the reward function that adequately represents the desired behaviour is crucial.

In this work, the reward function has been designed in a way, that it guides the agent towards finding its goal state. It is not only pointing out the failures and/or successes but also providing instantaneous information throughout the whole performance.

The reward function used for training our learning agent consists of two subfunctions: (1) indicate the failure and give a negative reward, (2) provide continuous gradual information along the way. When a failure is indicated, the agent receives a negative reward, and the environment resets. This helps to decrease the probability of choosing the action that caused the failure, once the agent arrives in this state in the future. It also prevents unnecessary or undesirable exploration and directs the agent on practising the most important regions of the state space. The second subfunction aims to minimize the distance between the agent's current and goal position as well as the values of the pitch and roll angles. Such a design of reward function provides a smooth gradient of rewards as the agent approaches the objective.

Before defining the main reward function, there are a few important points that need to be considered. First, the initial and desired states should be determined. In the case of the hovering task, both, the initial and desired states are the same: $[x,\ y,\ z] = [x^*,\ y^*, z^*] = [0,\ 0,\ 1]$, where $[x,\ y,\ z]$ is the agents current position, whereas $[x^*,\ y^*, z^*]$ is the desired positions, respectively.

The next step is to define the termination states. Here constraints are set on the distance between the desired and current states of the agent as well as on its pitch and roll angels: $d \geqslant 2m$, $\phi \geqslant \pi/2$, $\theta \geqslant \pi/2$ where d is the distance between the desired and current states, computed as follows: $d = \sqrt{(x^* - x)^2 + (y^* - y)^2 + (z^* - z)^2}$. If these thresholds are exceeded, the learning process is being terminated, the environment is reset to its initial conditions and a new episode is initialized. If none of these conditions are met, the episode ends after 650 environmental interactions (timesteps). Finally, the main reward function is structured as follows:

$$r = d_{discounted} * \phi_{discounted} * \theta_{discounted} \tag{5.25}$$

where $d_{discounted}$ is the discounted distance between the desired and current states.

$$d_{discounted} = 1 - (d/d_{max})^{0.4} \tag{5.26}$$

The idea behind the $d_{discounted}$ function is to provide sharper gradient towards the goal. Thus, to normalise its values we first divide the current distance by the maximum distance and after raise over 0.4.

Although, shaping the distance may enable the agent to learn to reach the given goal position in a few training episodes, such formulation may not be enough for learning a stable hovering task. This kind of reward function does not provide any

information about the pitch, roll and yaw angles of the agent. Whereas, for performing stable hovering at a goal position the agent should learn to maintain values of these angles closer to zero. Thus, to provide complete information we jointly shape the distance along with pitch and roll angles, which allows the agent to not only learning to minimize the distance, but to also learn to stabilizes its orientation.

Finally, the discounted functions for ϕ, θ are as follows:

$$\phi_{discounted} = 1 - \phi^{1/d} \tag{5.27}$$

$$\theta_{discounted} = 1 - \theta^{1/d} \tag{5.28}$$

Lastly, the reward function that the agent receives for each performed action is the multiplication of 5.26 and 5.28 discounted functions. The graphical representation of the reward function is shown on figure 5.4.

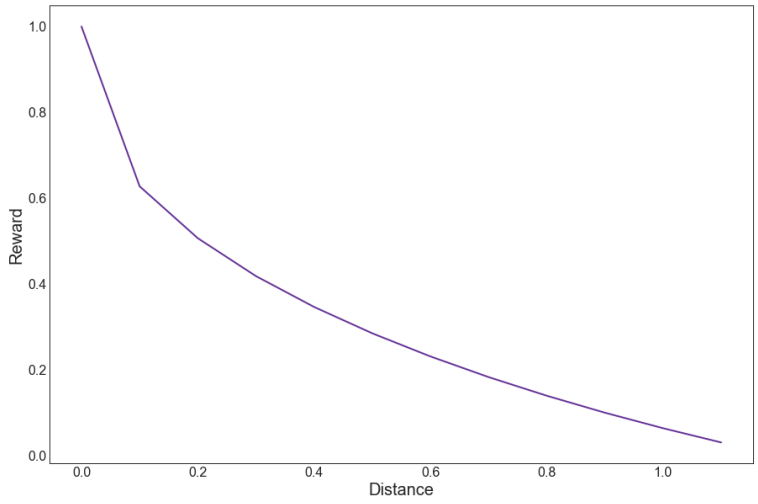


FIGURE 5.4: The graphical representation of the reward function composed of discounted distance and rotational values.

It is worth noting, that designing an optimal reward function is one of the most challenging tasks in reinforcement learning, and requires tremendous amount of explorations and experiments. The reward function presented above provided an efficient learning of different tasks. Thus, in this work, this formulation of the reward function has been used as base when defining the other learning tasks, presented later in this thesis.

Network Architecture

Yet another essential component in reinforcement learning is the function approximator. Because of its significant and direct impact on the resulted policy, it must be designed very carefully. In this work, the function approximator is represented by a neural network. As stated before, TRPO employes an actor-critic architecture, where the actor model is the policy function, and the critic model is the value function. Hence, two different neural networks have been constructed representing the policy and value function approximations.

We use a two layer feedforward neural network of 400 and 300 hidden nodes respectively, with rectified linear units (ReLU) between each layer for both the actor and critic, and a final tanh unit following the output of the actor. As shown in figure 5.5, the input of the actor-network is a 6-dimensional vector, representing the $[x,\ y,\ z]$ position and $[\phi,\ \theta, \psi]$ orientation of the UAV. As output, it returns a 6-dimensional vector, consisting of $[\omega_1, \omega_2, \ldots, \omega_6]$ representing the rotational speed of each motor. On the other hand, the critic network evaluates the performance of the actor. It receives $[x, y, z]$ position and $[\phi, \theta, \psi]$ orientation of the agent along with the actions, where Q-values are estimated using TD-Learning, and based on this information outputs a value, and updates the weights of the network. Both networks consist of two hidden layers. The parameters of both networks are initialized randomly and are optimized using Adam optimizer [92].

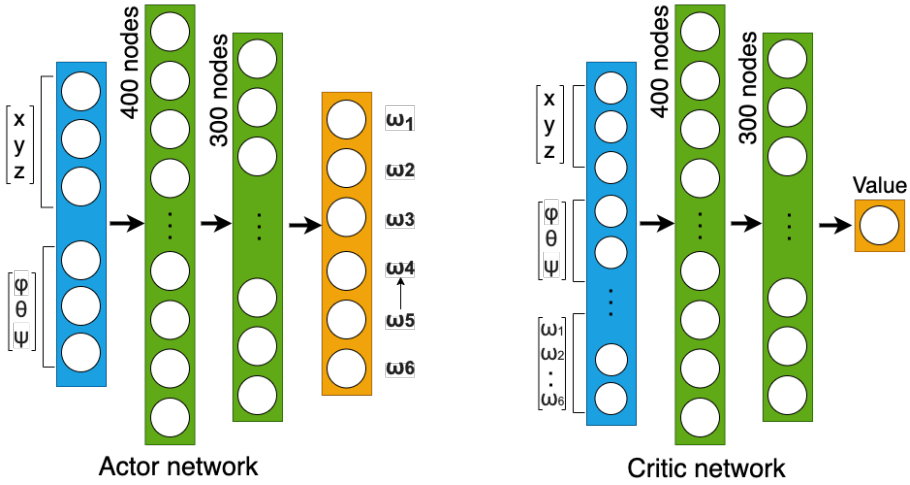


FIGURE 5.5: The architecture of actor network and critic network. Both networks consist of two hidden layers 400 and 300 nodes, respectively.

Hyperparameters

Tuned hyperparameters are yet another crucial elements in eliciting the best results for many algorithms. They play an important role when it comes to reproducibility of the proposed approach. However, the choice of hyperparameters may strongly differ across different learning algorithms and environments. Hence, to find the best suitable set of hyperparameters we have performed numerous experiments while varying one specific parameter at a time.

- $\alpha = 0.2$, learning rate, defines how often the parameters of the networks are updated
- $\gamma = 0.9$, discounted factor is used to indicate if the agent relies more on immediate or future rewards. A lower value emphasises an immediate reward.
- *batch size* is set to 32, which represents the number of experiences used for one iteration of a gradient descent update.
- Number of epochs specifies the number of passes through the experience buffer. A smaller number provides more stable updates, but slower learning

- $\epsilon = 0.2$, is a threshold of divergence between the old and new policies used during the gradient decent update step. A lower value will result in more stable updates but slower training process.

Moreover, the maximum time of an episode is set to 1000 timesteps, with a project step time of $t_s = 0.02s$, results in a maximum episode time of $t_{max} = 20s$.

It is worth noting that a constraint on maximum timesteps of an episode has been set only during the training. After the agent learned the objective behaviour and during demonstration, no time constraint is enforced and the simulation can only be manually interrupted. As the agent learned the task, it is able to indefinitely perform the task without crashing.

The project step time of $0.2s$ ($200ms$) has been chosen in preparation of running the learned model on a real robotic platform with an on-board computer. As computing the neural network is relatively resource-intensive and the on-board computer might not have the same computational power as the hardware used during training, the project step time allows us to implement a so-called reaction time close to a real-life scenario, where the UAV on-board computer must be able to process the environment, compute new actions and apply them in $200ms$ or less. Choosing a too small project step time is likely to create a situation where the real UAV is unable to perform the required computation in time, therefore falling behind and reacting lately to environmental changes, which will result in a catastrophic crash. A conservative value, such as $200ms$ avoids such a problem, but does not limit the UAV to perform faster computations as well.

Experimental Results

This section aims at analyzing the obtained results of the proposed learning-based controller applied to the hexacopter stable hovering task. Before presenting the results, we first recall what the performance metric is. The target of this reinforcement learning task is to guide an agent to achieve the goal in a continuous action-space domain by maximizing the cumulative reward.

The metric by which we measure the performance is the error rate with respect to distance d and ϕ and θ angles. We hereby list our metrics: success is defined when the error rate of d is less than or equal $0.1m$, whereas the error rate for ϕ and θ is set to less than or equal 0.1 radian. The learning performance must be considered failed and terminated if these conditions have not been achieved after 10000 episodes of training, which is equal to 5 million environment interactions. This may indicate either misspecification of the reward function, poor choice of hyperparameters and/or need of optimizing the neural networks structure. The maximum number of episodes has been chosen based on our past experience as well as other studies about convergence of reinforcement learning algorithms [73, 64].

The task is considered solved when the agent remains in the success state for more than 100 consecutive environmental interactions.

As stated above, several constraints on different environmental parameters are defined before starting the learning process. For instance, one single episode is limited to 650 environmental interaction. If the agent reaches this threshold, the environment is reset, the agent is placed on its initial starting position, the cumulative reward of the current episode is computed, and the next episode starts. One step in an episode, which can be defined as one environmental interaction, is equal to $200ms$ of simulated time which give the controller enough time to perform calculations and allow the agent to perform new actions and reach a steady state. At each step, the

agent chooses actions from a set of actions, describing each motor's rotational velocities. The velocity values are bounded in the range of $500 rad/s$ and $600 rad/s$. The limits have been provided by RotorS MAV model [49].

The training and evaluation have been executed on a machine equipped with $16GB$ of RAM and a $2.6Ghz$ Intel $i7-6600U$ CPU running on Ubuntu 16.04 operating system.

To draw a clearer view of the whole course of training and present how the performance of the agent has evolved within the millions of environmental interactions, we have randomly selected three different models of obtained policy representing the learning process from early to late stages.

Early stage of the learning process

The early stage of the learning process is when the agent prioritizes exploration over exploitation. By performing random actions it tries to find out the policy that leads to maximizing the cumulative rewards. This section focuses on the analysis of the agent's learned behaviour after about 100 training episodes. Figure A.7a shows the graphical representation of the the agent's position in three dimensional space. As the agent starts at the goal position and has to learn to hover there, we see that the agent is failing at that task and is actually crashing down very quickly.

To provide greater details we first demonstrate the performance on each x, y, z axis as shown in figures A.1b, A.1c and A.1a. Then, figures A.2a, A.2b and A.2c show the representation of each ϕ, θ, ψ angles. Thus, figures A.1a, A.1c and A.1b present that the episode lasts only about 20 environmental interaction steps, with a distance error greater than $1m$. After just 7 steps, the agent has crashed on the floor. The episode ends due to predefined constraints being met, thus returning a negative reward.

At the same time, figure A.2a and A.2b pinpoint an unstable behaviour of the hexacopter with a rotational error rate fluctuating between $-0.4rad$ to $0.4rad$, due to the agent crashing on the ground.

Middle stage of the learning process

The next step is to present the agent's performance after about 1000 episodes of training, which is about 500000 environmental interaction steps. At this point the agent frequently arrives at the success state. However, the learning task can not be considered as solved, yet. As was discussed above, the task is considered solved only if the agent remains at the success state for more than 100 consecutive environmental interaction steps.

By first taking a look at the three dimensional representation in figure A.7b, we can observe that, indeed, the agent stays closer and longer to the goal position. The agent learned to hover, but does not yet manage to do it in a stable manner, thus hitting the boundaries and forcing the simulation to restart.

The graphical representation of the agent's behavior during this phase is shown in figure A.3b, A.3c, A.3a and A.4a, A.4b, A.4c.

Compared to the early stage, the distance error rate has decreased from $1.2m$ to $0.25m$. In the beginning of the flight the agent is able to achieve stable hovering at the success state for about 80 consecutive environmental interaction steps. Still early in the learning, the agent is still exploring different rotational speed for each motor, leading it to create fluctuations in the flight path.

Although the agent's altitude varies after 80 steps, looking at figures A.4a and A.4b we can see that the stability in the performance remains unchanged up to 200 steps with a rotational error rate of less than $0.1rad$. The episode ends after 80 environmental interaction steps, because the agent became unstable and was unable to recover from increasing fluctuations.

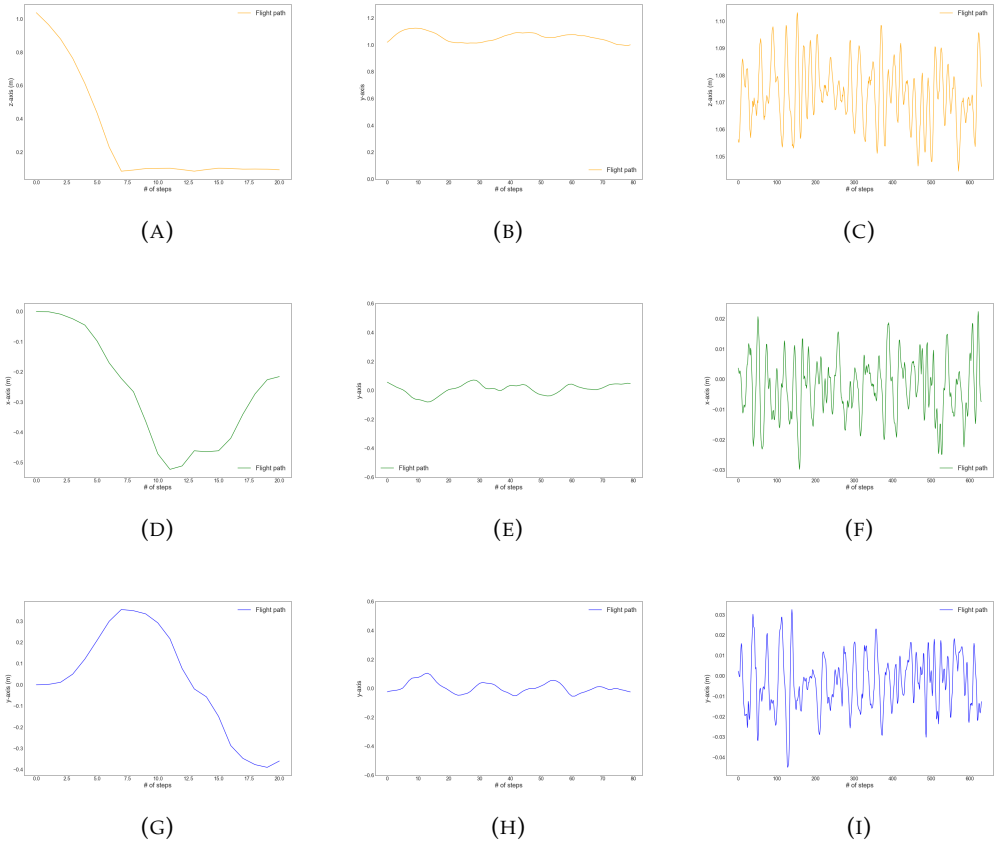
As it was noted above, the environmental constrains are being adjusted during the entire learning process which helps to accelerate the training.

Later stage of the learning process

Finally, after about 11000 training episodes, translating to more than 5.5 million environmental interactions, the agent demonstrates a remarkable performance as shown in figure 5.16. At this point, the agent is able to maintain a stable hovering state for the entire duration of an episode, which is 650 environmental interactions. By further analysing the the performance of the agent, we can see in figure A.5b, A.5c A.5a and A.6a, A.6b, that the distance error and the rotational error have decreased to $0.03m$, respectively to $0.02rad$.

At this point, the task can be considered learned and can be run in a simulated test environment, which differs from the training environment in a sense that the task is continuous. There as well, the agent successfully hovers at the goal position as long as the simulation is running. Several video clips have been recorded, highlighting the performance of the agent during the different learning stages and are available at [123].

Finally, figure 5.9 presents the average accumulated reward for about 20000 training episodes.

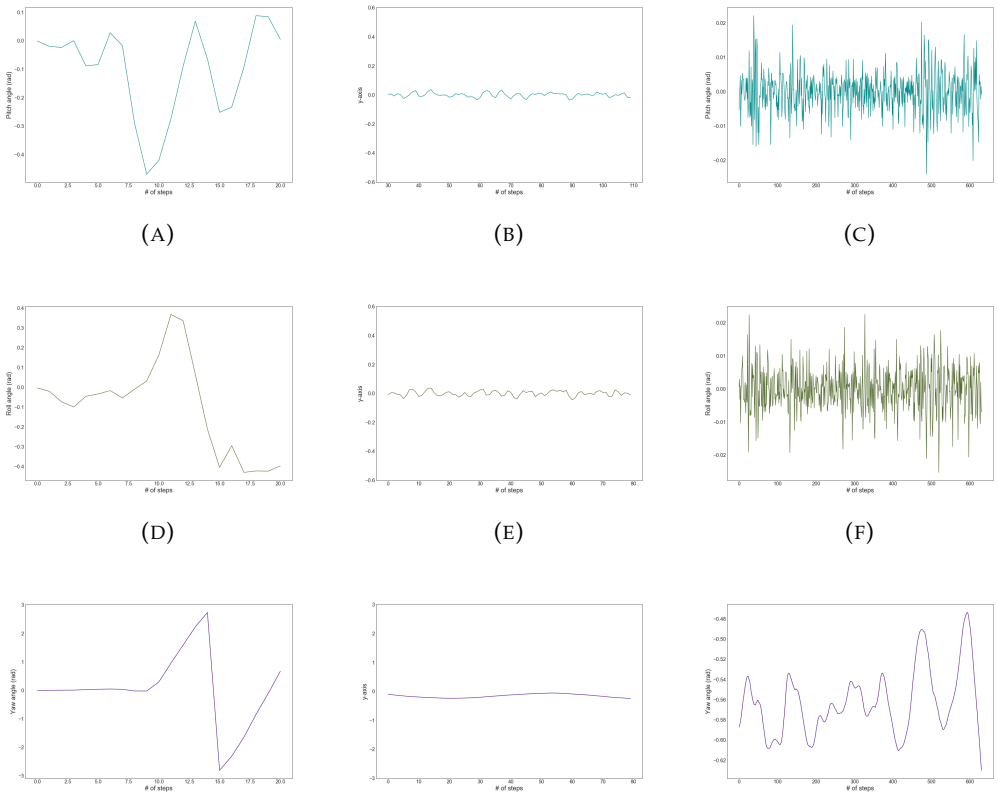


(A) (B) (C)

(D) (E) (F)

(G) (H) (I)

FIGURE 5.6: The graphical representation of the agent performance on x, y and z axis in early, middle and later stages of the learning process.

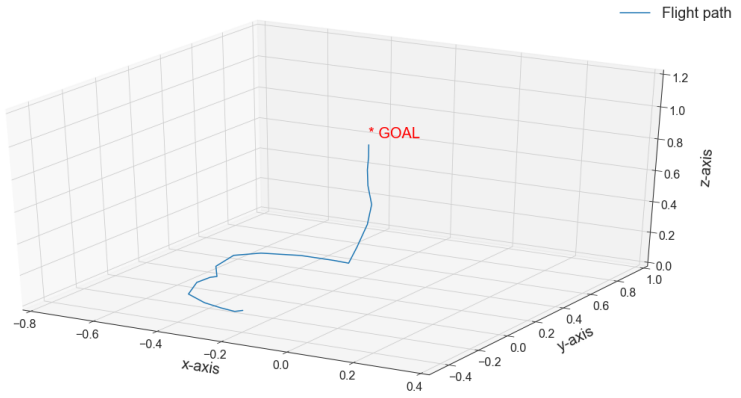


(A) (B) (C)

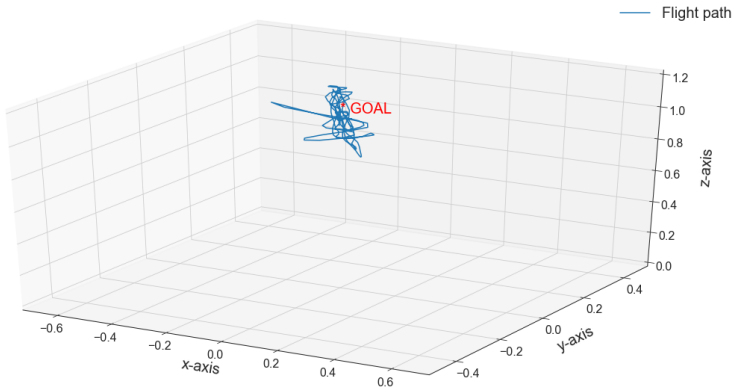
(D) (E) (F)

(G) (H) (I)

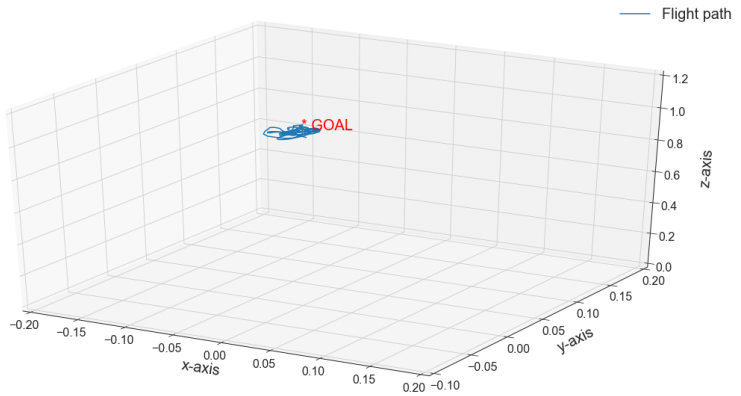
FIGURE 5.7: The graphical representation of agent's ϕ, θ and ψ angles in early, middle and later stages of the learning process.



(A)



(B)



(C)

FIGURE 5.8: 3*D* representation of the hexacopter's flight path.

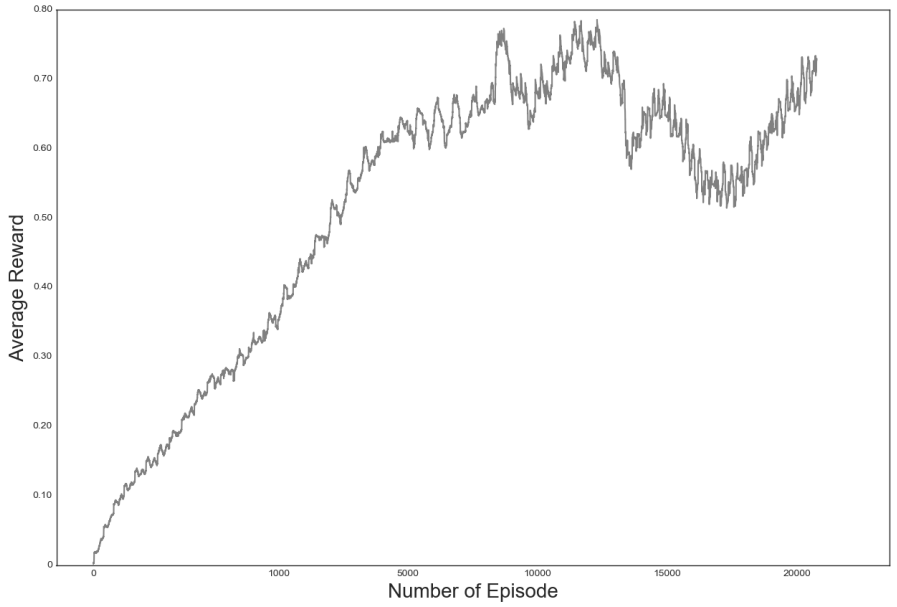


FIGURE 5.9: The accumulated reward growth during training for about 20000 episodes.

5.4.3 Hexacopter Navigation Scenario

To expand the agent's capabilities, another learning task has been designed. The scenario of this second learning task consists of an agent aiming to learn a mission composed of a takeoff, navigation and stable hovering task. During this scenario the environment and the agent remain the same as in the first scenario.

The idea behind the second learning task can be defined as follows: a hexacopter AscTec Firefly, with 6-DoF, operates in a 3-dimensional simulated environment with continuous action-space settings. At each episode the agent starts its performance at the world origin $x = 0m$, $y = 0m$, $z = 0m$ with an objective of arriving to a given goal position, $x = 0m$, $y = 0m$, $z = 1m$, where it should maintain a continuous stable hovering state for as long as possible. The reward function, hyperparameters and the neural network architecture remain unchanged.

Experimental Results

To provide a detailed overview of the learning process we randomly select three different models representing different phases of training.

Before starting the training, the environmental constraints are set. The maximum environmental interaction in one episode is limited to 650. Exceeding this value halts the training and resets the environment. As before, the performance is evaluated using the error rate of the distance as well as ϕ, θ and ψ angles as metrics.

The agent is considered to achieve a success state if it maintains a distance and orientation error rates below $0.2m$. Additionally, the learning problem is considered solved when the agent is able to arrive to the given goal position and maintain a continuous and stable hovering state for more than 70 environmental interaction steps.

Early stage of the leaning process

To represent the early stage of the learning process we have chosen the model learned after 500 episodes. Figures A.8b, A.8c, A.8a show that shortly after taking off, the agents reached $0.3m$ before losing control and crashing to the ground, thus ending the simulation with a distance error rate greater than $1m$. The graphical representation of the angle's ϕ, θ and ψ angles are shown in figure A.9a, A.9b, where it is seen that the error rate of ϕ and θ angles are greater than $0.2rad$. Figure A.14a confirms the crash in a three dimensional representation.

Middle stage of the leaning process

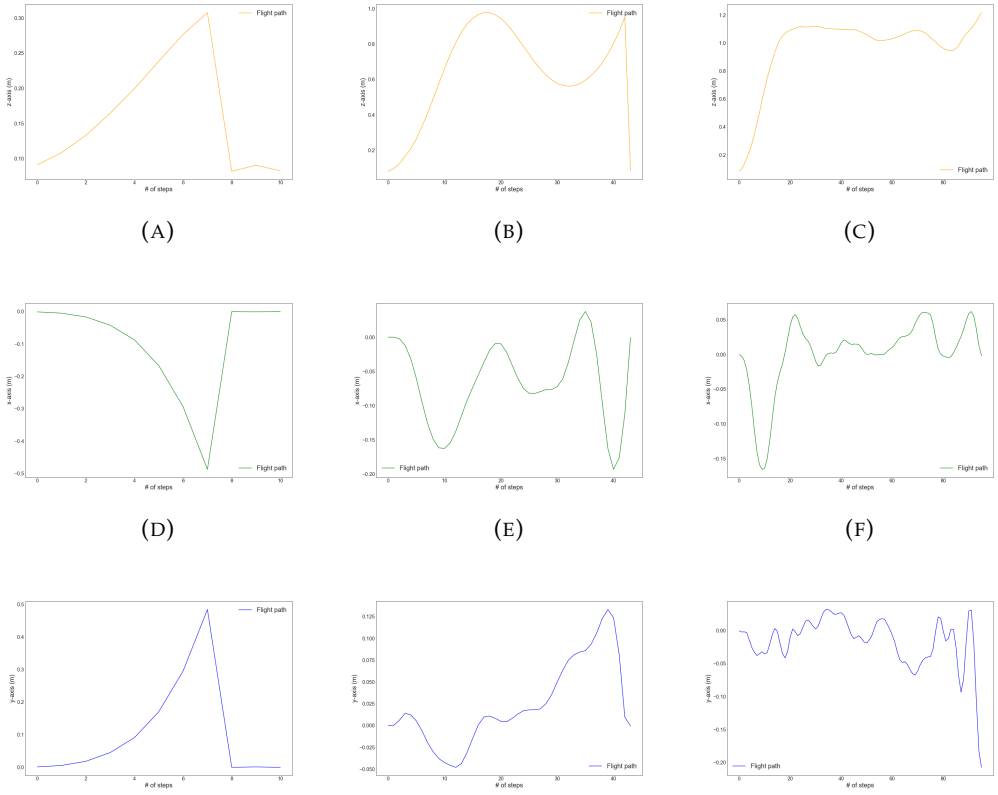
After about 7000 training episodes the agent improves its performance by arriving to the desired position and maintaining hovering state for about 40 environmental interactions. Figure A.14b shows the unstable flying behaviour of the agent while trying to solve the task. We can observe that the agent learned where to fly to but is still unable to do it in a stable and controlled manner, thus overshooting the goal position several times. Therefore, event though the desired state has been reached at least once, the task can't be considered solved as the distance error is higher than what we defined as acceptable. The graphical representation of the flight along with ϕ, θ and ψ values is shown in figure A.10b, A.10c, A.10a and A.11a A.11b A.11c, respectively, and further demonstrating the struggle of the agent to remain stable.

Later stage of the leaning process

During a later stage the agent discovers its objective after about 14000 training episodes as seen in figure A.14c. At this point, the agent is able to fly to the destination and maintain a stable hovering state for more than 80 consecutive environmental interactions with an error rate of less than $0.2m$ as shown in figure A.12b, A.12c, A.12a and A.13a, A.13b, A.13c, respectively.

Finally, figure 5.13 presents the average accumulated reward during the training of about 22000 episodes. We can see that the cumulative reward per episode grows larger and larger after about 6000 episodes, which is equivalent to $3million$ environmental interactions.

For this scenario as well, several video clips have been recorded to provide a clear overview of the entire training process [122].

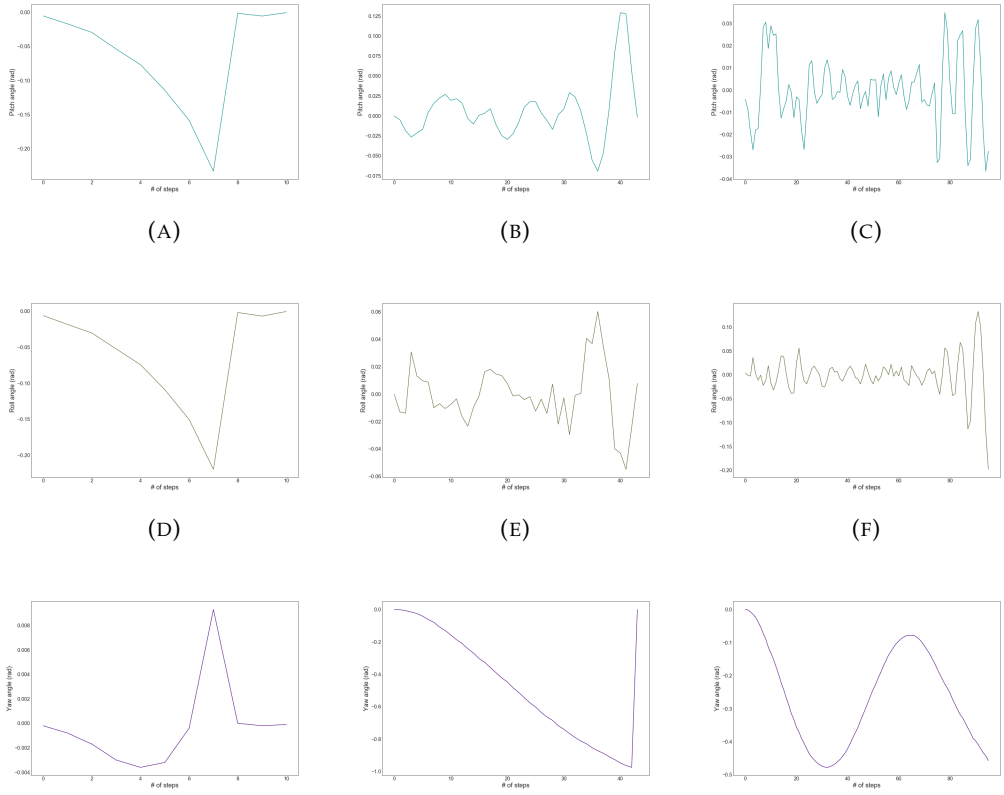


(A) (B) (C)

(D) (E) (F)

(G) (H) (I)

FIGURE 5.10: The graphical representation the agent's performance on x, y and z axis in early, middle and later stages of the learning process.

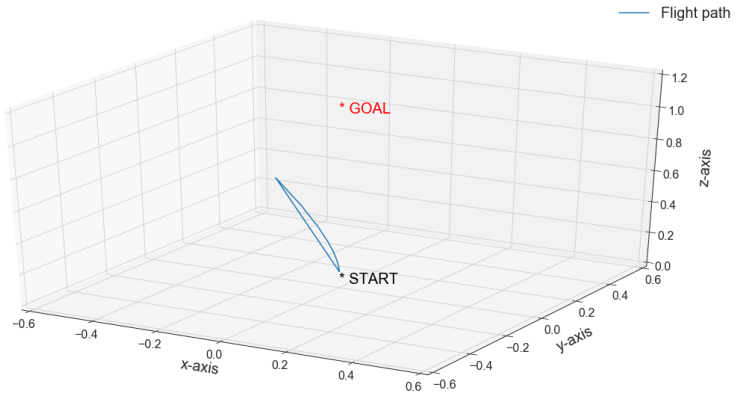


(A) (B) (C)

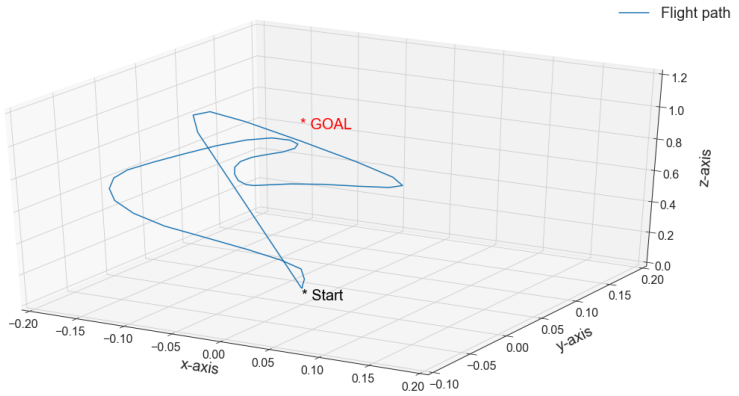
(D) (E) (F)

(G) (H) (I)

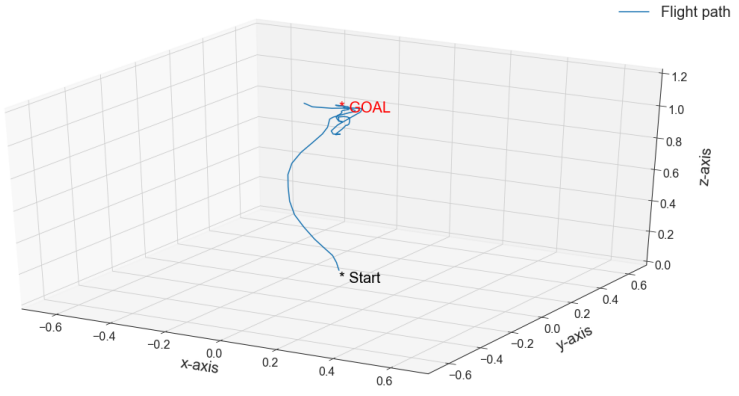
FIGURE 5.11: The graphical representation of the agent's ϕ, θ and ψ angles in early, middle and later stages of the learning process.



(A)



(B)



(C)

FIGURE 5.12: 3D representation of the hexacopter's performance.

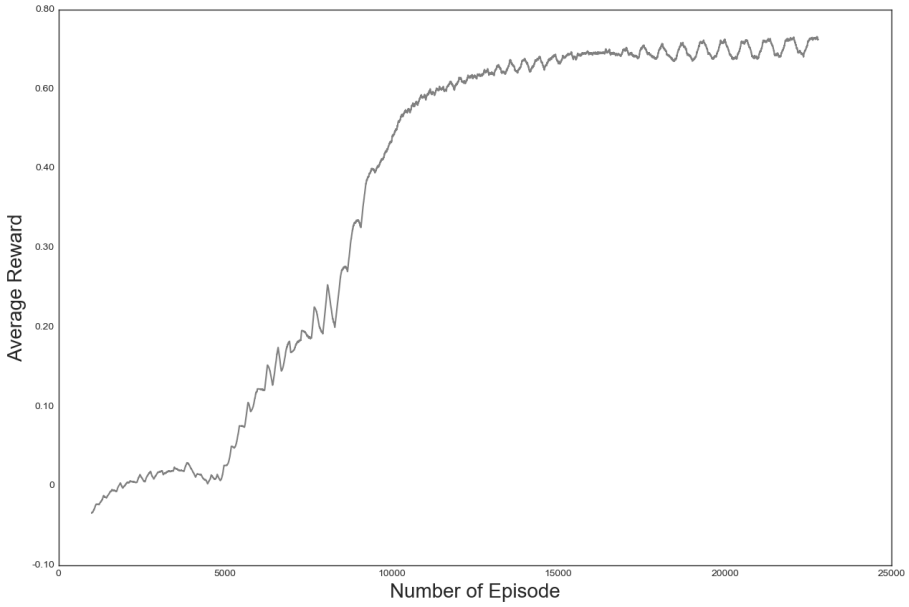


FIGURE 5.13: The accumulated reward growth during training for about 23000 episodes.

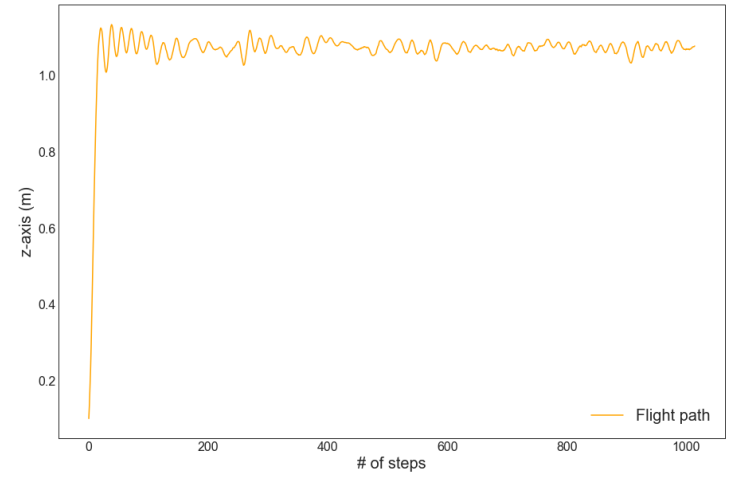
5.4.4 Concept-based Mission Controller

Although the agent demonstrated successful performance for previous learning scenarios the performance can be further improved when applying the concept-based mission controller proposed in chapter 4.

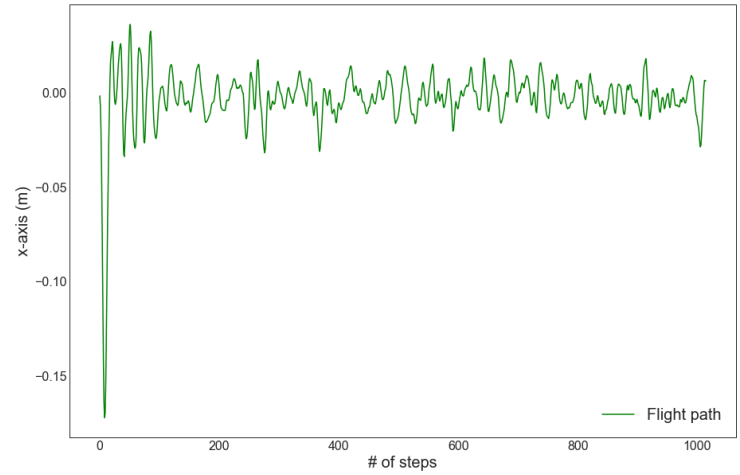
More details about Cyber Gym Robotics along with the mission controller will be given in chapter 4 and 6, however, below we present the results obtained for above discussed learning scenarios. By choosing the best model for navigation as well as for hovering, the agent is able to reach the goal position, at which point the concept changes and the agent continues to maintain a stable hovering state using the appropriate model.

The graphical representation of the agent's flight performance along with the ϕ, θ and ψ angles are shown in figure 5.14 and 5.15. Figure 5.16 presents the $3D$ representation of the flight and visually confirms the success of the mission. It is worth noting that the concept-based approaches demonstrates an overall improved performance compared the to the second scenario. This can be explained by the fact that the concept-based mission controller uses the learned model that is best suited for the current environmental setting.

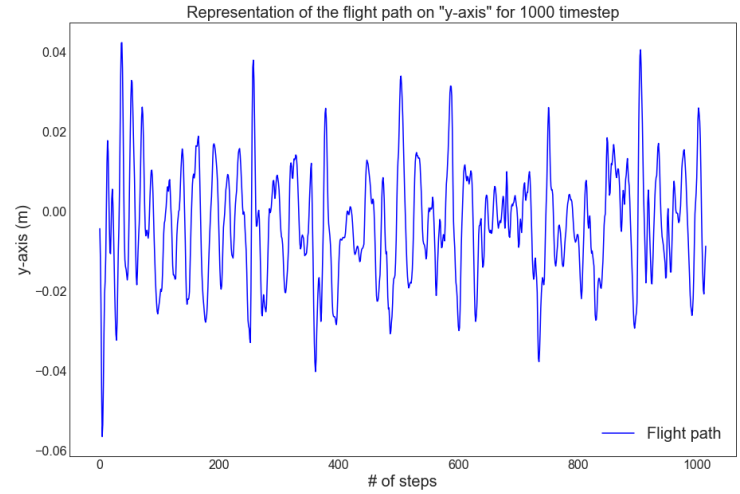
Additionally, several video clips have been recorded to provide a clear overview of the entire training process [124].



(A)

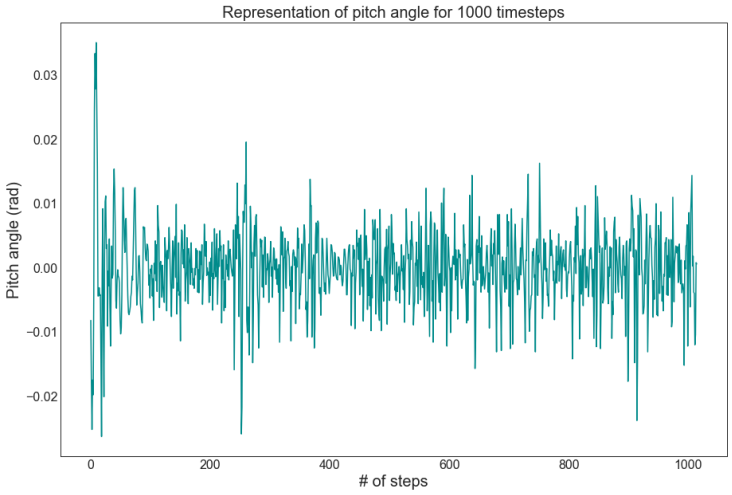


(B)

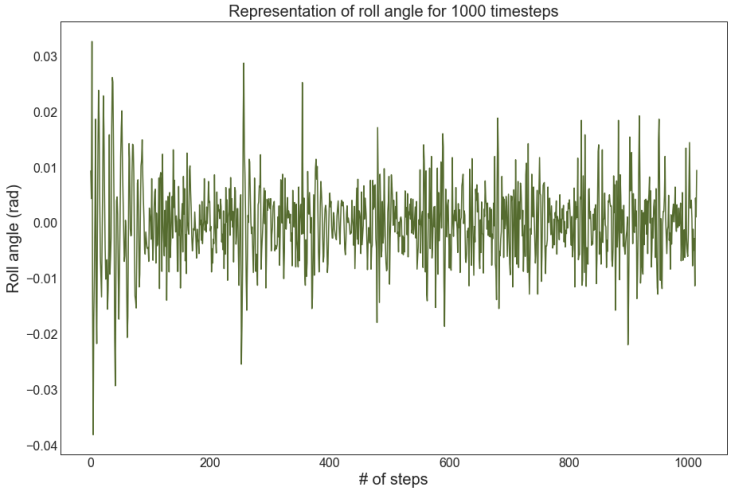


(C)

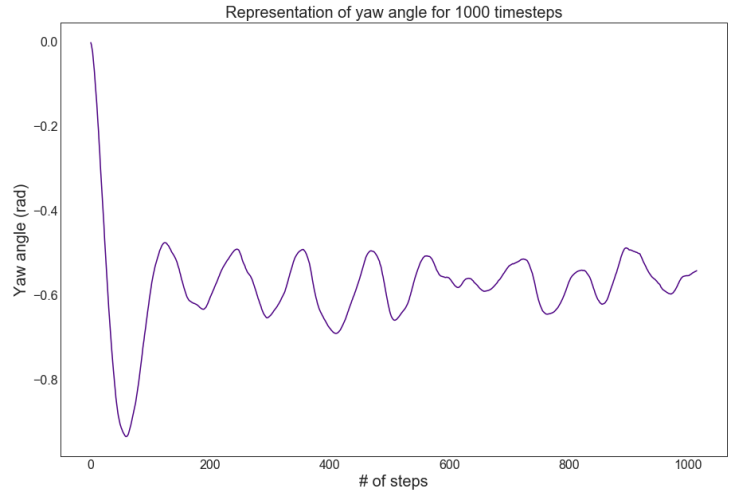
FIGURE 5.14: The graphical representation of the agent's performance on x, y and z axis using concept-based mission controller.



(A)



(B)



(C)

FIGURE 5.15: The graphical representation of the agent's ϕ, θ and ψ angles during its performance, using concept-based mission controller.

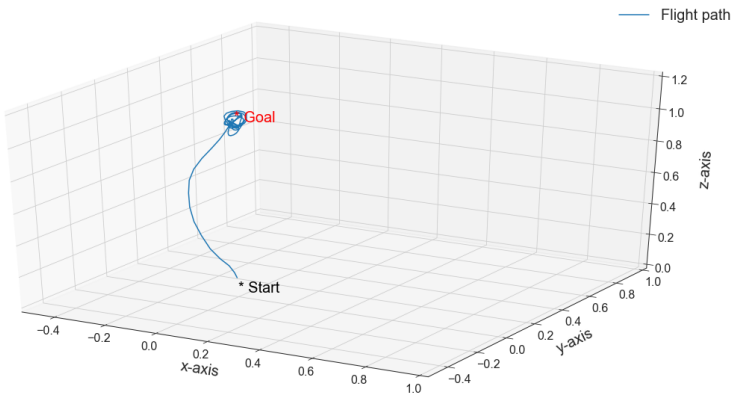


FIGURE 5.16: 3D representation of the hexacopter's performance using concept-based mission controller.

5.5 Conclusion

In this chapter, we set out to tackle the problem of designing intelligent controllers for flying robots. However, before dealing with such a challenging problem, we first investigate the nuances and potentials of reinforcement learning methods for learning different control behaviours of UAVs. We have found out that reward function is the fundamental component in reinforcement learning, and poorly defined rewards may result in unexcepted behaviours. Thus, we have presented in great details the reward function designed in this work. To validate the effectiveness of the approach, two experiments have been conducted in the RotorS framework. In the first scenario, a hexacopter UAV, with 6-DoF, is trained to perform a stable hovering task in continuous action-state spaces. The second scenario extends the first model by means of learning to navigate towards the given goal position, where the agents learn to continue hovering for a defined number of timesteps. To draw a clearer view of the whole course of training and present how the performance of the agent has evolved within episodes, we have randomly selected three different models for each scenario. Finally, we have shown that the hexacopter UAV achieved both given objectives successfully.

6
Learning of Control Behaviours in Flying Manipulation

In the previous chapter we have demonstrated how machine learning can be used as a versatile tool for engineers when designing control policies for robotic systems with highly nonlinear dynamics. We have shown two different mission scenarios. In the first scenario the agent, a hexacopter UAV, learned to maintain a stable hovering state at a given position. The second scenario extended the first one, where the hexacopter had to learn to perform a takeoff and fly towards the goal position, where it should continue to maintain a stable hovering state for as long as possible.

The performance of the agent for both learning problems have been presented in great details and several video clips have been provided, demonstrating different stages of the learning process. In this chapter we aim to apply the proposed controller for solving more sophisticated mission, such as the control of a flying manipulation as well as demonstrating the modularity and extendibility of the approach. Flying manipulations are known as complex robotic systems due to the additional disturbances that the robotic arm creates during the UAV's flight.

"Deep Reinforcement Learning-based Continuous Control of a Flying Manipulator" **(unpublished)**

6.1 Introduction

With the recent advances in robotic systems, its real-life applications grew tremendously. For instance, Unmanned Aerial Vehicles (UAVs) have already been deployed in many different civilian applications, ranging from military missions to videography and goods delivery [221]. Despite their increased popularity and usage in everyday tasks, these robots are incapable of physically interacting with objects in their environment, thus, categorising them as observer-type of robots.

Recently, a novel robotic system, known as an unmanned aerial manipulation (UAM) or flying manipulation (FM), has become a hot research topic in the field of robotics. They expend the functionalities of UAVs, endowing them with manipulation/interaction capabilities. Some of the state-of-the-art flying manipulations are shown in figure 6.3. Being the combination of two multifunctional robotic platforms, on one side, flying manipulations open the immense potential for solving numerous problems impossible or dangerous to perform by humans or other mobile robots. On the other hand, designing control policies remain a challenging issue, due to their highly nonlinear dynamics and the additional disturbances introduced by the robotic arm during operations. When the arm of an aerial robot moves, the mass distribution changes. Significant variations in the moments of inertia and centre of mass generate additional forces and torques on the flying robot which can rapidly destabilize the platform to a point of no return. These physical changes are further increasing the difficulty of countering destabilisations in free flight. [135].

While most of the state-of-the-art solutions are based on classical control theory approaches [75, 24, 109, 12, 213, 212], many suffer from limitations in terms of model design, nonlinear approximations and/or computation efficiency. Therefore, providing autonomous and stable navigation in an unknown environment remains a ambitious problem. In this context, reinforcement learning techniques have proven to overcome many of these limitations. During the last years, they have shown great potential for learning sophisticated robotic behaviours and have been extensively deployed in different robotic applications ranging from autonomous navigation of UAVs to motion control of underactuated manipulations [174, 156, 145, 87, 115]. Moreover, in the previous chapter, we have shown that a hexacopter UAV was able to successfully learn the given tasks in a reasonable amount of training time. Thus, in this chapter, we aim to increase the complexity of the learning problem and apply deep reinforcement learning for controlling a flying manipulation in a continuous state-action setting. The learning process starts with a blank controller where a flying manipulation is trained to perform a task composed of a takeoff, navigation towards a given goal position and positioning its end-effector at a predefined target point. During the entire performance, the flying manipulation should maintain and perform stable movements, thus, to handle the disturbances created by the onboard robotic manipulation, the flying robot is allowed to correct its position and actuation angle throughout the whole process.

For achieving such behaviour, the controller requires the position of the end-effector and the orientation of the flying manipulation as input and outputs the actuation angle along with the six rotor velocities. The fulfilment of this task is a precondition for further implementation of other functionalities for flying manipulations with more sophisticated kinematics and dynamics models. As it was stated before, in a reinforcement learning setting, the agent discovers an optimal control behaviour by interacting with its environment through trials and errors, which may often result in unexpected crashes during the early stages of the learning process. Thus, for the training of the flying manipulation, the RotorS Gym experimental framework,

proposed in this work, has been used as a starting point. Moreover, we extend the RotorS Gym framework to our new Cyber Gym Robotics modular platform by expanding its functionalities. The Cyber Gym Robotics aims to provide not only close-to-real world settings but also allow control engineers to seamlessly switch between the learning agents and algorithms. Additionally, we developed and implemented the ability to create sophisticated mission controllers via the configuration of concepts. Hence, for solving the learning control behaviour in flying manipulations described above, we have extended the hexacopter UAV used in chapter 5, by adding a robotic manipulation with 1-DoF, as shown in figure 6.3a. Both, the training and further experiments have been performed in the Cyber Gym Robotics platform using a concept-based mission controller presented in chapter 4. Later, to increase the complexity of the learning problem, we upgrade the robotic manipulation from a 1-DoF to 2-DoF configuration, resulting in a flying manipulation with 8-DoF. We demonstrate that CGR along with a concept-based mission controller is both modular and extensible for learning various tasks of increasing complexity.

To obtain the desired objectives, the Proximal Policy Optimization has been used as a learning algorithm. Although TRPO has achieved successful and consistent high performance when training a hexacopter UAV, demonstrated in chapter 5, it often suffers from sample inefficiency as well as computational and implementational complexities. As it was stated before, policy gradient methods for estimating the right step size for updating policies remains one of the most challenging problems, since an inappropriate step size may result in severe policy degradation due to the fact that the input data strongly depends on the current policy [77]. To solve this issue, TRPO imposes a trust region constraint onto the objective function in a way to control the KL divergence between the new and old policies [176]. While this optimizes the policy within the trust region monotonically improving its performance, due to the need of computing several second-order matrixes, it makes the algorithm computationally inefficient and difficult to scale up for larger network architectures. Thus, to improve the learning procedure, in this chapter, the Proximal Policy Optimization algorithm has been used. PPO has been proposed by Schulman et al. [175], which adopts all the benefits of TRPO algorithm, at the same time, significantly reducing its complexity by adopting a clipping mechanism. Unlike TRPO, PPO uses a first-order optimizer like the Gradient Descent method to optimize the objective, completely avoiding imposing the hard constraints.

To the best of our knowledge, to this date (21.07.2020), there is no other research focusing on designing intelligent control policies for flying manipulations using deep reinforcement learning techniques. However, through numerous investigations, we found out the following works that attempt to solve relatively similar problems. For instance, Figueroa et al. [45] propose a reinforcement learning approach for balancing an inverted pendulum attached on a UAV. The authors split the main task into two subtasks, referring as (1) initial balancing and (2) balance and hover. In the first phase, the pendulum learns to stay closer to the upright position, and in the second phase, the UAV learns to hover while maintaining an inverted pendulum at the same place. Their experiments were performed in a simulated environment and have shown the UAV learning to reduce its velocity rapidly to zero for maintaining the pendulum at the desired position. Another study, presented by Palunko et al. [144], addresses the task of suspended load manipulation attached on an aerial robot. Using the least squares policy iteration (LSPI) reinforcement learning algorithm [100] the agent learns to follow different trajectories while having a suspend load attached on its center of mass. Although these works use reinforcement learning for solving a balancing task of a UAV with an external disturbance,

there is no other work found in the literature aiming to solve a control problem of a flying/aerial manipulation.

6.2 Kinematics and Dynamics of a Flying Manipulation

As a learning agent the digital replica of the AscTec Firefly, presented in chapter 5, has been extended to a 7-DoF flying manipulation, shown in figure 6.2. Below we describe the kinematics and dynamics models of a flying manipulation.

Kinematics

Similar to the hexacopter's model, described in section 5.2, flying manipulation also is consist of a world-fixed inertial frame, F_I, a body frame F_B, fixed to the center of mass of the flying manipulator, and F_E, a frame attached to the end-effector of the manipulator.

The position of F_B is given by $p_B = [x, y, z]$ and the orientation with respect to F_I is given by the rotation matrix R_B (5.1). Finally, the position of F_E with respect to F_I is given by [8, 82]

$$p_E = p_B + p_{EB}^B R \tag{6.1}$$

where p_{EB}^B gives the position of F_E with respect to F_B.

The orientation of F_E is described by the rotation matrix

$$R_E = R_B R_E^B \tag{6.2}$$

where R_E^B describes the orientation of F_E with respect to F_B [79].

Dynamics

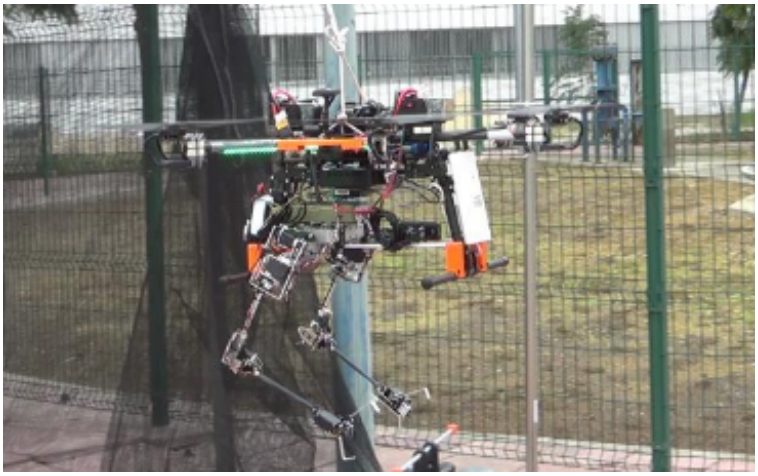
Let v and Ω be the linear and angular velocities of F_B. The model used to describe the dynamics of the flying manipulator is based on [81, 79]:

$$\begin{bmatrix} \mathbf{m} & 0 \\ 0 & \mathbf{I} \end{bmatrix} \begin{bmatrix} \dot{v} \\ \dot{\Omega} \end{bmatrix} + \begin{bmatrix} 0 \\ \Omega \times \mathbf{I}\Omega \end{bmatrix} = \begin{bmatrix} \mathbf{G} + T \\ \mathbf{Q} + M \end{bmatrix} \tag{6.3}$$

where $\mathbf{G} = \mathbf{m}gR^T e_3$ with $e_3 = [0\ 0\ 1]^T$. $T = \sum_i t_i, \mathbf{Q} = \sum_i q_i$ and $M = \sum_i m_i$, where t_i, q_i and m_i are the rotor thrust, torque and moment, respectively, for all $i \in \{N, S, E, W\}$. The flying manipulator is actuated by the variation of forces and moments which can be obtained as a variation of rotor speeds [81]. It has in total six rotors of different directions, presented on figure 6.2. The angular velocity of rotor i is denoted by $\omega_i, (i = 1, 2, \ldots, 6)$. Thrust and torque are related to the rotor velocity of the rotors as follow:

$$\begin{bmatrix} F_t \\ \tau \end{bmatrix} = \begin{bmatrix} c_T & c_T & c_T & c_T & c_T & c_T \\ -dc_T & dc_T & \frac{1}{2}dc_T & -\frac{1}{2}dc_T & -\frac{1}{2}dc_T & \frac{1}{2}dc_T \\ 0 & 0 & \frac{\sqrt{3}}{2}dc_T & -\frac{\sqrt{3}}{2}dc_T & \frac{\sqrt{3}}{2}dc_T & -\frac{\sqrt{3}}{2}dc_T \\ -c_\tau & c_\tau & -c_\tau & c_\tau & c_\tau & -c_\tau \end{bmatrix} \begin{bmatrix} {\omega_1}^2 \\ {\omega_2}^2 \\ {\omega_3}^2 \\ {\omega_4}^2 \\ {\omega_5}^2 \\ {\omega_6}^2 \end{bmatrix} = \mathrm{M}\overline{\omega} \tag{6.4}$$

(A) QARM1 quadrotor with 3-link manipulator arm, developed by University of Seville and CATEC [76].



(B) Aerial manipulation system consisting of the anthropomorphic compliant dual arm manipulator integrated in a hexarotor platform [189].

(C) Unmanned Flettner helicopter with KUKA LW [101].

FIGURE 6.1: Examples of different flying manipulations.

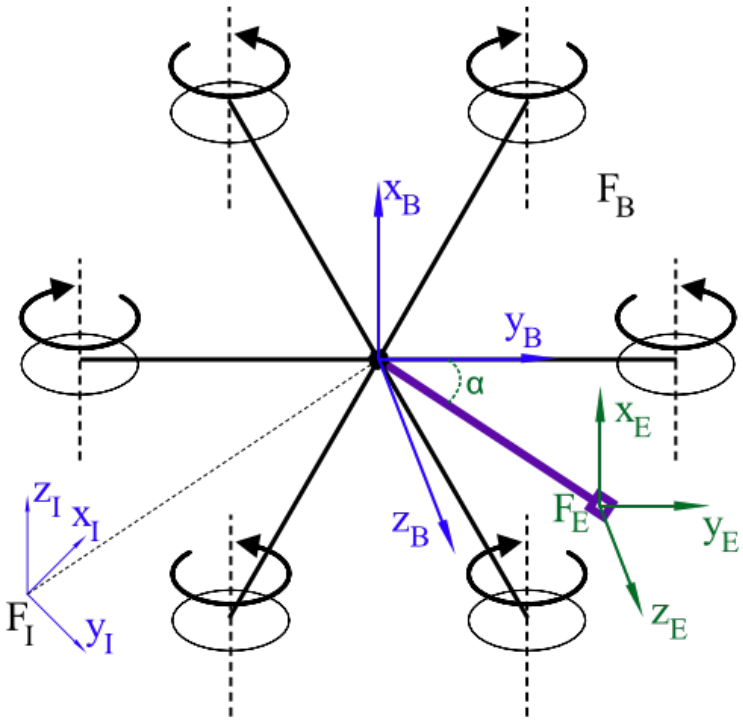


FIGURE 6.2: The schematic representation of the flying manipulator with $1 - DoF$.

where F_t and τ are the thrust and torque acting on the flying manipulator, generated by its rotors, c_T is thrust coefficient, c_τ is torque coefficient, d is the distance from the center of rotor to the center of mass of the system, M is the mixer matrix of the hexacopter, and finally, $\overline{\omega}$ is the vector of square of rotor speed [219]. Since providing a comprehensive theoretical description of the mathematical model of a flying manipulation is not included in the scope of this work, the kinematics and dynamics model of a multirotor system presented above by no means is complete. Hence, for more detailed introduction refer to [81, 79, 82].

6.3 Learning-based Control Algorithm

To obtain the desired behaviour and discover an optimal policy for a defined objective task, the Proximal Policy Optimisation (PPO) algorithm has been used. Similar to TRPO, discussed in chapter 5, PPO is also a policy gradient-based model-free, on-policy algorithm that provides robust learning performance. In contrast to TRPO, which aims to optimize a surrogate function, using KL divergence between the current and previous policies, PPO introduces a clipping mechanism which dramatically reduces the complexity of the implementation of TRPO algorithm. To prevent drastic alterations in the policy, PPO pushes away the old policy if the probability ratio between the current and previous policies is out of the clipping range.

PPO adopts the actor-critic architecture using two different networks as function approximations. A first network, known as the actor, makes decisions, a second network, called the critic evaluates the actions of the actor and updating the network's weights.

6.3.1 Proximal Policy Optimization

Similar to TRPO, Proximal Policy Optimization as well aims to optimize the surrogate objective function. It adopts a clipping surrogate objective function, aiming to simultaneously optimize a stochastic policy π_θ.

The clipping surrogate objective function of a state-action (s_t, a_t) is defined as follows:

$$L_t^{\text{CLIP}} = \mathbb{E}_t(\min(g_t(\theta)A_t, \mathcal{F}^{\text{CLIP}}(g_t(\theta, \theta_{\text{old}}), \epsilon)A_t)), \tag{6.5}$$

where $g_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio, θ and θ_{old} are the parameters of the new and old policies, and A_t is the estimated advantage value. $\mathcal{F}^{\text{CLIP}}$ is the clipping function defined as follows:

$$\mathcal{F}^{\text{CLIP}}(g_t(\theta), \epsilon) = \begin{cases} 1-\epsilon & g_t(\theta) \leqslant 1-\epsilon \\ 1+\epsilon & g_t(\theta) \geq 1+\epsilon \\ g_t(\theta) & otherwise \end{cases} \tag{6.6}$$

The idea behind (6.6) is that $\mathcal{F}^{CLIP}$ clips the probability ratio $g_t(\theta)$ at $1-\epsilon$ or $1+\epsilon$, depending on the advantage value A_t being positive or negative. This removes the incentive for moving the probability ratio $g_t(\theta)$ outside the interval $[1-\epsilon, 1+\epsilon]$. Finally, ϵ is a hyperparameter, $(0 \leqslant \epsilon \leqslant 1)$, used to define the clipping range, in the most cases $\epsilon \approx 0.2$. The final clipped surrogate objective loss can be written as follows:

$$L^{CLIP}(\theta) = \frac{1}{T}\sum_{t=1}^{T} L_t^{CLIP}(\theta) \tag{6.7}$$

A proximal policy optimization (PPO) algorithm that uses fixed-length trajectory segments is shown below. Each iteration, each of N (parallel) actors collect T timesteps of data. Then we construct the surrogate loss on these NT timesteps of data, and optimize it with minibatch SGD, for K epochs.

The algorithm can be descried more practically as follows: given a fixed-length trajectory, at each iteration N actors collect T timesteps of data. Then, the advantages of each T timestps of data is computed.

Finally, the surrogate function is constructed on these NT timesteps of data and for each K epoches an optimization is performed using stochastic gradient descent.

Figure 2 presents the pseudocode of the described algorithm.

Algorithm 2: PPO, Actor-Critic Style

for $i = 1, 2, \ldots$ **do**
  for $i = 1, 2, \ldots, N$ **do**
    run policy $\pi_{\theta_{old}}$ for T timesteps ;
    compute advantage estimates $A_1, \ldots, A_T$
  end
  optimize surrogate objective L, with K epochs and minibatch size $M \leq NT$;
  $\theta_{old} \leftarrow \theta$
end

6.4 Experimental Validation

To demonstrate the effectiveness of the proposed approach, two experiments have been conducted in the Cyber Gym Robotics modular platform. As a learning agent

two custom flying manipulations with 7 and 8-DoF, respectively, has been used, shown in figure 6.3a and 6.3b. The main body of the flying manipulation is composed of a hexacopter UAV, presented in chapter 5, and a custom robotic manipulation that has been attached to the centre of mass of the aerial vehicle. The first learning scenario of the flying manipulation with 7-DoF is composed of a takeoff, navigation towards a given goal position where it should situate its end-effector at a predefined target point. During the entire performance, the flying manipulation should maintain stable movements, thus, to handle the disturbances created by the onboard robotic manipulation, the flying robot is allowed to correct its position and actuation angle throughout the whole process.

The second scenario extends the complexity of the first one by increasing the degrees of freedom of the flying manipulations. For this purpose, one more link has been added on the attached robotic manipulation, which creates additional disturbances during its performance.

Before demonstrating the obtained results, first, we recall briefly the Cyber Gym Robotics multifunctional platform, that offers two different mission controllers: (1) a mission controller that considers the learning task as one objective, (2) concept-based mission controller, especially effective when the learning agent and/or the given task is a combination of more sophisticated actions. The concept-based mission controller allows the engineer to decompose the whole mission into more specific subtask, which dramatically accelerates the learning process as well as the quality of the learned actions.

In the previous chapter the first mission controller has been used for training the learning agent. Thus, to validate the effectiveness of the second mission controller, we used the concept-based mission controller.

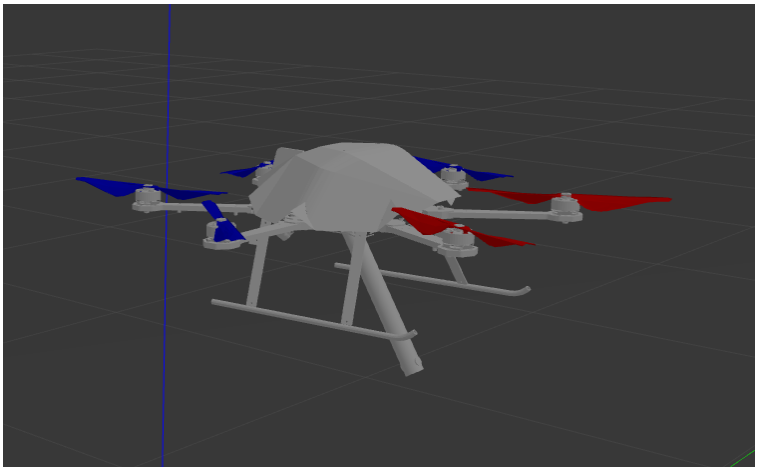
First, we describe the main workflow of Cyber Gym Robotics platform, after we present the formulation of the reward function, along with the architecture of the neural network and the chosen values for the hyperparameters. Finally, we conclude the chapter with the presentation of the results obtained by the learning agents.

6.4.1 Environment Setup

Robotic systems such as UAVs are known as inherently unstable robotic systems with highly nonlinear dynamics. Coupled with robotic manipulations, the complexity of kinematics and dynamics model is being increased, putting an additional strain on the controller. The mass distribution of such special aerial platform is changing continuously, due to the movement of the robotic manipulator, resulting in a positional variation of the rotorcraft UAV's centre of gravity. This is a complex control problem, since the control system must be able to deal with numerous dynamic changes and external disturbances caused by the robotic manipulation in near real-time. To solve such sophisticated control problem, with only limited knowledge about the kinematics and dynamics model of the system we designed a reinforcement learning-based mission controller that learns to adapt to the instabilities created by the manipulator during its operations. For all experiments, Cyber Gym Robotics (CGR) platform has been used. Below we discuss the main workflow of CGR in great detail.

6.4.2 Cyber Gym Robotics Platform

Although RotorS Gym framework is a powerful tool for training rotorcraft UAVs to perform different missions in a safe environment with close-to-real world settings,



(A) Flying manipulation with 7-DoF.

(B) Flying manipulation with 8-DoF.

FIGURE 6.3: The simulated models of the flying manipulation with 7-DoF and 8-DoF.

and we demonstrated that our hexacopter UAV successfully achieved the given objectives, incorporating the previously learned knowledge when designing new missions remains challenging. Herewith, for every new learning problem, the models are being entirely retrained, requiring increasingly longer training as problems grow in complexity. Since our objective is to train a flying manipulation, a robotic system with highly nonlinear dynamics, which continuously varies as the robotic manipulation moves, we extended the RotorS Gym framework to Cyber Gym Robotics platform, which offers an additional mission controller which allows the engineer to decompose the main mission into particular subtasks. Moreover, every subtask is reusable, and can be included in any future missions. It also allows to jointly use different types of control approaches such as reinforcement learning and classic control techniques. Finally, CGR platform allows to seamlessly swap reinforcement learning algorithms as well as targeted learning agents (simulator or real physical robot). We have described the CGR platform in chapter 5 in great detail.

The learning procedure starts with a flying manipulation operating in continuous 3-dimentional states and 7-dimentional action spaces. The hexacopter having six motors, the action space consists of six rotational velocities, $\omega_1, \omega_2, \ldots, \omega_6 \in [550, 650]$ as well as the angle of the manipulator relative to the base of the UAV, $\alpha \in [-\pi/2, \pi/2]$. As a result of the training, the flying manipulation aims to discover an optimal policy for manipulating the rotational velocity of each motor, along with the angle of the robotic arm, which moves alongside the y-axis. Finally, the main workflow of the Cyber Gym Robotics platform, presented on figure 6.4, can be described as follows: at each step the controller receives the position of the end-effector and the orientation of the flying manipulator and outputs seven actions, which, as stated above, consist of the actuation angle along with the rotational velocities of the six motors.

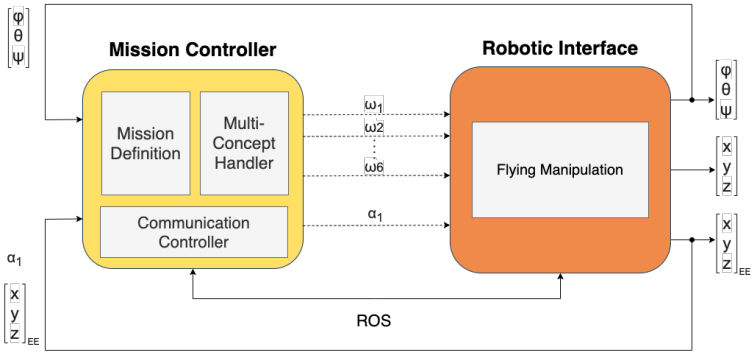


FIGURE 6.4: The main workflow of the Cyber Gym Robotics platform.

To remind, the reinforcement learning algorithm has no knowledge of the actual physical configuration of the hexacopter and learns the best actions which maximises the cumulative reward.

After receiving new actions, the flying manipulation operates in the environment and outputs its position (x, y, z), orientation (ϕ, θ, ψ) and the position of its end-effector $(x, y, z)_{EE}$. As an input, the controller receives the position of the end-effector, $(x, y, z)_{EE}$, along with the orientation, (ϕ, θ, ψ), of the flying manipulation. This process continuous until the environment reaches a terminal state. After, the cumulative reward over the entire episode is computed, a new episode is initialised. When the desired behaviour is achieved the model of the learned subtask is saved in the skills database for further use. It is worth noting that each subtask is trained individually, following the same procedure, thus, also allowing for parallel training.

For instance, to achieve the learning behaviour described above, the entire mission, has been decomposed into the following subtasks:

- Navigation, including takeoff
- Hovering with end-effector positioning

Then, the agent is trained for each subtask, and after the desired performance is achieved, each model is saved in the skills database. Finally, the concept-based mission controller is designed, which switches between each subtask based on predefined conditions. In general, a subtask is being executed if the agent arrives to the state of the corresponding subtask, where it continues to execute its policy until it hits one of its terminal conditions. We have defined two mission-wide terminal conditions: (1) the subtask is being executed until the agent exceeds the validity region, a configurable set of states where the subtask is allowed to run, (2) a predefined threshold of environmental interactions steps. If the agent appears in a state where no subtask is available, the mission is being terminated and the agent returns to its initial state. The graphical representation of the concept-based mission controller is shown in figure 6.5.

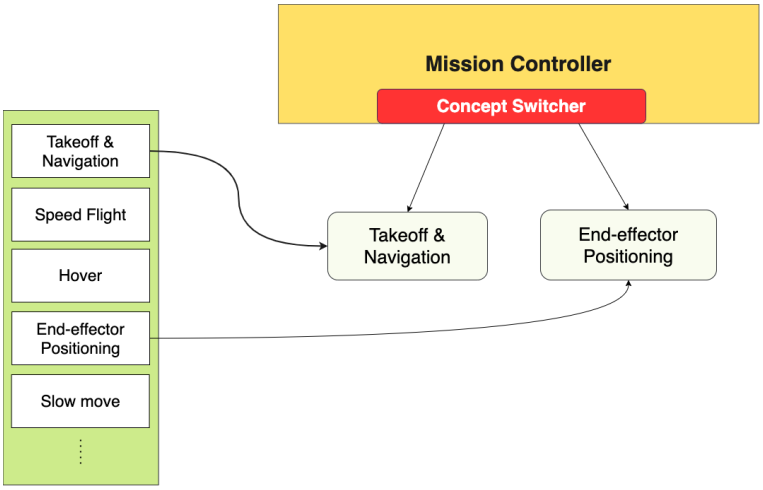


FIGURE 6.5: The general workflow of the concept-based mission controller.

It is worth to mention, that such way of decomposition of missions is not only efficient when designing sophisticated tasks, but also helps to drastically reduces unnecessary explorations, while dramatically accelerating the learning procedure. If a subtask is still too complex, it can be further decomposed until the agent successfully learn each individually task.

6.4.3 Learning Agent

To design a flying manipulation, the simulated model of the AscTec Firefly has been extended in the Gazebo simulator. In general, Gazebo is one of the leading robust physical engines, that provides realistic rendering of environments, offers the ability to accurately and efficiently design and simulate populations of robots as well as to develop custom plugins and sensors for robots.

Gazebo allows to model a new robotic system or customize an existing one using the Unified Robot Description Format (URDF) package. It provides many simulation-specific tags for defining the kinematics and dynamics properties of the

robot. The robotic manipulation attached on the body frame of the AscTec Firefly consists of a revolute **joint** and a **link** with a cylinder shape. The properties of the robotic manipulation are as follows: weight $0.2kg$, radius $0.01m$, length $0.1m$. The graphical representation of the flying manipulation is demonstrated in figure 6.3a.

6.4.4 Reward function

As it was stated before, for any reinforcement learning task the reward function plays one of the most fundamental roles, since it describes the objective given to the robot. Thus, for obtaining a successful training, a careful design of the reward function must be performed. A misspecified or poorly engineered reward function may result in unexpected behaviours or inconvenient policies.

The reward function, used for training the two flying manipulations to learn to desired behaviour described above, is constructed on the top of the base formulation given on equation (5.25). We then extend it, by adding an additional reward for the time that the agent spends in a success state. This allows the agent to discover the states that bring him even closer to its objective. Thus, the final reward function can be described as a combination of three subfunctions: (1) indicate the negative terminal states in which the agent receives a negative reward based on its distance between the current and desired states upon exceeding these states, (2) provide continuous gradual information along the way, (3) give extra cumulative points for the amount of time spent in the success state. Since the agent operates in a world with infinite states, the first subfunction is used to reduce the unnecessary explorations using predefined conditions, known as terminal states. The second subfunction is used to guide the agent towards its entire operation. It aims to minimize the distance between the end-effector of the flying manipulation and the given objective state at the same time considering its ϕ, θ angles. Finally, the third subfunction is used when the agent is closer to its objective. This helps to lengthen the time that the agent spends at the desired state and reinforces its chosen actions when behaving correctly.

6.4.5 Network Architecture

Although reward functions are the base when designing reinforcement learning-based controllers, neural networks are yet another important component having direct impact on the resulting policy. As in TRPO, in the Proximal Policy Optimization algorithm the function approximator is represented by two neural networks, adopting the architecture of actor-critic method. The graphical representation of the neural network architecture is shown in figure 6.6. It consists of two different neural networks, one for actor and one for critic, respectively.

The input of the actor network consists of a 7-dimensional vector, representing the position of the end-effector, denoted as $[x, y, z]_{EE}$, the angle of the manipulation, denoted as α, and the orientation of the flying manipulation, denoted as $[\phi, \theta, \psi]$, respectively. As output, it returns a 7-dimensional vector consisting of the following elements: α, the angle of the manipulation, and $\omega_1, \omega_2, \ldots, \omega_6$, the rotational velocities of the six motors. On the other hand, the critic network evaluates the performance of the actor. As input, it receives the position of the end-effector, the orientation of the flying manipulation and the 7 actions, uses this information to evaluate the performance of the actor and outputs a value, which updates the weights of the network. Both networks have two layers, composed of 64 nodes. The parameters of the networks are initialized randomly and are optimized using the Adam optimizer.

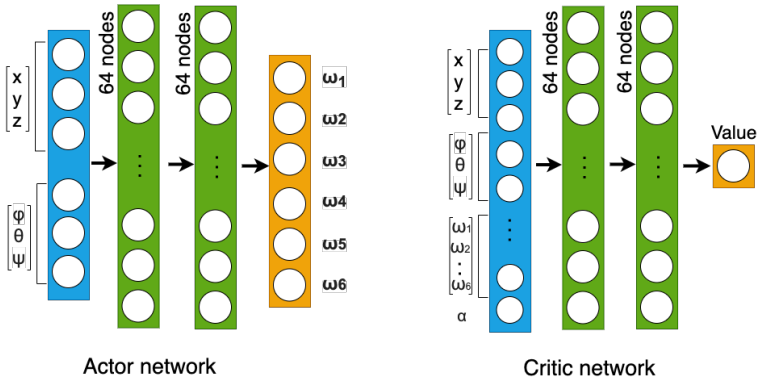


FIGURE 6.6: The architecture of actor and critic networks consisting of two hidden layers, each containing 64 nodes.

6.4.6 Hyperparameters

The tuning of the algorithm's hyperparameters play an important role in reinforcement learning. Hence, to find the best suitable hyperparameters we have performed numerous experiments while varying one specific parameter at a time. Below we describe the role of a few, yet essential hyperparameters.

- ***Discounted factor*** is used to indicate if the agent relies more on immediate or future rewards. A lower value emphasises an immediate reward.
- ***Batch size*** represents the number of experiences used for one iteration of a gradient descent updates.
- ***Number of epochs*** specifies the number of passes through the experience buffer. Bigger number provides more stable updates, but slower learning.
- ***Epsilon*** is a threshold of divergence between the old and new policies used during gradient decent update step. A lower value will result in more stable updates but slower training process.

  The full list of hyperparameters along with their values are given in table 6.1.

TABLE 6.1: Values of Hyperparameters

Hyperparameter	**Value**
Discount (γ)	0.9
Learning Rate	0.00025
ε	0.2
Batch Size	32
Number of workers	1
Timestep per episode	2048

Although, these values of hyperparameters provided relatively fast convergence, further tuning may improve performance.

6.5 Experimental Results

Expanding on our experiments in chapter 5, we evaluate the results and performance of the flying manipulation. Even though the learning problem remain unchanged, this task increased in complexity for the learning agent as 1-DoF has been introduced in the form of a robotic arm attached to the bottom part of the hexacopter. This additional manipulable payload changes the dynamic of the hexacopter, by not only adding weight but also by creating disturbances continuously.

With the increase in difficulty, the agent as been trained using the concept-based mission controller implemented in the CGR platform. Thus, we decompose the main task into two subtasks to be learned individually. One subtask consist of taking off and navigating to a goal position, whereas the second subtask focuses on hovering and end-effector positioning. As the second task is harder to learn due to the focus on the manipulation of the robotic arm, we focus the presented results on the described second subtask. For that, we have performed two experiments using flying manipulations of 7-DoF and 8-DoF, respectively, to push further the difficulty and limits of the platform. Upon learning each subtask successfully, we configure CGR to use the best models for each task and let the agent perform the entire mission in a continuous test environment, first as a 7-DoF flying manipulation, then again with a 8-DoF flying manipulation, where the robotic arm is made out of 2 links and joints, creating more disturbances and challenges for the agent to handle.

Similar to previous experiment, success is defined when the error rate of d is less than or equal $0.15m$, whereas the error rate for ϕ and θ is set to less than or equal $0.3rad$. In addition, those conditions must be met for more than 70 consecutive environmental interactions for the task to be considered learned. On the opposite, the performance is considered failed if the agent is not able to reach the success conditions at least for 10 consecutive environmental interactions after 10000 episodes, which is equal to 5 million environment interactions.

The environmental constraints, parameters as well as hardware used are identical to the ones described in chapter 5.

6.5.1 Flying Manipulation with 7-DoF

Before we show the full performance of the agent, first, we analyse the learning process of the end-effector positioning subtask. As before, we have randomly selected three different models of obtained policy representing the learning process from early to late stages.

The simulated model of the flying manipulation used during the training is shown on figure 6.3a.

Early stage of the learning process

As in previous experiments, the early stage of the learning process prioritizes exploration over exploitation. To achieve this, the agent maximizes the cumulative reward by taking random actions and, thus finding a good policy. In this section, we analyse the agent's learned behaviour after about 100 training episodes, which is considered as early stage. Figure A.19a presents the graphical representation of the agent's positions in three dimensional space. At this stage and as in previous learning examples the agent fails at choosing appropriate velocities and crashes quickly.

By going into the details of the performance this early in the learning process, we can not identify any pattern that would suggest that the robotic arm affects the

agent in any way. Figures 6.7d, 6.7g and 6.7a show the graphical representation of the flight path of the end-effector on x, y, z axis and figures A.15a, A.15b and A.15c represent ϕ, θ, ψ angles of the agent. The mentioned figures clearly show the crash after just 7 environmental interaction steps. Lastly, figure 6.9a show the random change in the robotic arm's angle as the agent did not learn any behaviour related to it.

Middle stage of the learning process

Continuing the learning process, we then analyse the agent's performance after 4000 training episodes, which equals to roughly 2 million environmental interaction steps. Looking at the three dimensional flight path in figure A.19b, we can observe the agent improved performance compared to the previous stage. Despite being still unstable, the agent learned to steer toward the goal position, but fails as it reaches positional boundaries of the simulation.

Figure 6.9b presents the angle of the robotic arm during each steps of the episode. There, the agent is trying to change the angle of the robotic arm in order to reach to the goal position, however the angle fluctuates between $-0.3rad$ and $-0.7rad$, which in turn, create additional disturbances, hence further destabilizing the agent. The graphical representation of the position of the end-effector and the orientation of the agent is given in figures A.16b, A.16c, A.16a and A.17a, A.17b and A.17c, respectively.

Later stage of the learning process

Last but not least, the agent reaches the late stage of the learning process, which translates to about 7000 training episodes or 3.5 million environmental interactions. As we considered the agent to start adopting the correct behaviour during the middle stage, the late stage shows that behaviour reinforced, with the agent able to hover in a stable manner, countering the disturbances of the robotic arm and positioning its end-effector to the goal position for the length if one episode, which is 100 environmental interactions. To validate the behaviour, figures A.18b, A.18c, A.18a and 6.8c, 6.8f show acceptable distance errors and rotational errors of about $0.1rad$.

Following our environmental setup, the model has been tested in a continuous environment and is considered to have been successfully learned. It is worth noting that the agent's robotic arm angle still shows similar fluctuations in figure 6.9c, but also seem to stabilize toward the end of the episode. The explanation behind these values is that the agent learned to use the robotic arm's freedom and resulting disturbances to achieve a stable hovering state.

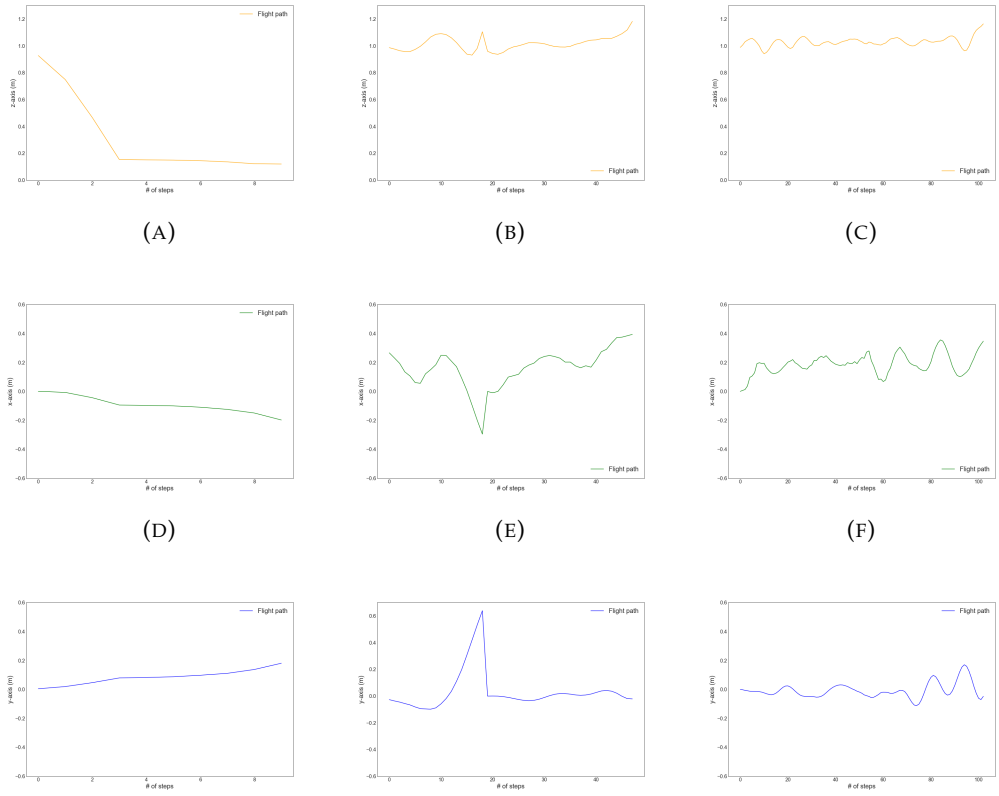
As for other experiments, video clips highlight the performance of the agent during the different learning stages and can be viewed here [120].

Moreover, figure 6.9 shows that the angle was moving along y-axis during entire performance, which was creating an additional challenge for the agent. However, since while the agnet was learning to minimize the distance between the end-effector and the goal position, it was as well learning the best angle of the arm that help to achieve the given objective. Thus, we can see that in the beginning of the learning process the angle of the arm has larger values shown on 6.9a, then in the middle of the performance and at the end of the training shown on figure 6.9b and 6.9c, respectively.

Finally, figure 6.11 presents the average accumulated reward during the training of 10000 episodes. We can see that the cumulative reward per episode grows

larger and larger after about 1000 episodes, which is equivalent to 500000 million environmental interactions.

Lastly, figures A.20 and A.20b demonstrate the graphical representation of the full performance of the flying manipulation with 7-DoF using concept-based mission controller.

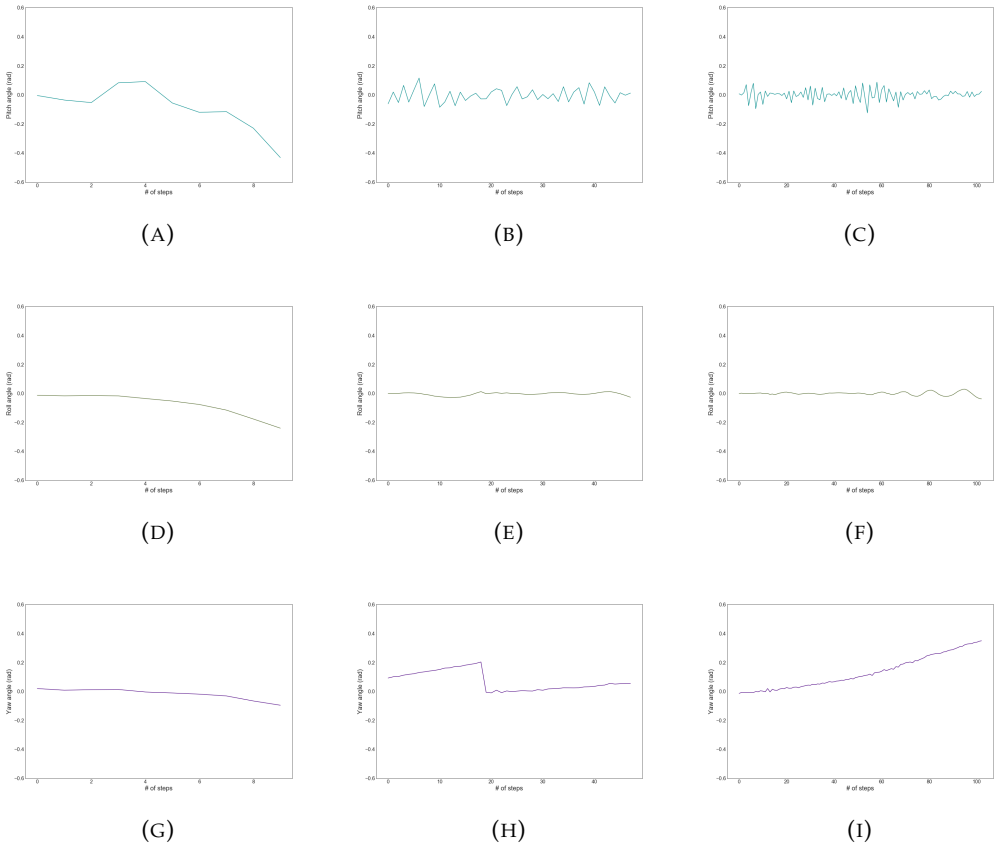


(A) (B) (C)

(D) (E) (F)

(G) (H) (I)

FIGURE 6.7: The graphical representation the agent's performance on x, y and z axis in early, middle and later stages of the learning process.

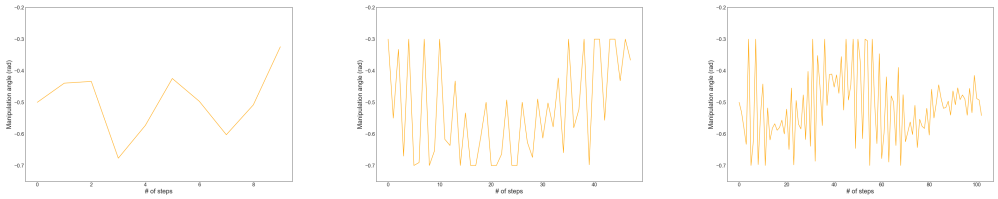


(A) (B) (C)

(D) (E) (F)

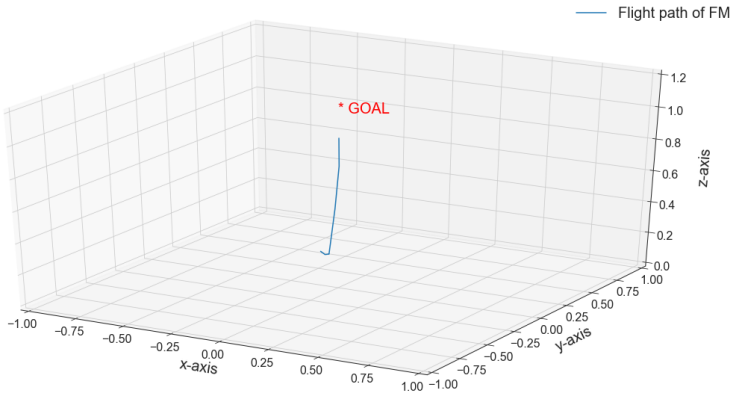
(G) (H) (I)

FIGURE 6.8: The graphical representation of the agent's ϕ, θ and ψ angles in early, middle and later stages of the learning process.

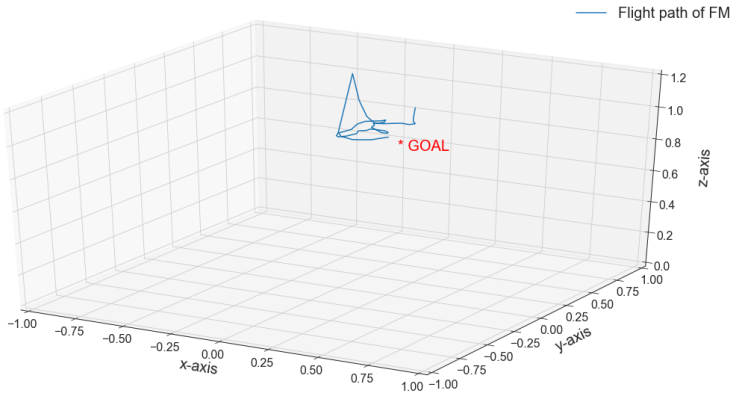


(A) (B) (C)

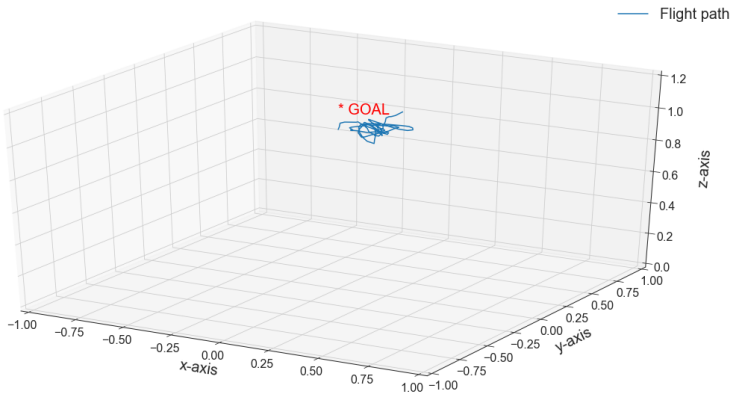
FIGURE 6.9: The graphical representation of manipulation actuation angle in early, middle and later stages of the learning process..



(A)



(B)



(C)

FIGURE 6.10: 3D representation of the performance of the flying manipulation with 7-DoF.

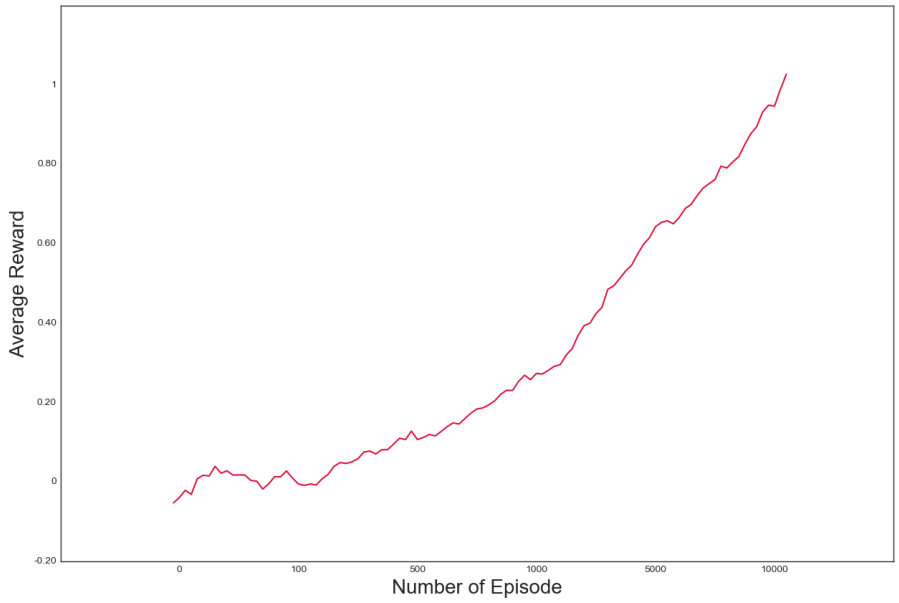


FIGURE 6.11: The representation of the average accumulated reward growth during training for about 22000 episodes.

6.5.2 Flying Manipulation with 8-DoF

To demonstrate the whole course of training and present how the performance of the agent has evolved thousands of training episodes, we have randomly selected three different models of obtained policy representing the learning process from early to late stages.

The simulated model of the flying manipulation used during the training is shown on figure 6.3b.

Early stage of the learning process

The performance of the agent after roughly 100 training episodes is similar to the previous scenario where the agent does not yet seem to achieve and progress toward the desired goal and crashes shortly after the simulation starts. To have the complete analysis, we still present the data of the described event in figures A.21b, A.21c and A.21a related to x, y, z axis, respectively, figures A.22a, A.22b, A.22c, related to ϕ, θ, ψ angles of the flying manipulation. Finally, figure A.29a shows the three dimensional representation of the agent's poor performance.

Middle stage of the learning process

The middle stage, shows as with other learning processes, promising results. As it can be seen in figures A.23b, A.23c and A.23a, the agent manages to achieve partially stable hovering, but demonstrates strong deviations in its flight path, most probably due to the larger and more complex robotic arm, creating even more disturbances and change in dynamics.

Analysing the results in greater detail via figures A.23b, A.23c and A.23a and A.24a, A.24b, A.24c, we can observe that the agent overshoots the target position several times. Having a heavier robotic arm with 2-DoF, it is indeed an expected behaviour as the agent is initially prone to swing in an unstable manner through random actions combined with a start of understanding of steering toward the goal position. This behaviour is also reflected in figures A.27b and A.28b, presenting the constant fluctuations in the angle each join of the robotic arm.

Later stage of the learning process

With an increased complexity of 8-DoF, the agent needed about 10000 training episodes, translating to more than 5 million environmental interactions, for solving the task and thus, demonstrating a great performance as it can been seen in figures A.25b, A.25c and A.25a.

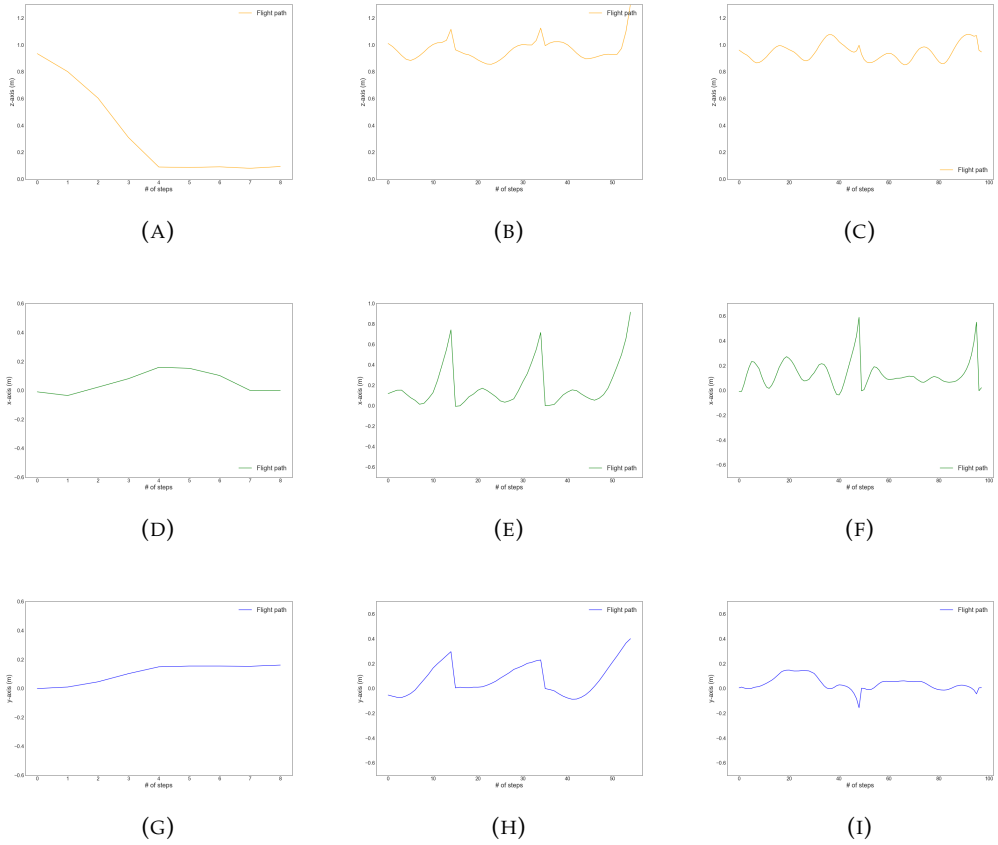
Looking at the positional data in figures A.25b, A.25c and A.25a and orientation data in figure A.26a, A.26b, A.26c, we can validate that a stable hovering state is reached. Lastly, both joint angles presented in A.27c and A.28c show only small fluctuations and a relatively constant behaviour. This shows that the agent has been successful at learning to manipulate a 2-DoF robotic arm during flight.

As with other experiments, the task has been learned and validated through runs in a continuous simulated test environment. Video clips of the learning task is available [121].

Finally, figure 6.16 presents the average accumulated reward during the training of 22000 episodes. We can see that the cumulative reward per episode grows larger and larger after about 6000 episodes, which is equivalent to 2.8 million environmental interactions.

Moreover, figure A.28 shows that the angle was moving along y-axis during entire performance, which was creating an additional challenge for the agent. However, since while the agnet was learning to minimize the distance between the end-effector and the goal position, it was as well learning the best angle of the arm that help to achieve the given objective. Thus, we can see that in the beginning of the learning process the angle of the arm has larger values shown on A.28a, then in the middle of the performance and at the end of the training shown on figure A.28b and A.28c, respectively.

Lastly, figures 6.18a and A.30b demonstrate the graphical representation of the full performance of the flying manipulation with 8-DoF using concept-based mission controller.

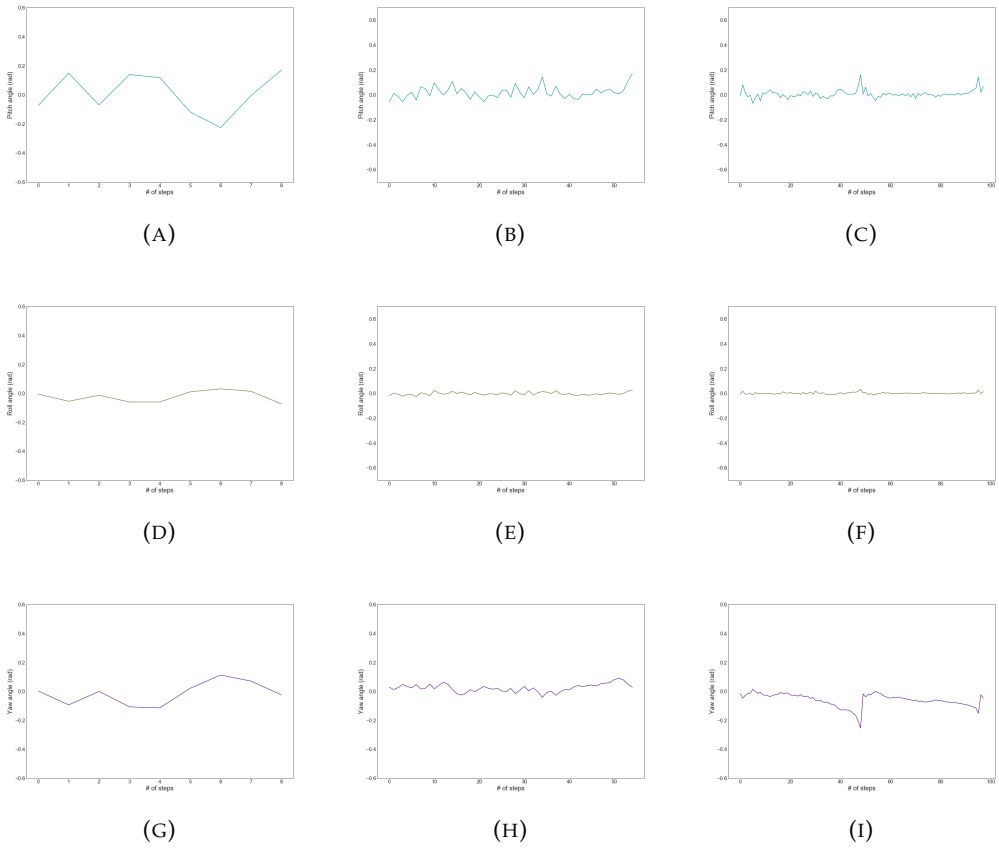


(A) (B) (C)

(D) (E) (F)

(G) (H) (I)

FIGURE 6.12: The graphical representation the agent's performance on x, y and z axis in early, middle and later stages of the learning process.

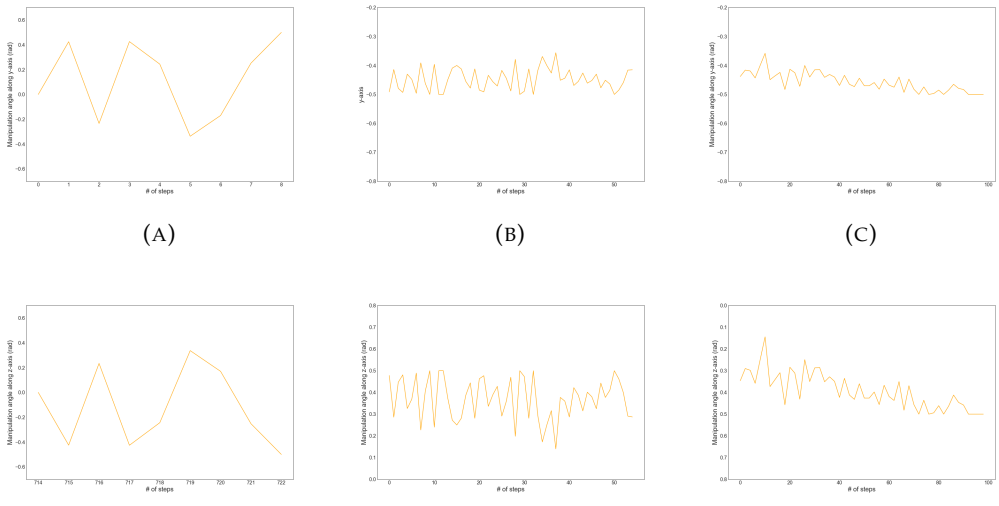


(A) (B) (C)

(D) (E) (F)

(G) (H) (I)

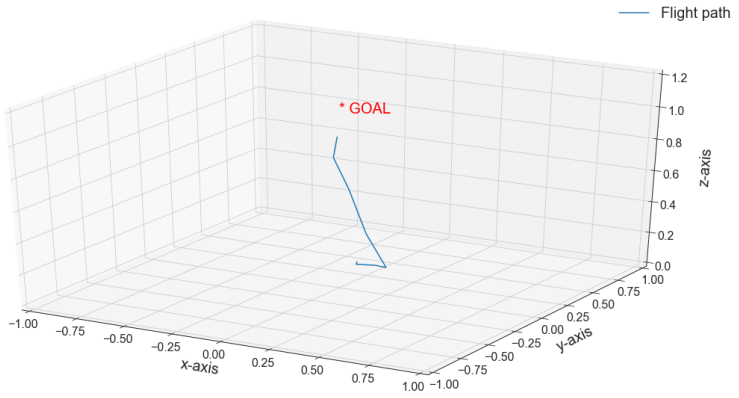
FIGURE 6.13: The graphical representation of the agent's ϕ, θ and ψ angles in early, middle and later stages of the learning process.



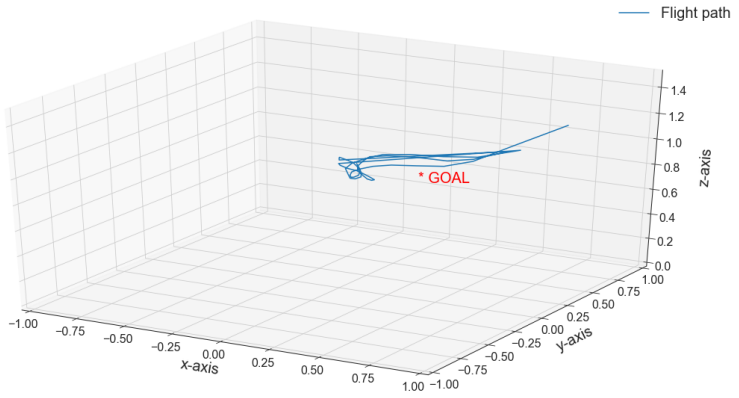
(A) (B) (C)

(D) (E) (F)

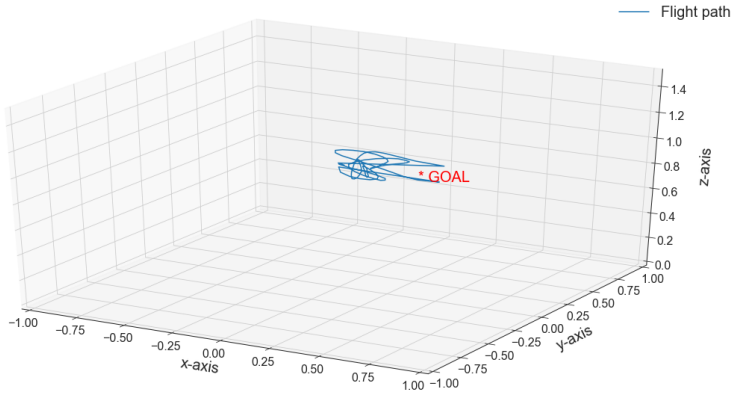
FIGURE 6.14: The graphical representation of the manipulation angles in early, middle and later stages of the learning process.



(A)



(B)



(C)

FIGURE 6.15: 3D representation of the performance of the flying manipulation with 8-DoF.

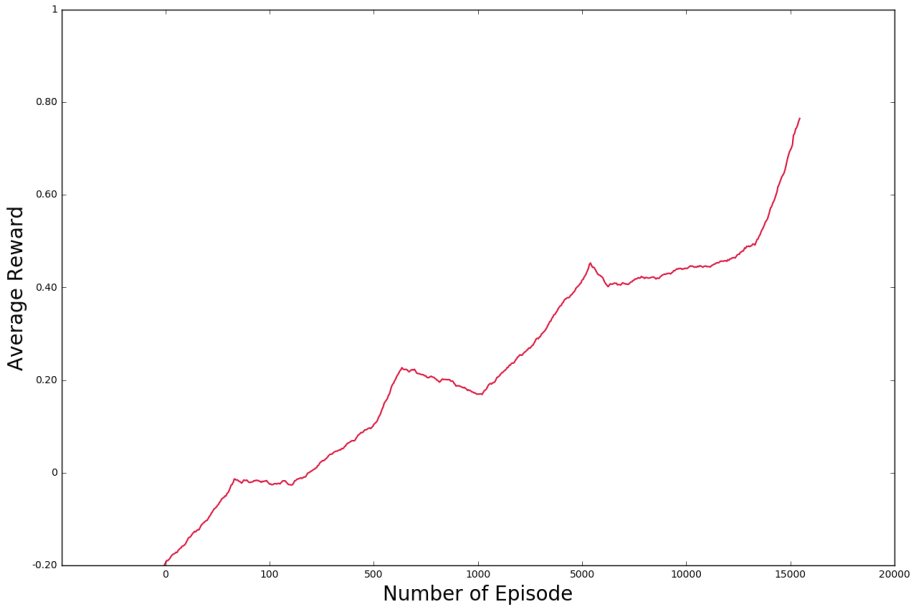
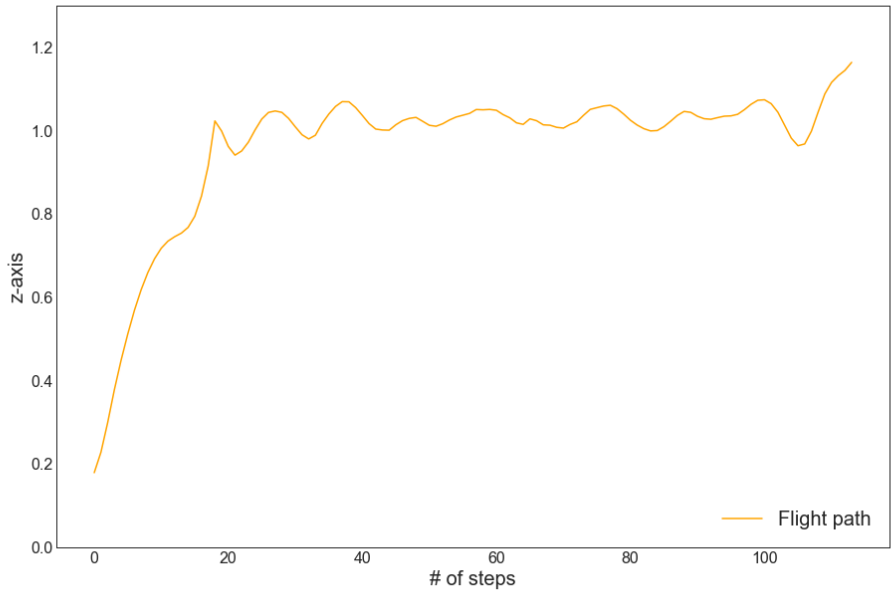
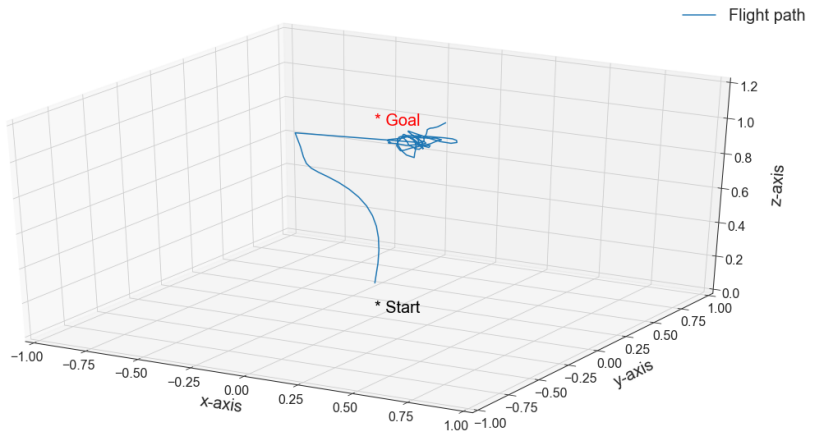


FIGURE 6.16: The representation of the average accumulated reward growth during training for about 15000 episodes.

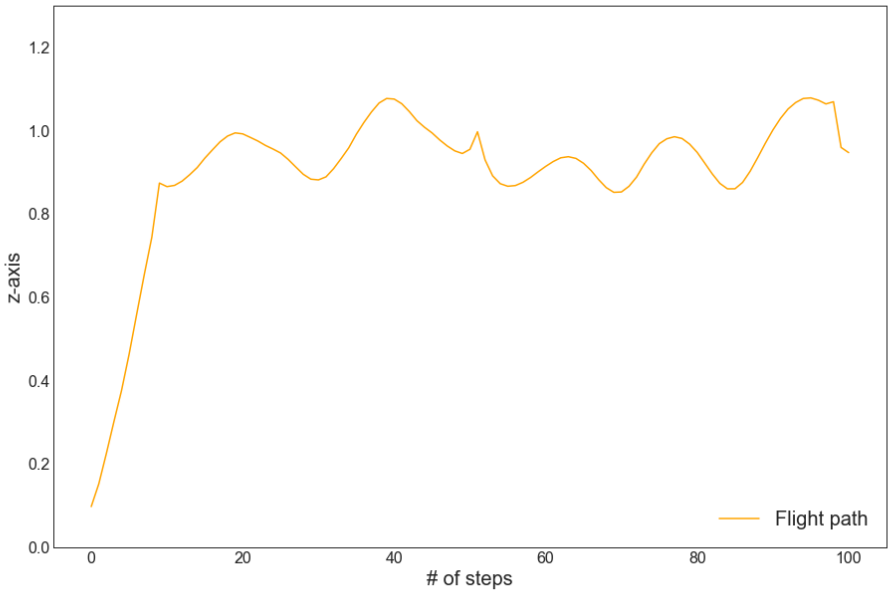


(A)

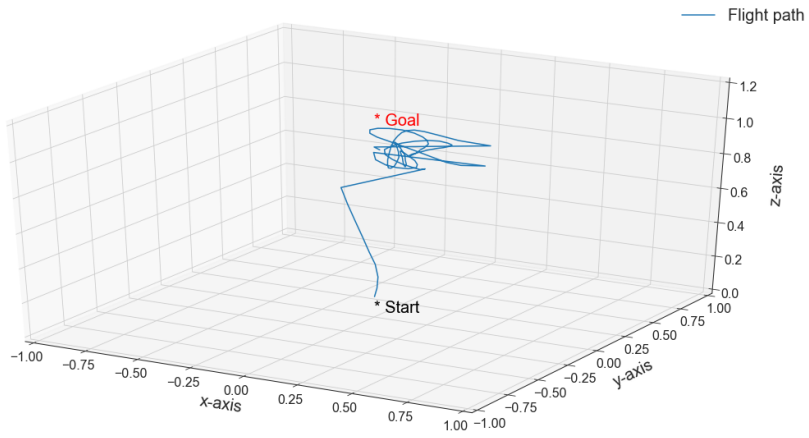


(B)

FIGURE 6.17: The graphical representation of the agent's full performance using the concept-based mission controller.



(A)



(B)

FIGURE 6.18: The graphical representation of the agent's full performance using the concept-based mission controller.

6.6 Conclusion

In this chapter, we tackled the problem of training a hexacopter with the added complexity of a 1-DoF robotic manipulation, resulting in a robotic platform with 7-DoF. We extended our RotorS Gym framework to the new Cyber Gym Robotics modular platform and demonstrated that the agent successfully learned to execute

the mission, consisting of take-off, navigation, hovering and end-effector positioning, within 10000 training episodes, using a concept-based mission controller. The mission has been split into coherent subtasks and the agent has been trained for each task individually. To further demonstrate the possibilities and potential of our approach, we increased the complexity of the task by upgrading the robotic manipulation from 1-DoF to 2-DoF, resulting in a robotic platform with 8-DoF. Cyber Gym Robotics not only proved its ability to successfully make an agent learn complex tasks, but also showed that the developed platform is both, extensible and modular.

7 Conclusions and Future Outlooks

Machine learning is an ever-expanding field of research with a wide range of potential applications. It has been increasingly used in different robotics tasks enhancing their autonomy and intelligent decision-making properties.

This thesis presents how machine learning techniques can enhance the decision-making ability for control tasks in aerial robots as well as amplify the safety, thus broadly improving their autonomy levels. In this chapter, we aim to summarize our research findings by presenting the main contributions and drawing final conclusions of the entire work. Lastly, we discuss the future outlook along with possible extensions of this work.

7.1 Main Findings and Conclusions

Over the past decade, many machine learning-based methods have shown remarkable breakthroughs in different robotic applications, ranging from safety to autonomous navigation and decision-making. Thus, in this work, our main purpose was to apply machine learning techniques on robotic tasks in order to discover and understand if such techniques can enhance control tasks, improve the safety as well as increase their autonomy levels.

As a summarisation of this work, we started by developing an online degradation identification technique for UAV hardware-related components using traditional machine learning methods described in chapter 3. We proposed a lightweight approach that by analysing flight data stream from a UAV following a pre-defined mission, predicting the level of degradation of hardware-related components at early stages. More specifically, that work focused the experiments the propellers of a commercially available quadcopter UAV. To present the effectiveness of the proposed approach we have performed real-world experiments showing that such approach can be used as a safety system during different experiments where the flight path of the UAV is defined a priori, for predicting unnoticeable degradation that often occur during UAV's performances. Although the proposed method can be easily deployed in the autopilot of UAVs, improving its autonomy and reducing the risk of unexpected breakdowns, it has a few limitations. Since the learning algorithm is based on supervised learning techniques, a training dataset should be available. However, we provide the essential steps for gathering and training the data in chapter 3. Lastly, as a proof a concept, a Python based command line tool has been implemented, which provides an interface to analyse the data of UAVs, creates and trains models and finally, simulates prediction of degradations in online setting.

As machine learning showed promising results in chapter 3, we set out to tackle the problem of actually controlling flying robots operating in a continuous action-state setting, having only limited knowledge about their model dynamics. To discover the existing notable successes of machine learning in the field of robotics, we have done an extensive review of the state-of-the-art, and drew out our findings in chapter 2. From this research, it has been clear that Reinforcement Learning (RL), a subclass of machine learning, would fit the problem that we are trying to solve as RL techniques have been successfully applied on different robotic tasks in a continuous action-state setting as well.

The main objective of the thesis was to design intelligent control policies for flying manipulations, but before dealing with such highly nonlinear robotic systems we first investigate the nuances and potentials of reinforcement learning methods for learning different control behaviours of UAVs. As the first learning agent, we have chosen a hexacopter UAV, in other words a flying robotic with 6-DoF, which has to learn to perform stable navigation and hovering actions by directly mapping observations to low-level motor commands. To overcome some of the challenges of reinforcement learning, we have developed the RotorS Gym experimental framework, which provides a safe and close to real-world simulated environment for training multirotor systems.

While working on this learning problem, we have uncovered many important findings essential for successfully designing any reinforcement learning-based control strategy. For instance, the reward function is one of the fundamental components in reinforcement learning. A poorly designed reward function can lead to several problems, such as the derived policy not reflecting the intended outcome or even the need of a tremendous amount of explorations until a desired behaviour is

understood by the agent. Therefore, careful crafting the reward function that adequately represents the desired behaviour is crucial. There exist different ways for designing reward functions. For instance, one of the straightforward ways to design a reward function is to give the agent a reward when it achieves the given goal and a punishment otherwise. However, getting such narrowed feedback from the environment may result in infinite explorations. Thus, such sparse reward functions rarely provide successful learning. Efficient reward functions guide the agent along the way, since they provide continues information about its environment as well as the desired behaviour. This way of defining the reward is known as reward shaping, and we found out that our agents achieved successful performance when the reward function was designed in such a way. We present the detailed description of the reward function that was the base of every other reward function used in this work in chapter 5. Finally, to validate the effectiveness of the proposed reinforcement learning-based controller, two experiments have been conducted. In the first scenario, a hexacopter UAV, is trained to perform a stable hovering task in a continuous action-state space. The second scenario extends the first model by means of learning to navigate towards the given goal position, where the agent learns to continue hovering for a defined number of timesteps. To draw a clearer view of the whole training process and present how the performance of the agent has evolved within the millions of environmental interactions, we have randomly selected three different models, also called policies, each representing the learning process from early to late stages. Finally, we have shown that the hexacopter UAV achieved successfully both given objectives. The obtained results have been discussed in great detail in chapter 5.

Having successfully completed the second learning objective led us to the third scenario, where we increased the complexity of the learning task by adding a flying manipulation, giving the agent a total of 7-DoF and afterwards 8-DoF. As we increased the complexity, we developed the Cyber Gym Robotics platform, which is a major extension of our RotorS Gym framework. Several core functionalities have been added to be able to handle increasingly growing complex learning tasks. One of its main extension is the additional mission controller that allows to decompose complex missions into several subtasks, thus accelerating and facilitating the learning process of a whole mission. Yet another advantage of the Cyber Gym Robotics platform is its modularity which allows to seamlessly switch both, learning algorithms as well as agents. To validate this claims, real-world experiments have been conducted, demonstrating that the model trained in the simulation can be transferred onto a real physical robot with only minor adaptations. The experiments along with the detailed description of Cyber Gym Robotics platform and concept-based mission controller are presented in chapter 4. Finally, the results obtained by the flying manipulation has been demonstrated in chapter 6. We have shown that the agent successfully learned to execute the mission, consisting of take-off, navigation, hovering and end-effector positioning using a concept-based mission controller. The mission has been split into coherent subtasks and the agent has been trained for each task individually. To further demonstrate the possibilities and potential of our approach, we increased the complexity of the task by upgrading the robotic manipulation from 1-DoF to 2-DoF, resulting in a robotic platform with 8-DoF. Cyber Gym Robotics not only proved its ability to successfully make an agent learn complex tasks, but also showed that the developed platform is both, extensible and modular.

Throughout this work and via the various contributions, we also identified a few limitations of reinforcement learning-only based approaches applied on robotic

systems.

As RL agents learn through trials and errors, not only it requires a large number of episodes to converge to an acceptable solution, but the shaping of the actual reward function requires several development iterations as well. As such, RL agents are best trained in a simulated environment, that resembles as close as possible the real target environment. In this work, an appropriate simulation environment existed and could be applied. However, other robotic applications might have the additional challenge of finding or even developing a simulator supporting their mission. Other two limitations, inherent from RL methods, are related to computation requirements. On one hand, training RL algorithms are resources intensive and require dedicated GPUs to run efficiently. Having available on-demand dedicated machines is crucial, as many iterations and days of training are needed for a given mission. It is worth noting that one time-consuming aspect is to wait for the agent to start learning as early stages of the training, typically the first few hours, are chaotic and difficult to evaluate if the agent is on the right way to learn a possible policy. On the other hand, even though a simulation-based environment allows to parallelize the agent's training, it has been identified that different computational resources have different effects on the RL algorithm. The differences mainly come from the underlying software stack, random seeds and hardware configuration, where small differences in CPU or GPU versions can have important impacts on the early learning process and thus the overall performance.

7.2 Future Outlook

In this thesis we have presented that reinforcement learning-based algorithms can indeed be used for designing control strategies for sophisticated aerial robots. However, due to still existing challenges, at this point of time, we believe that reinforcement learning techniques as a sole solution are not suitable for industrial application, yet. As learning the optimal policy through trials and errors in a simulated environment is very time-consuming and does not guarantee that every situation is covered and thus learned. One way to tackle this problem, as some studies proposed, is to use Model Predictive Controller to create an initial trained model and use as a base for fine-tuning the learned policy using RL techniques. This approach can dramatically decrease the training time and provide safety for the physical robot, however both algorithms, and neural networks, must be identical to be able to continue the training.

Given the above statement, we also conclude that RL can show promising applications when combined with other techniques, thus acting as a supporting feature. For instance, as controllers and robotic systems become more and more complex, using the proposed concept-based mission controller provide a structured way to extend robotic platforms. For instance, certain parts of the mission can be managed by classic control methods, while certain experimental tasks can be taken over by a reinforcement learning solution, thus utilizing the best of all worlds in order to solve a given problem, accelerate the development of new applications, while not compromising on safety.

Reinforcement learning being around for more than three decades, recent technological advances has drastically increased the interest in this field. Computational power is still an important constraint when working with this type of machine learning algorithms. Through this work, we have shown that RL can be applied on robotic tasks and handle increase in complexity unlike any other method. The power of RL

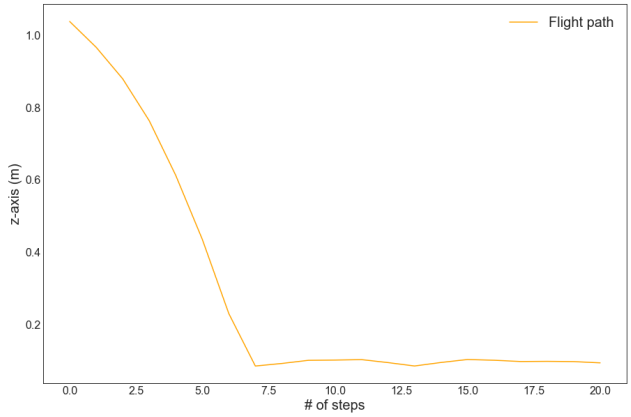
heavily relies on the fact that, if the reward function is appropriately define, the agent is able to learn a behaviour without prior knowledge of it's internal dynamics. Thus, even though RL is still at a relatively young stage, its real-world applications and research opportunities are massive and present numerous open questions and challenges to tackle.

A

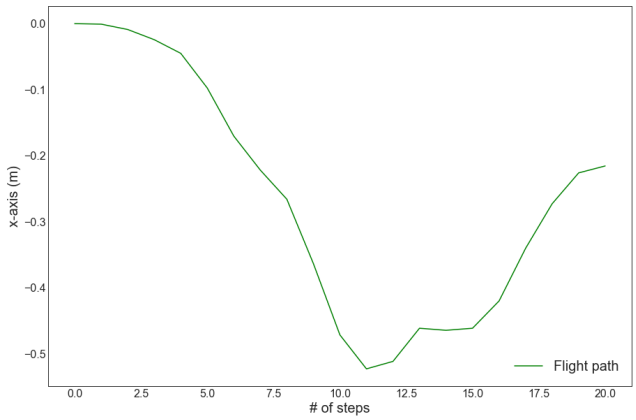
Appendix A

A.1 Hexacopter Hovering Scenario

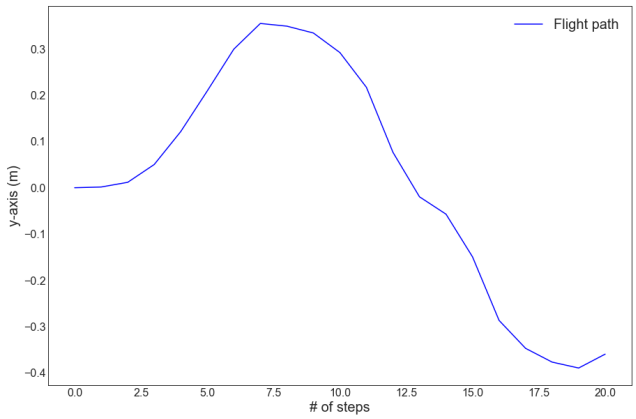
Early stage of the learning process



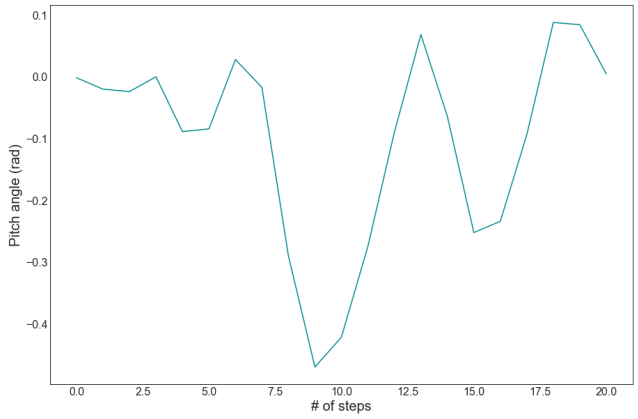
(A)



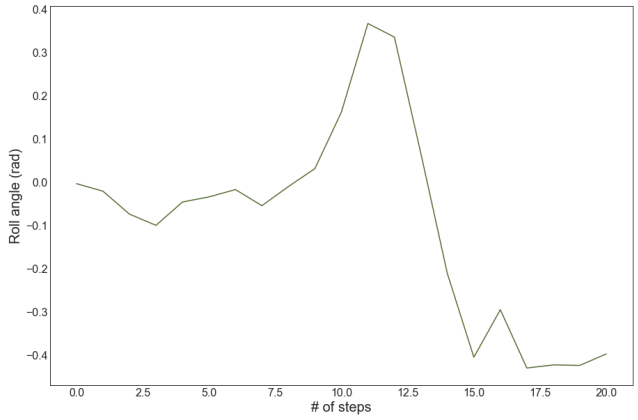
(B)



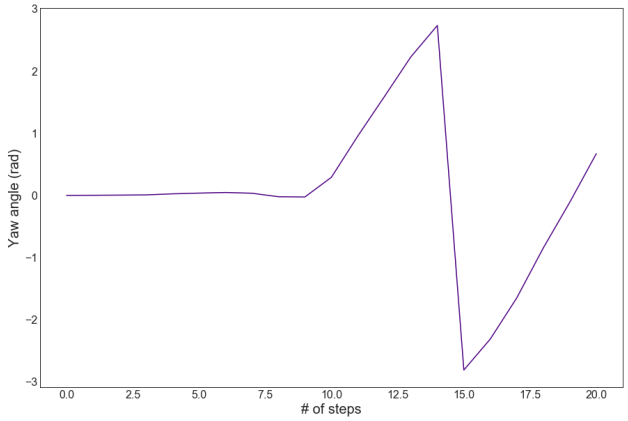
(C)



(A)



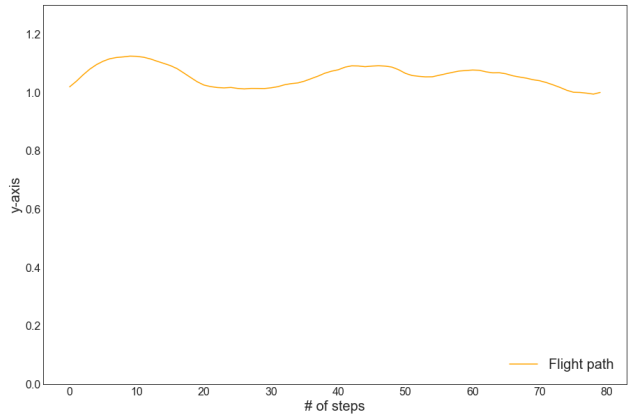
(B)



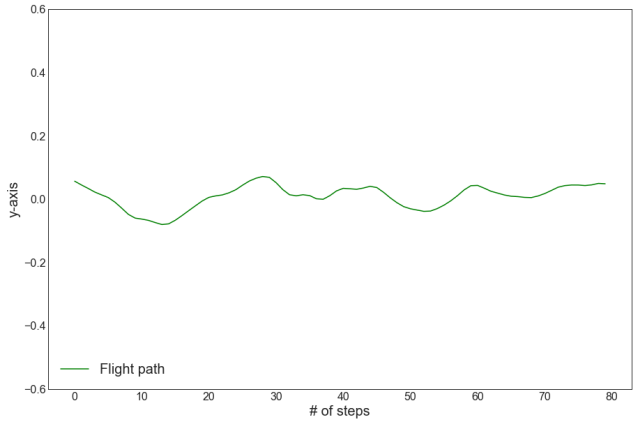
(C)

FIGURE A.2: The graphical representation of ϕ, θ and ψ angles for a single episode in the beginning of the learning phase.

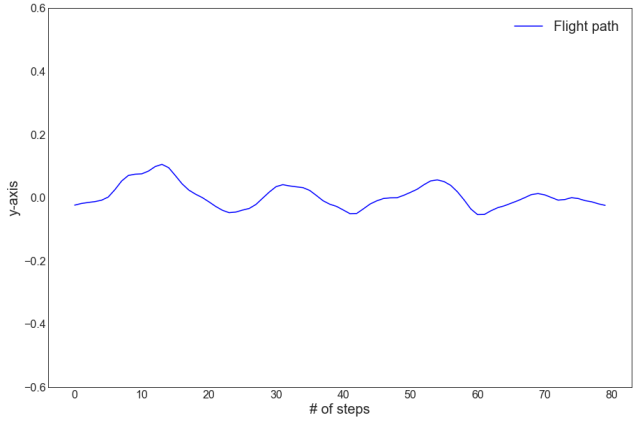
Middle stage of the learning process



(A)

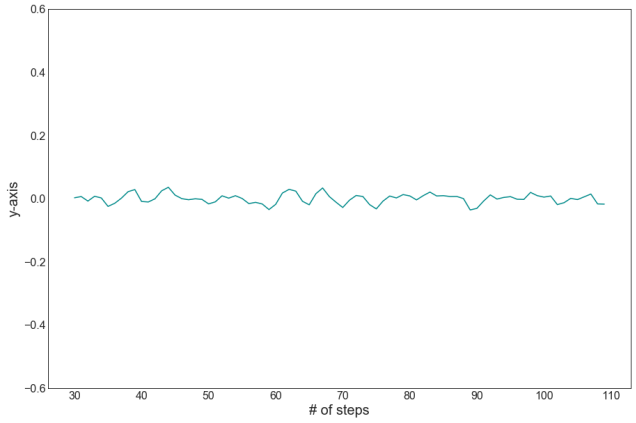


(B)

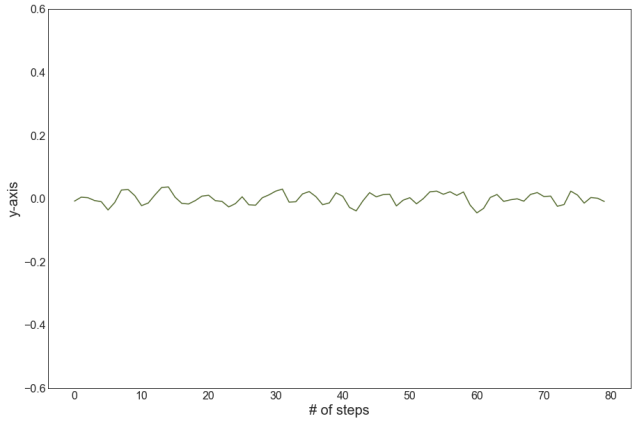


(C)

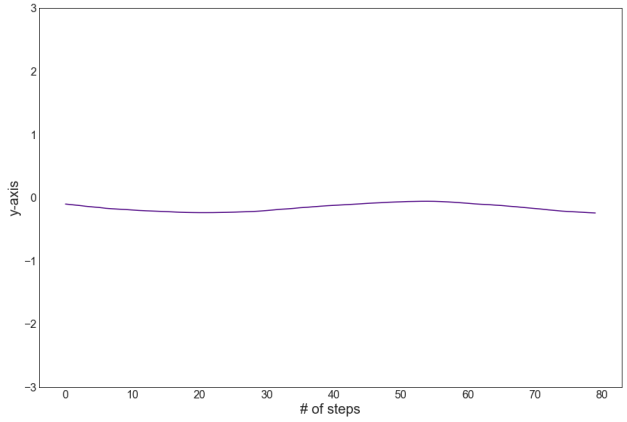
FIGURE A.3: The graphical representation of x, y and z axis for a single episode during the middle stages of the learning phase.



(A)



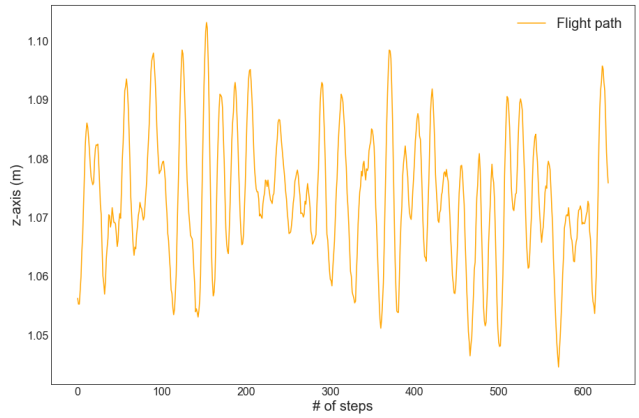
(B)



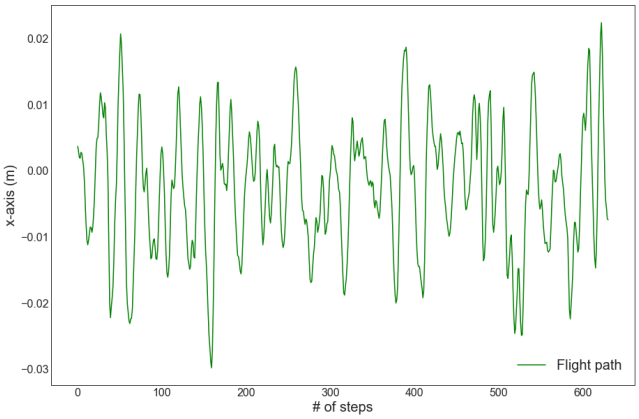
(C)

FIGURE A.4: The graphical representation of ϕ, θ and ψ angles for a single episode during the middle stages of the learning phase.

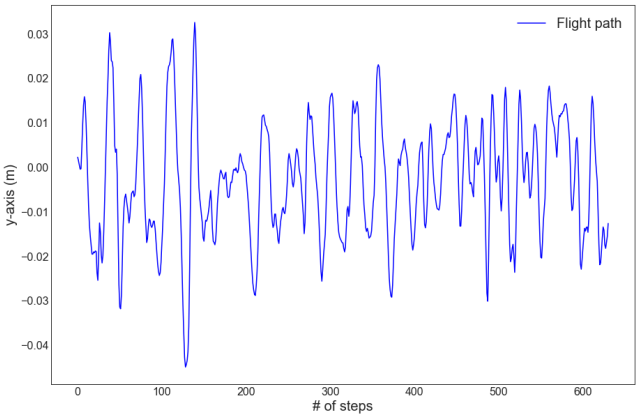
Later stage of the learning process



(A)

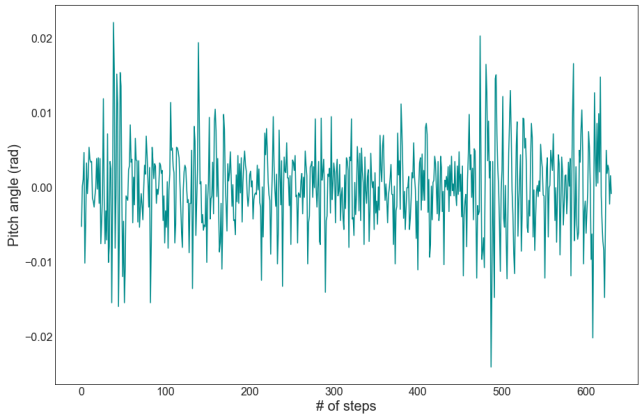


(B)

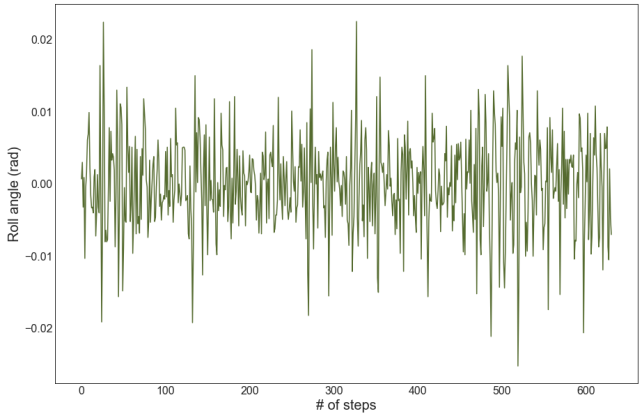


(C)

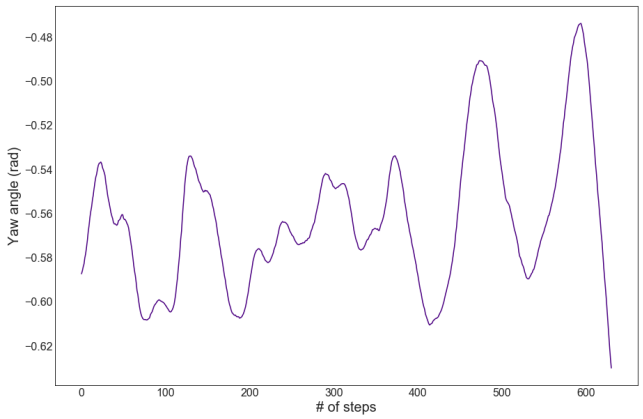
FIGURE A.5: The graphical representation of x, y and z axis for a single episode when the objective has been achieved.



(A)

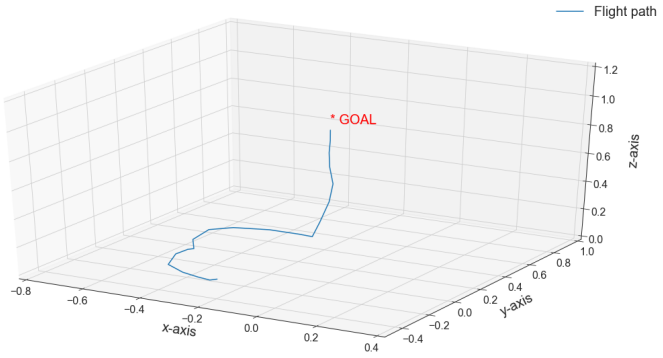


(B)

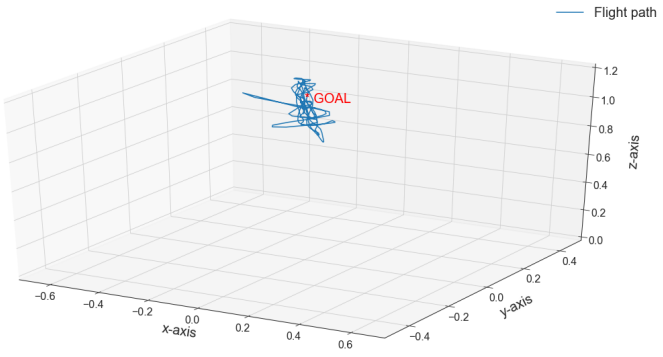


(C)

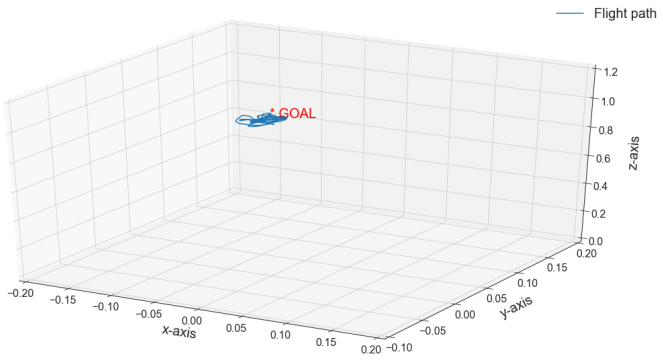
FIGURE A.6: The graphical representation of ϕ, θ and ψ angles for a single episode when the objective has been achieved.



(A)



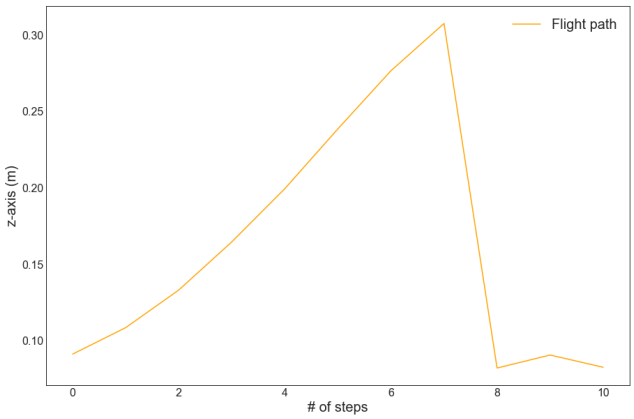
(B)



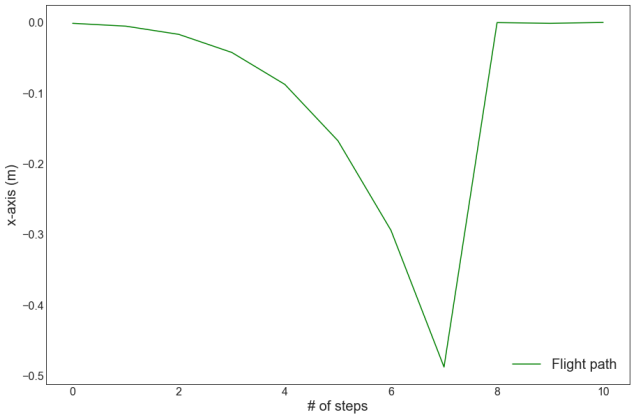
(C)

FIGURE A.7: 3*D* representation of the hexacopter's flight path during different stages of the learning process.

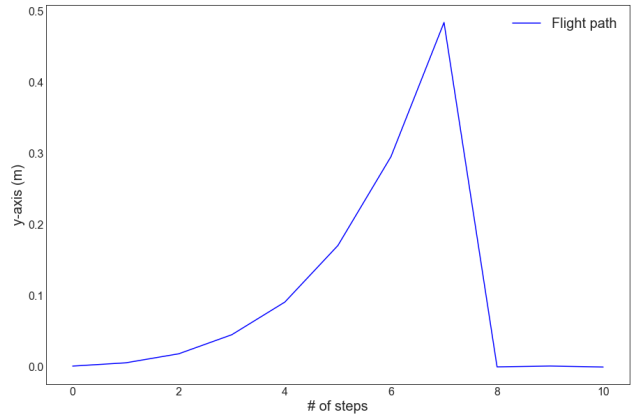
A.2 Hexacopter Navigation Scenario



(A)

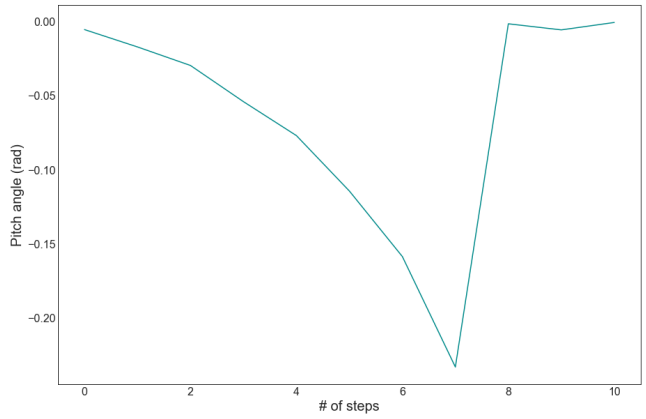


(B)

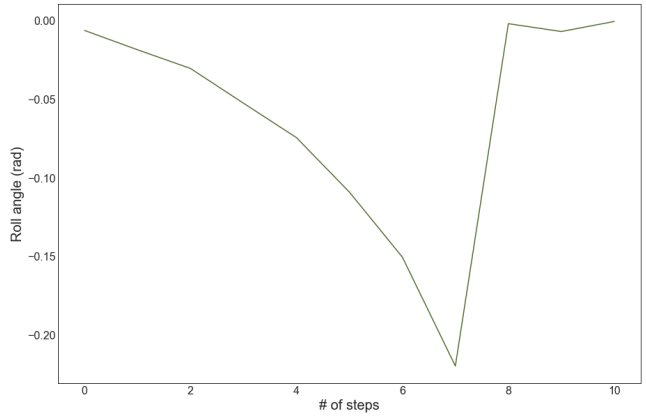


(C)

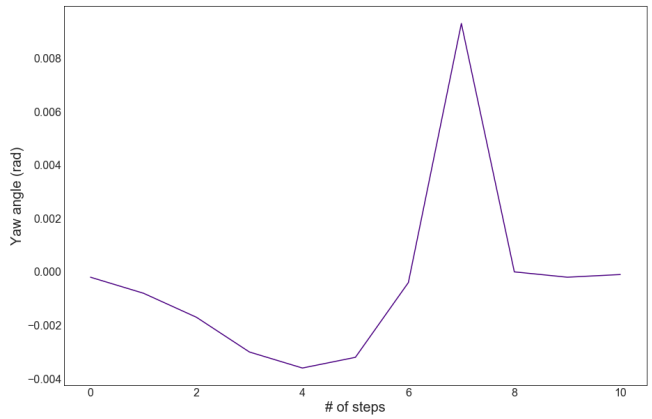
FIGURE A.8: The graphical representation the agent's performance on x, y and z axis in a single episode.



(A)

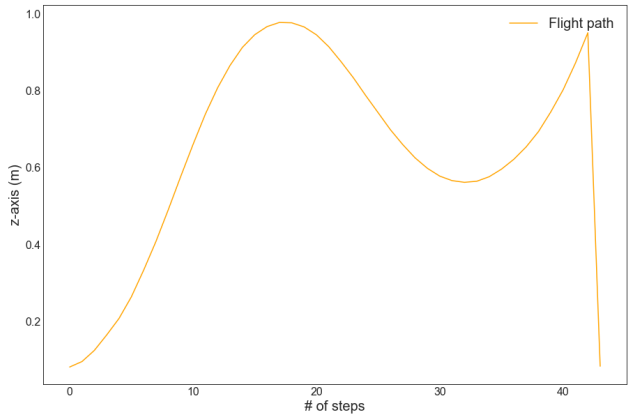


(B)

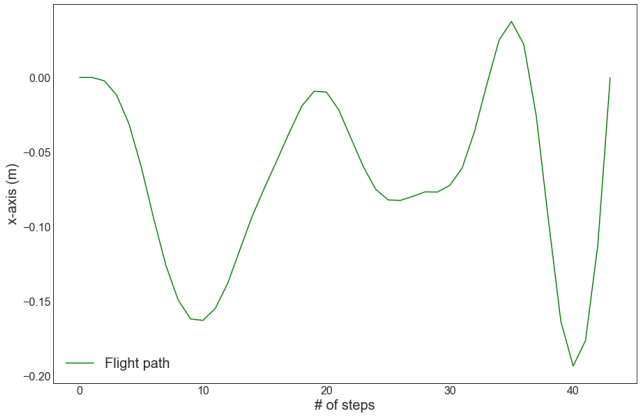


(C)

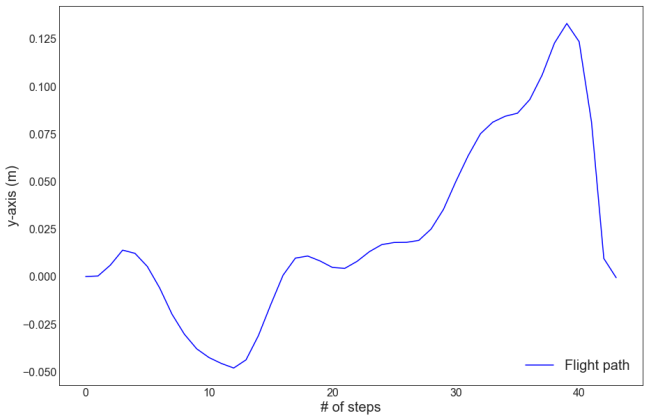
FIGURE A.9: The graphical representation of the agent's ϕ, θ and ψ angles in a single episode.



(A)

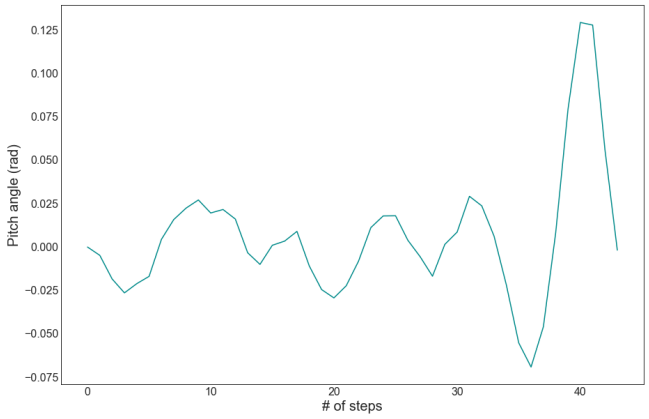


(B)

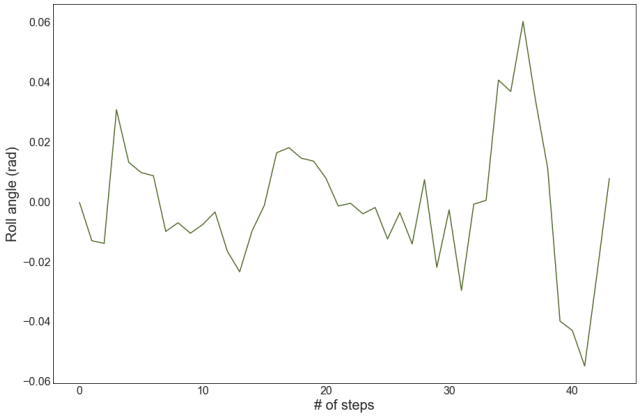


(C)

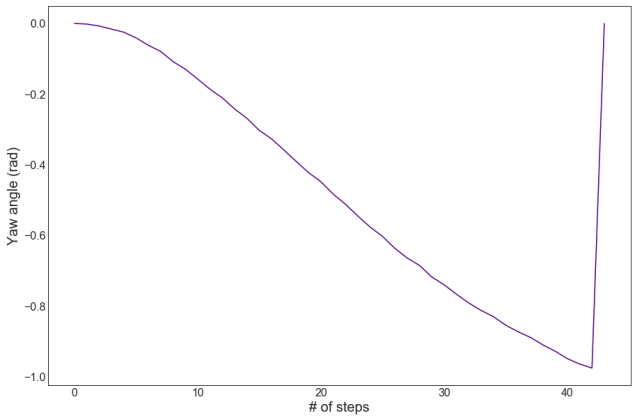
FIGURE A.10: The graphical representation the agent's performance on x, y and z axis in a single episode.



(A)

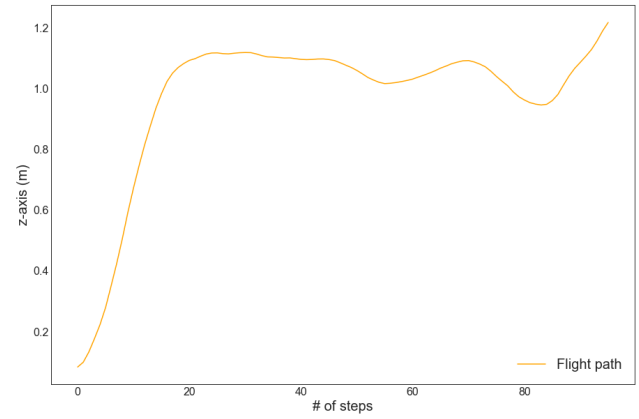


(B)

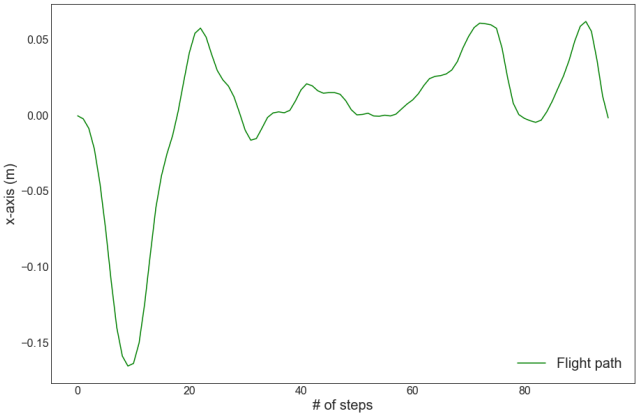


(C)

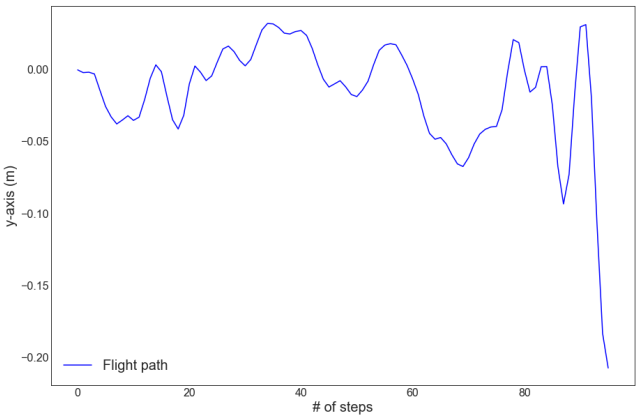
FIGURE A.11: The graphical representation of the agent's ϕ, θ and ψ angles in a single episode.



(A)

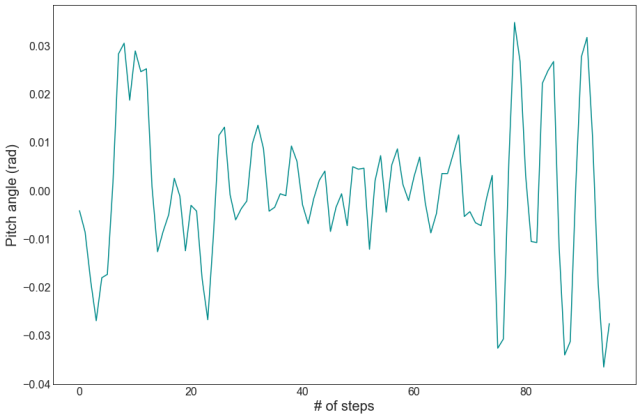


(B)

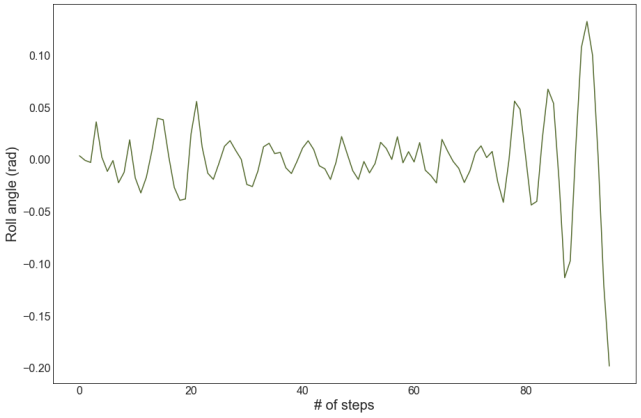


(C)

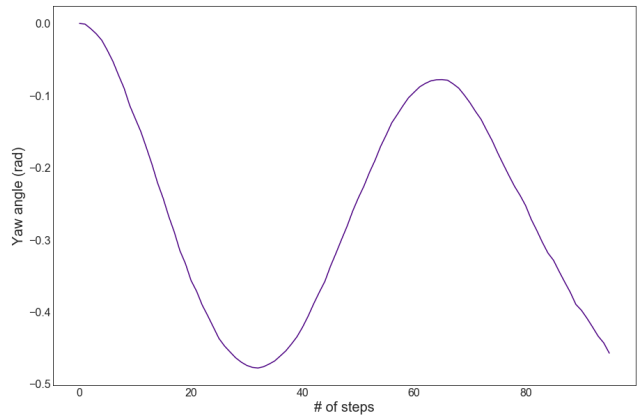
FIGURE A.12: The graphical representation of the agent's performance on x, y and z axis in a single episode.



(A)

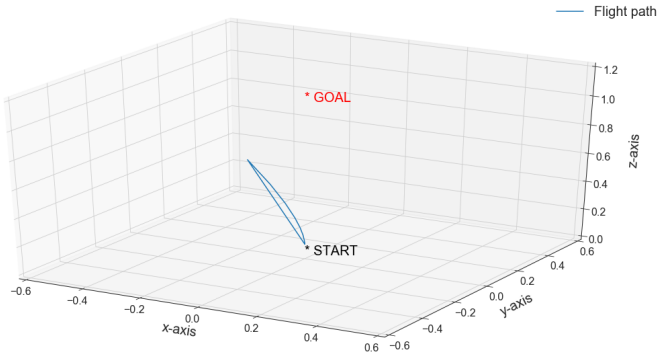


(B)

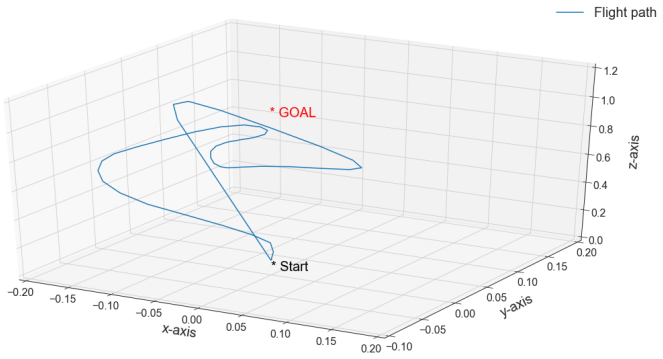


(C)

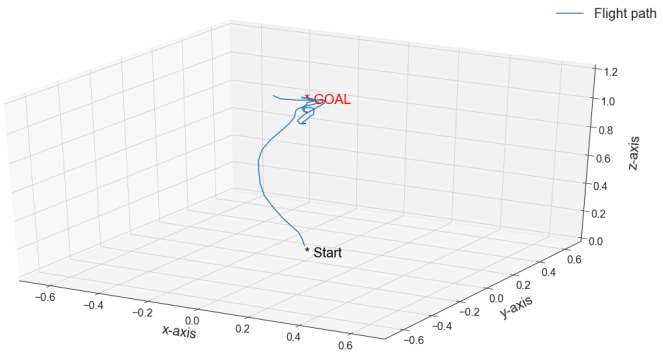
FIGURE A.13: The graphical representation of the agent's ϕ, θ and ψ angles in a single episode.



(A)



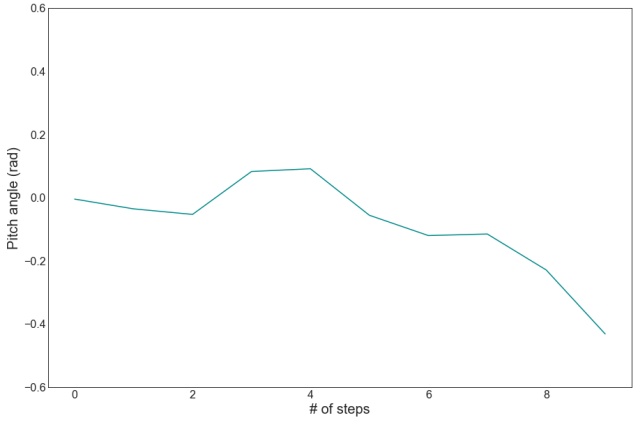
(B)



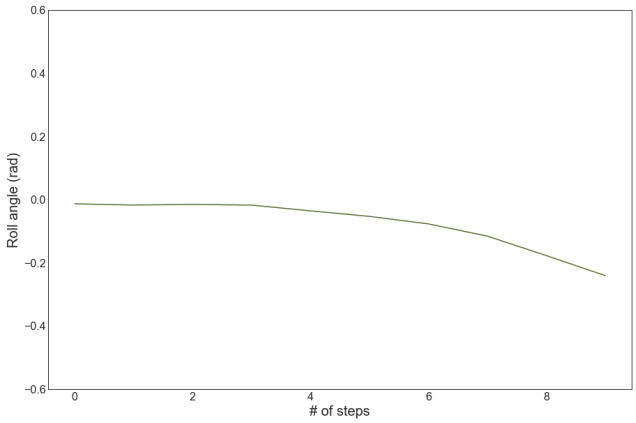
(C)

FIGURE A.14: 3D representation of the hexacopter's performance.

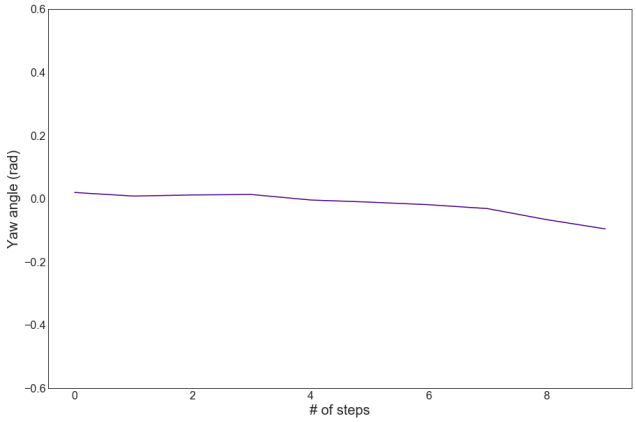
A.3 Flying Manipulation with 7-DoF



(A)

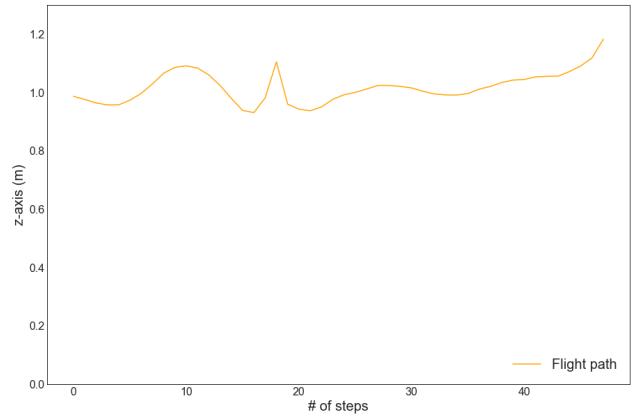


(B)

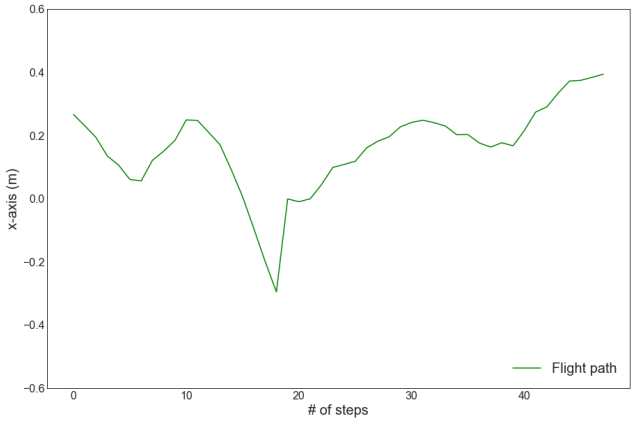


(C)

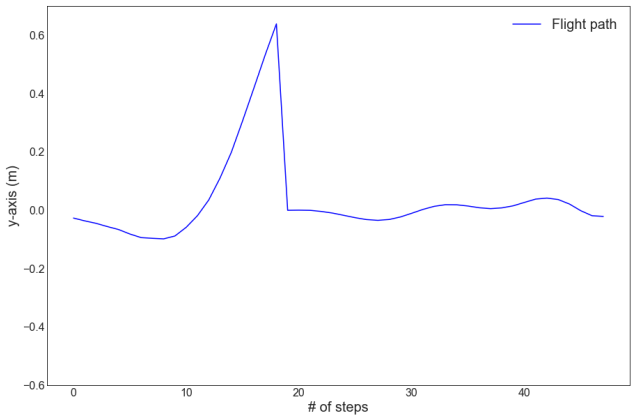
FIGURE A.15: The graphical representation of the agent's ϕ, θ and ψ angles in a single episode.



(A)

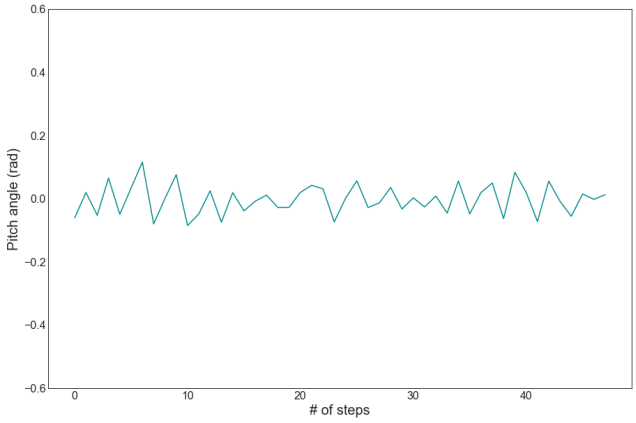


(B)

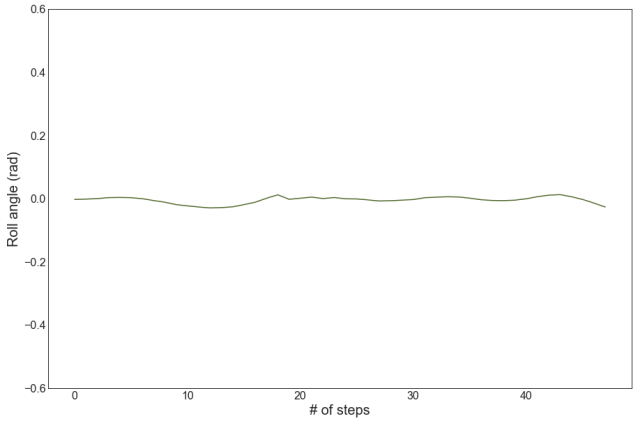


(C)

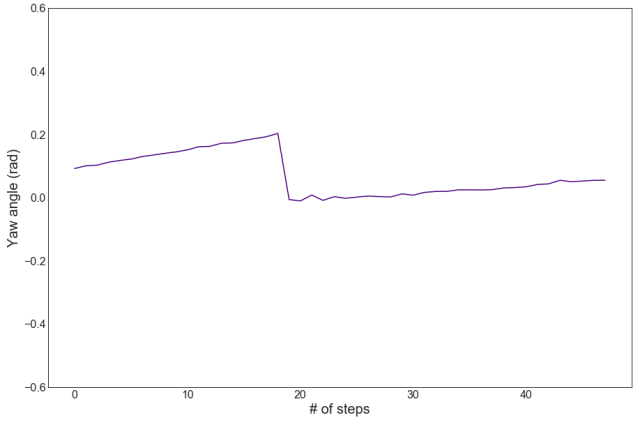
FIGURE A.16: The graphical representation the agent's performance on x, y and z axis in a single episode.



(A)

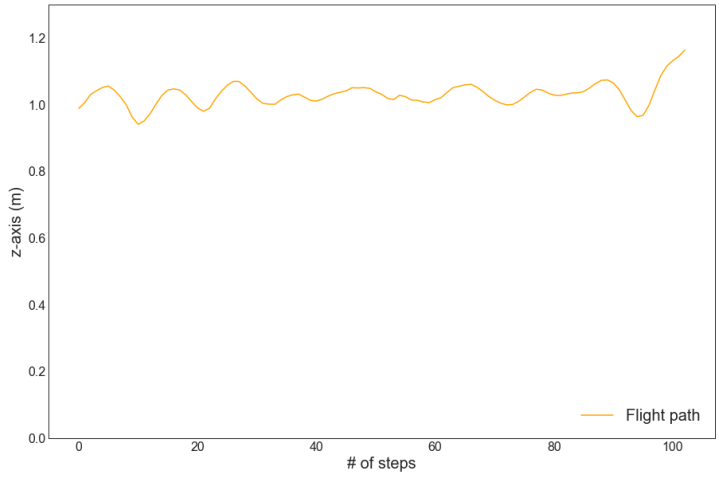


(B)

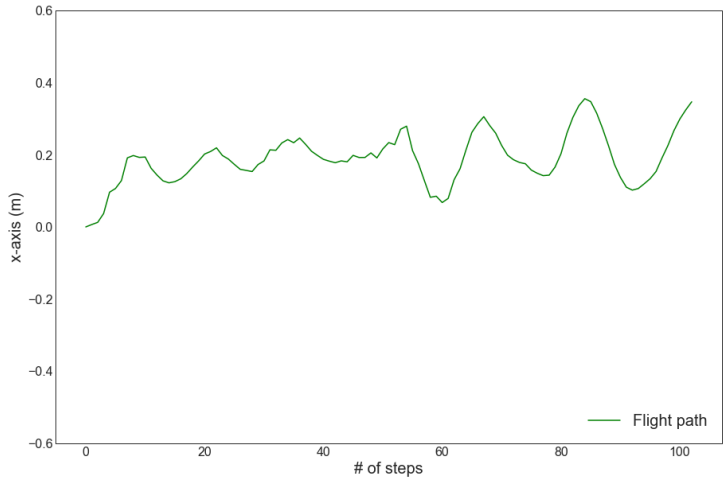


(C)

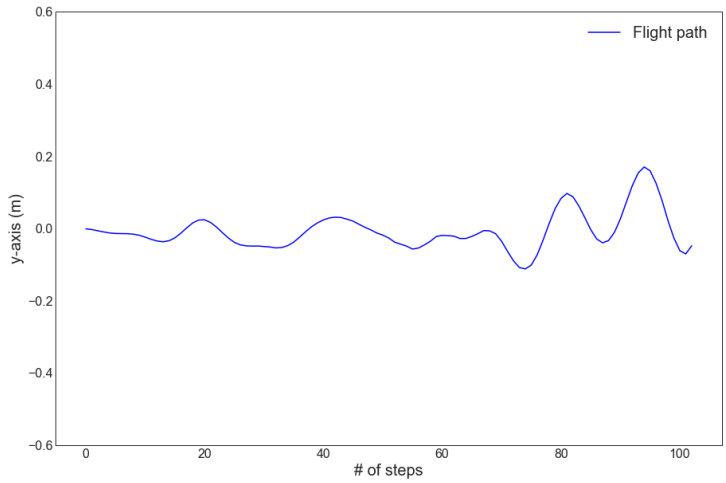
FIGURE A.17: The graphical representation of the agent's ϕ, θ and ψ angles in a single episode.



(A)

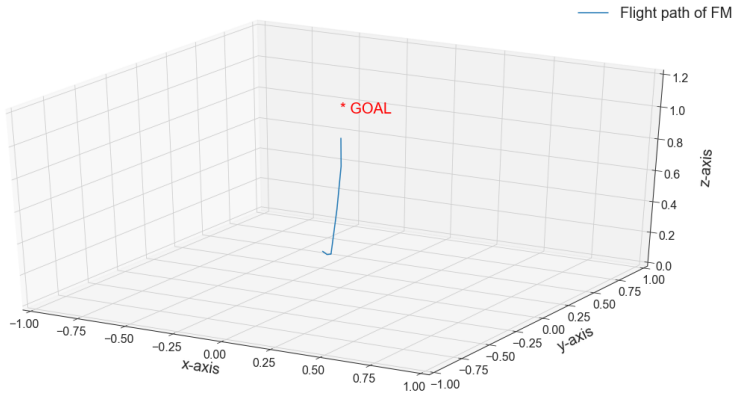


(B)

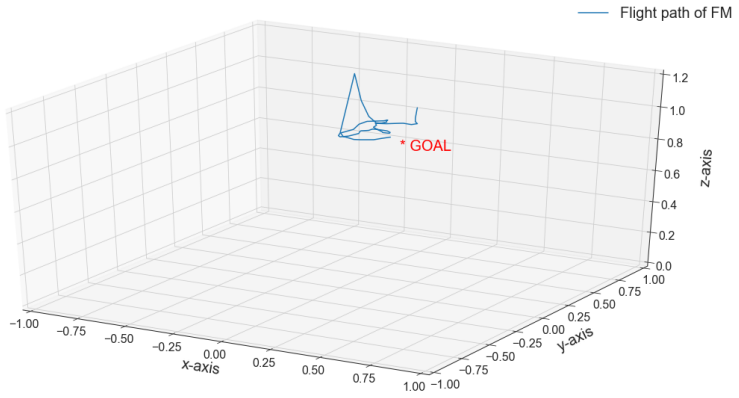


(C)

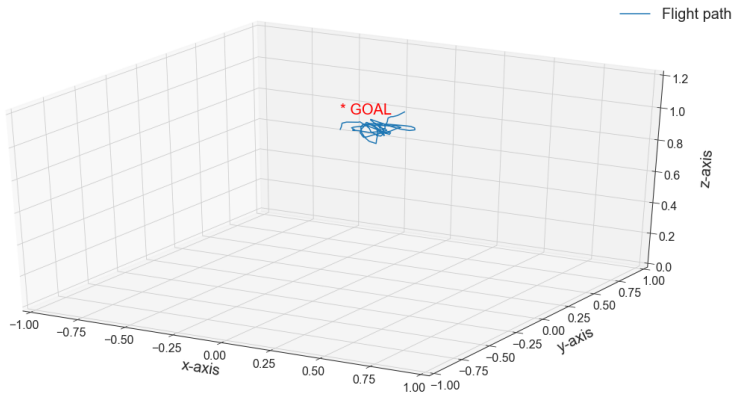
FIGURE A.18: The graphical representation the agent's performance on x, y and z axis in a single episode.



(A)

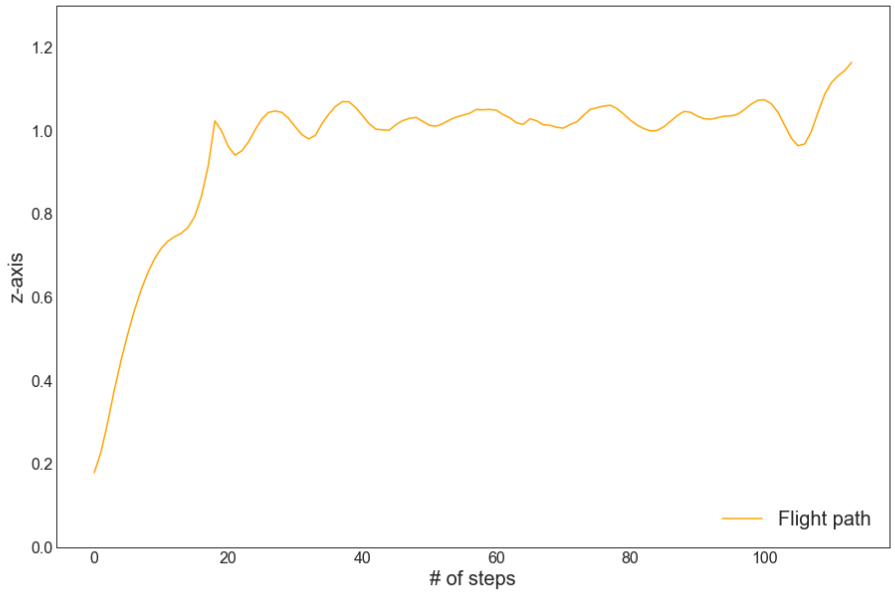


(B)

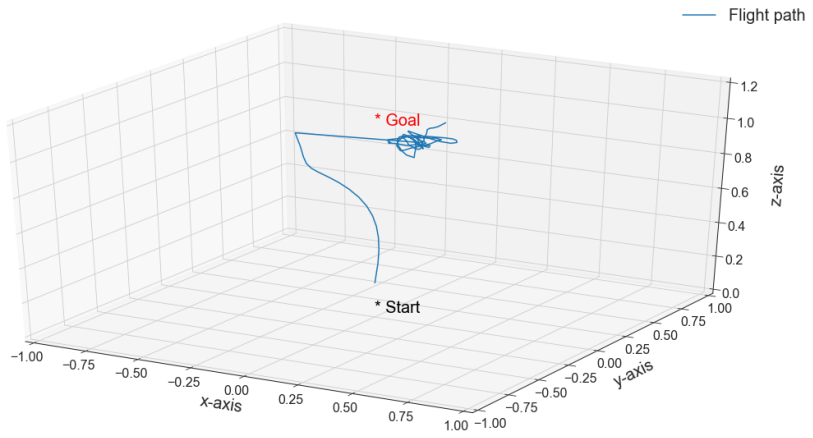


(C)

FIGURE A.19: 3D representation of the hexacopter's performance.



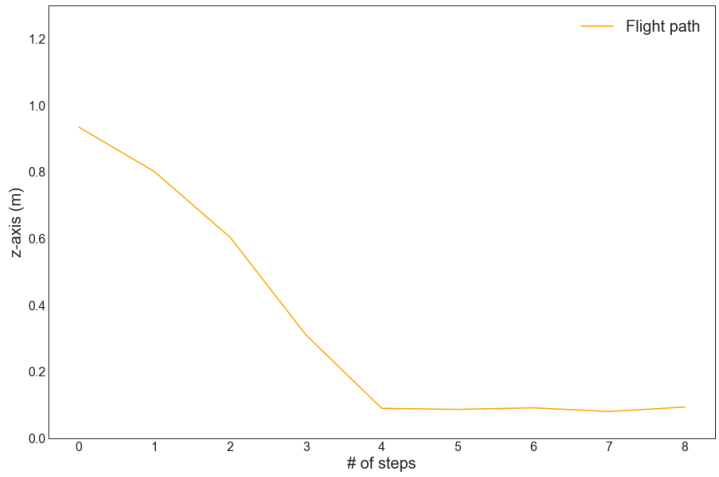
(A)



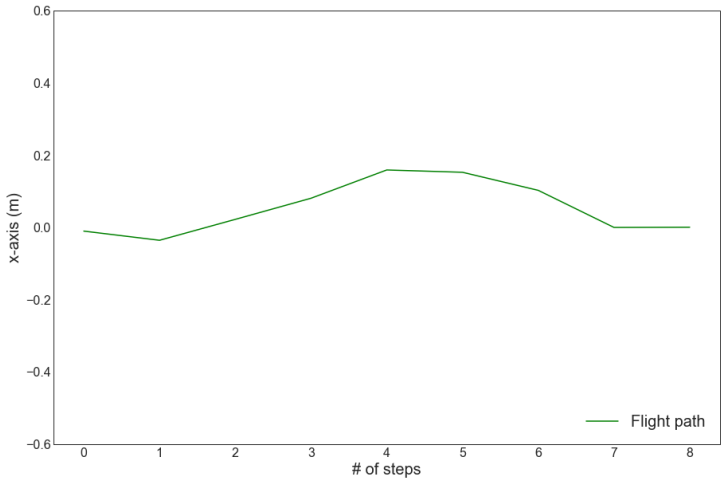
(B)

FIGURE A.20: 3D representation of the hexacopter's performance.

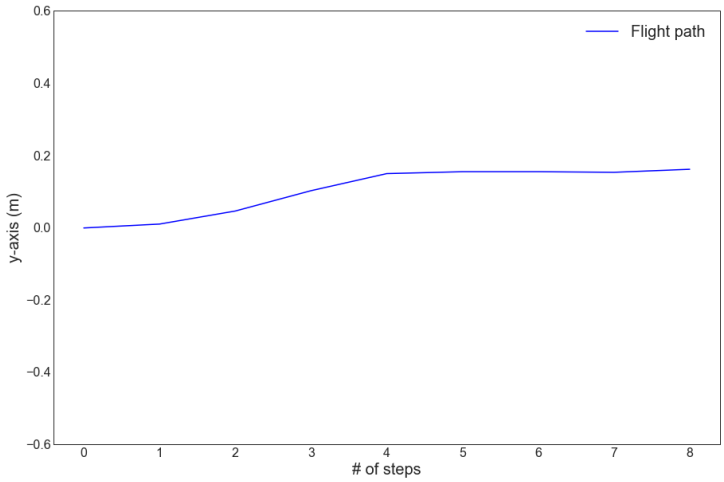
A.4 Flying Manipulation with 8-DoF



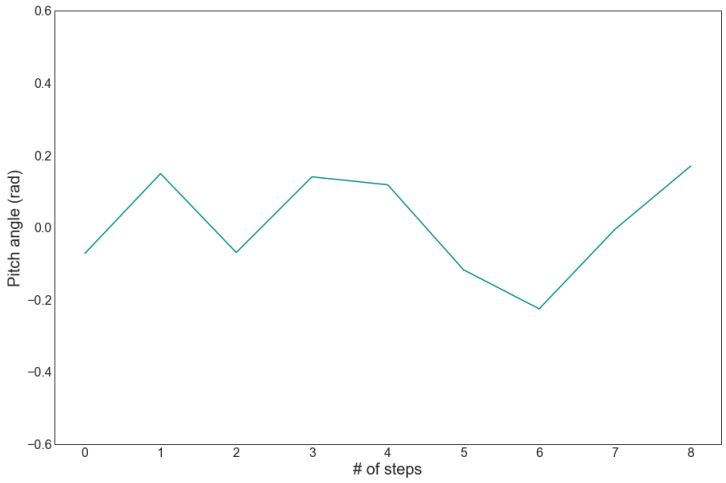
(A)



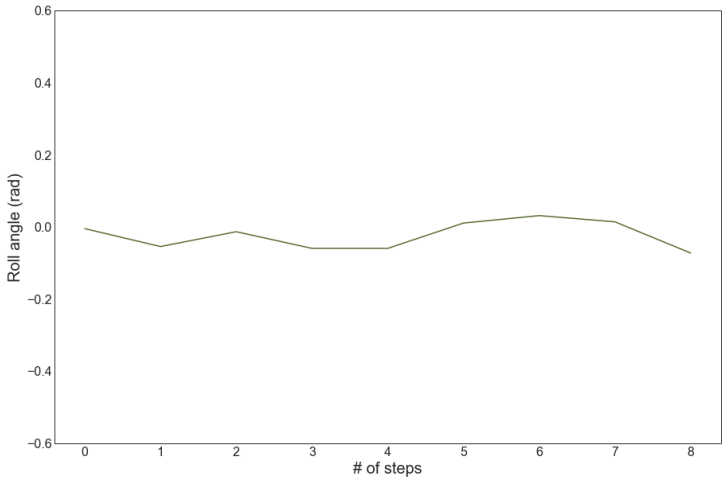
(B)



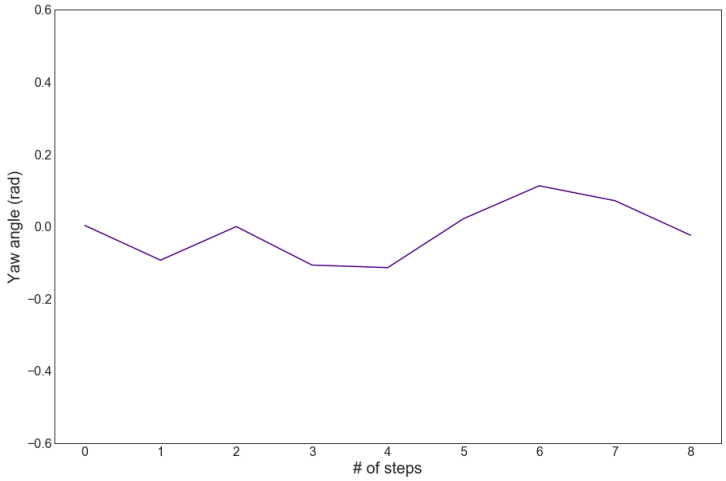
(C)



(A)

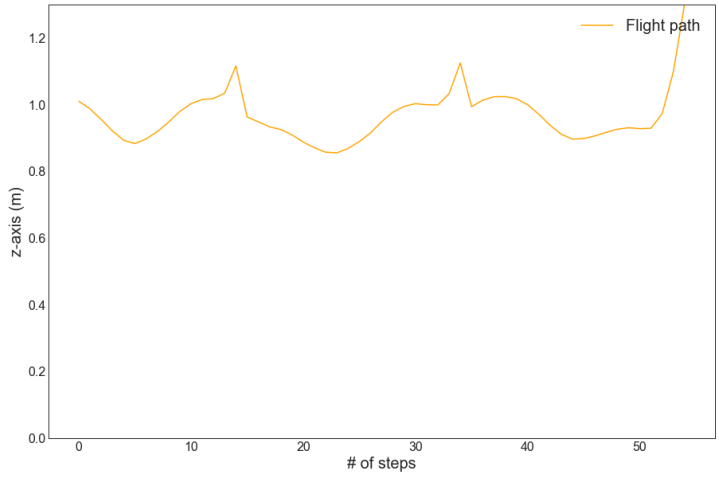


(B)

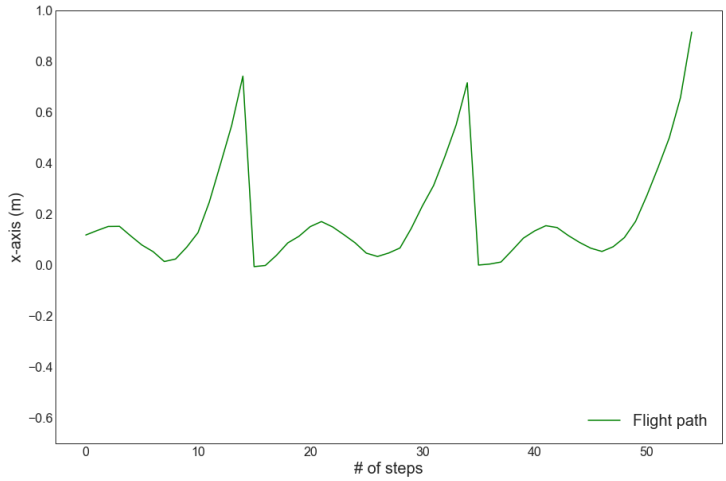


(C)

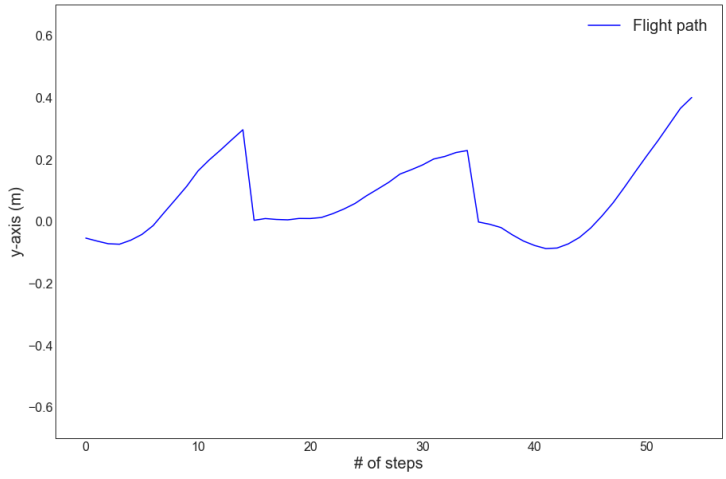
FIGURE A.22: The graphical representation of the agent's ϕ, θ and ψ angles in a single episode.



(A)

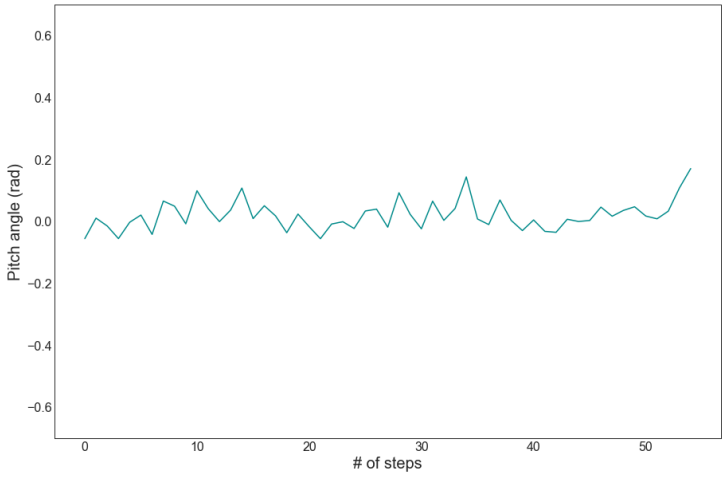


(B)

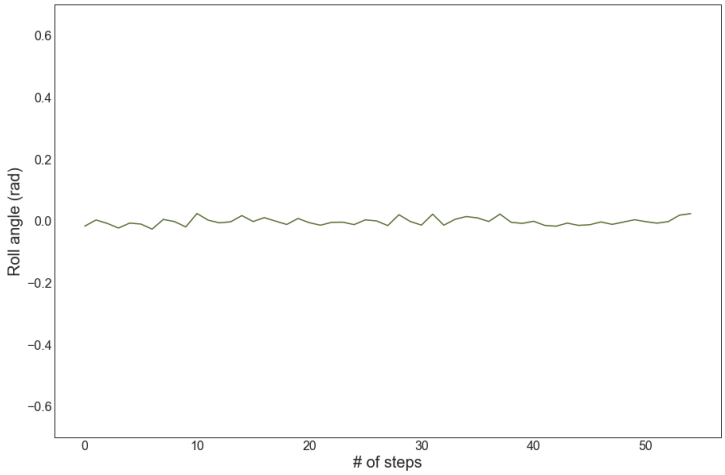


(C)

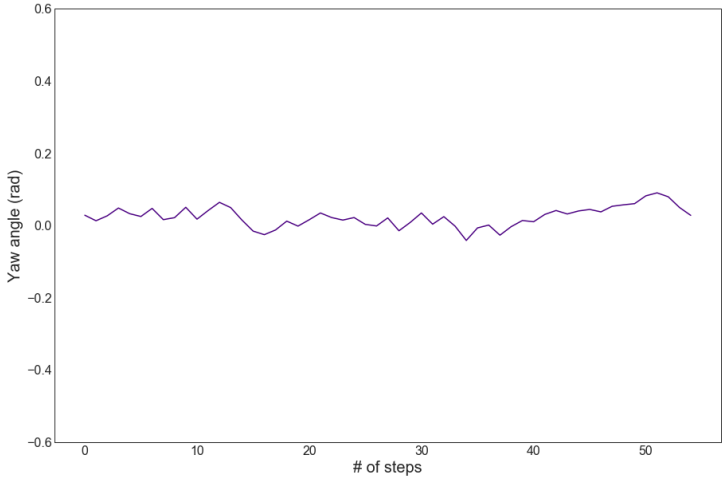
FIGURE A.23: The graphical representation the agent's performance on x, y and z axis in a single episode.



(A)

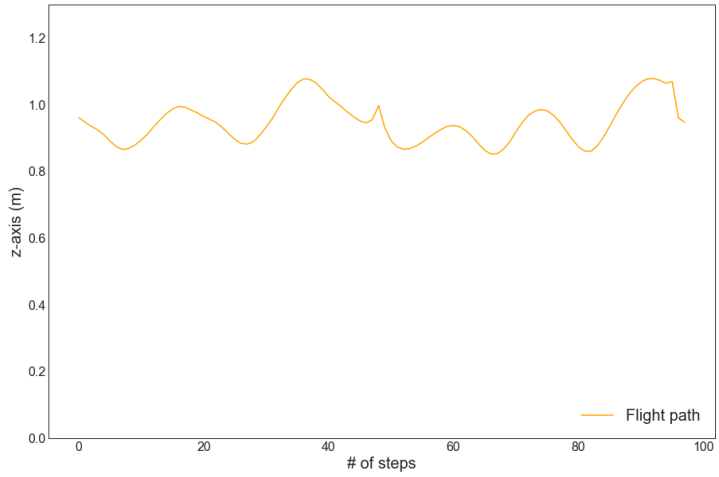


(B)

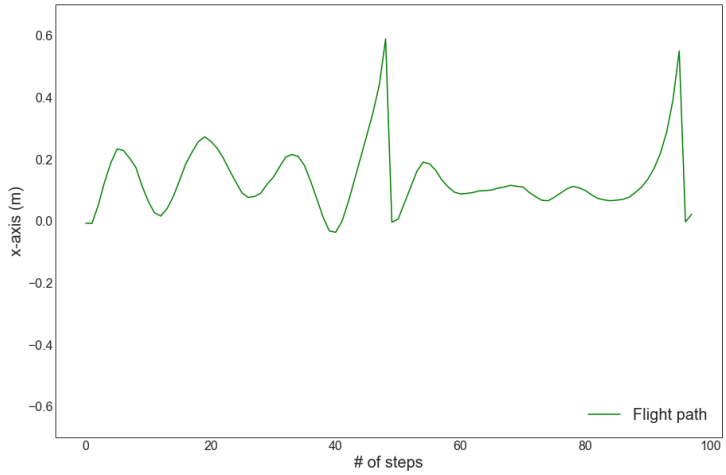


(C)

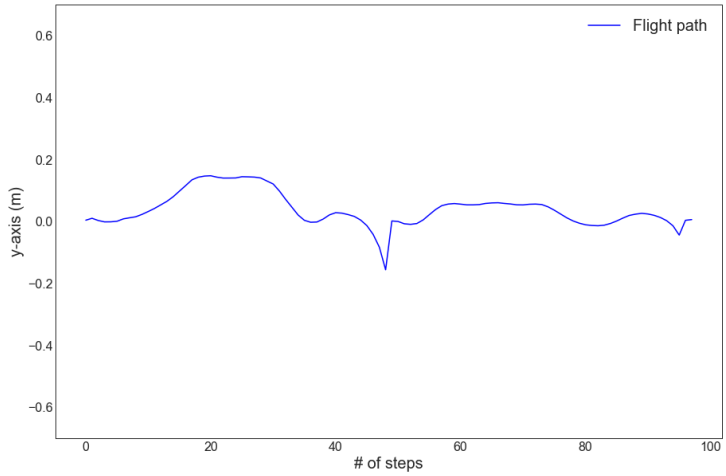
FIGURE A.24: The graphical representation of the agent's ϕ, θ and ψ angles in a single episode.



(A)

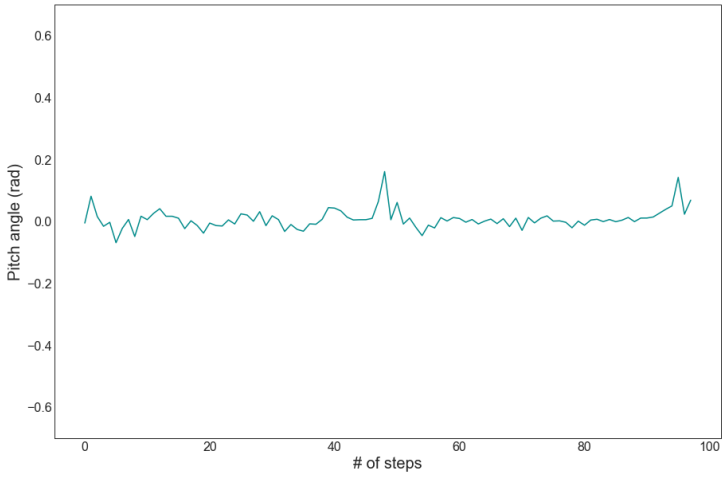


(B)

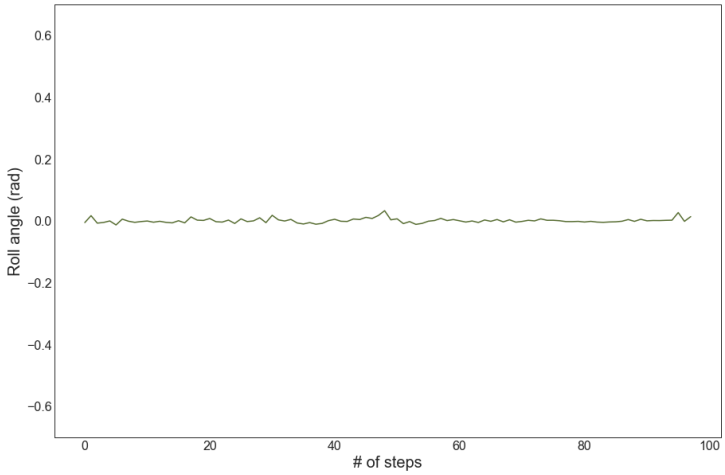


(C)

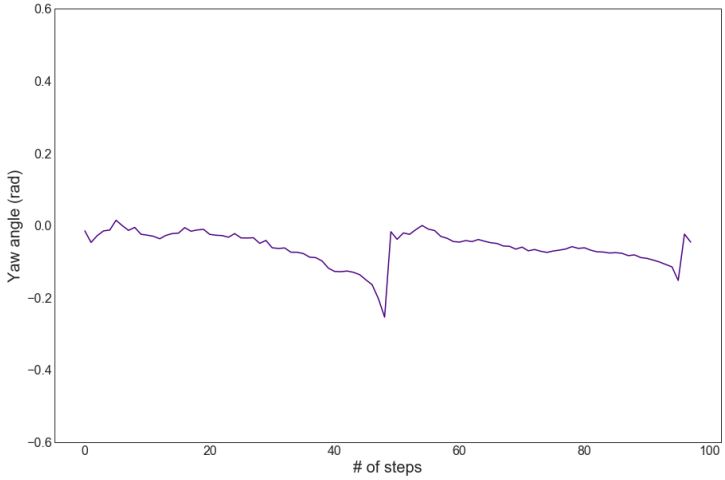
FIGURE A.25: The graphical representation the agent's performance on x, y and z axis in a single episode.



(A)

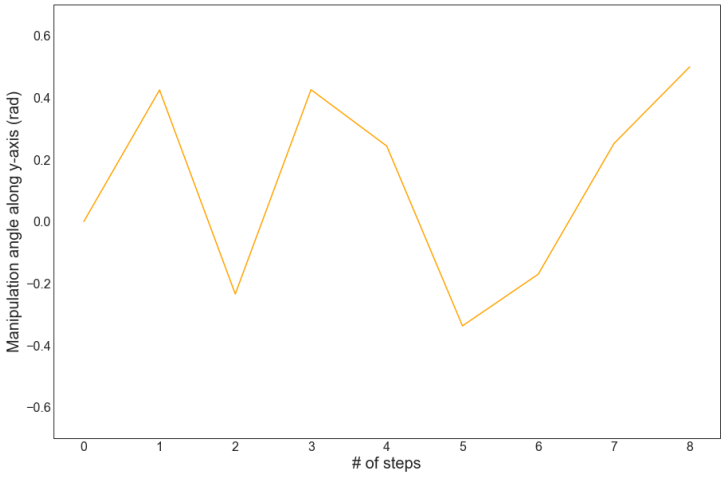


(B)

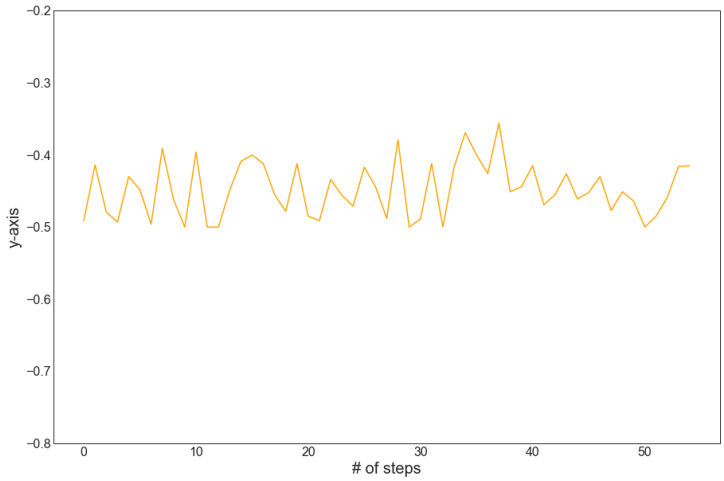


(C)

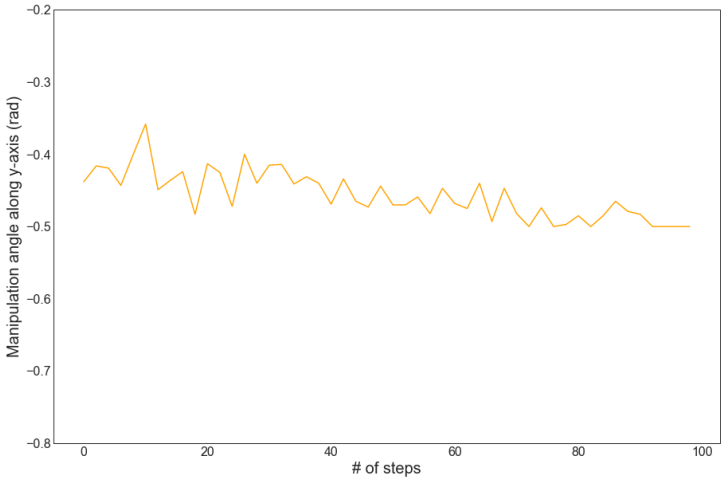
FIGURE A.26: The graphical representation of the agent's ϕ, θ and ψ angles in a single episode.



(A)

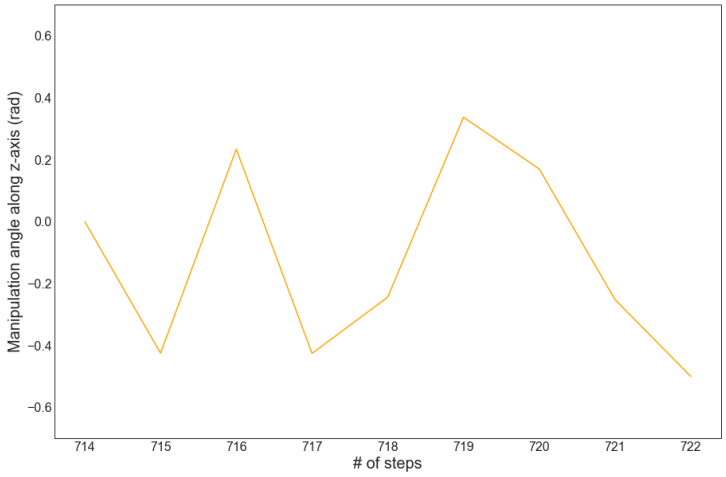


(B)

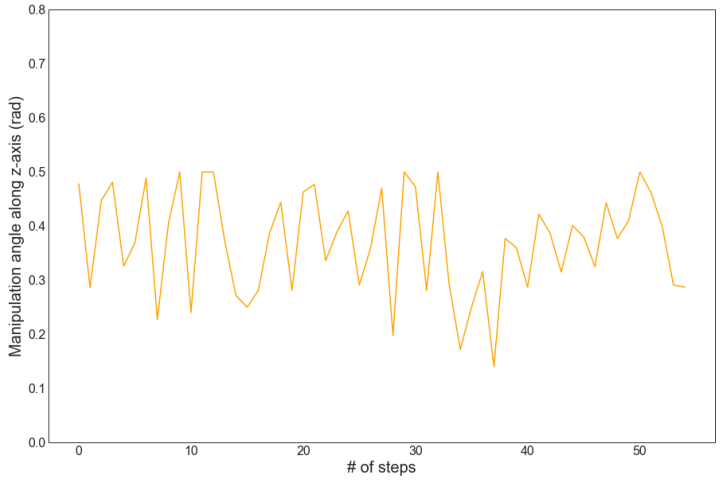


(C)

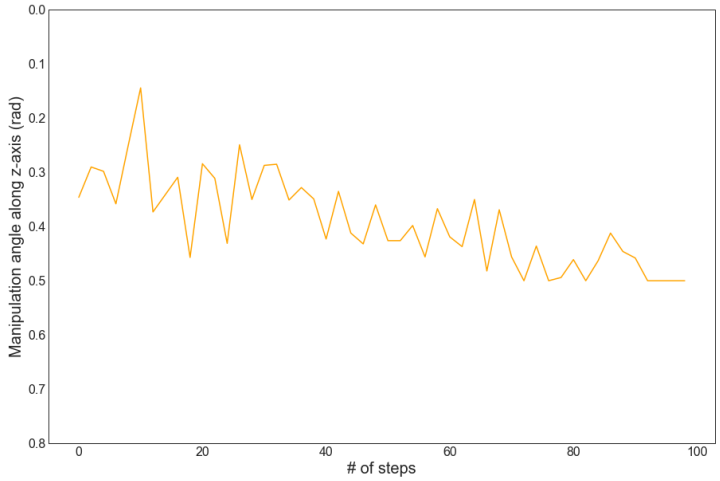
FIGURE A.27: 3D representation of the hexacopter's performance.



(A)

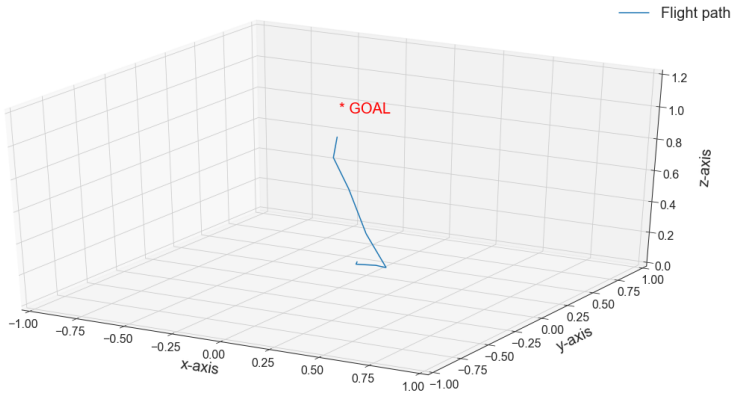


(B)

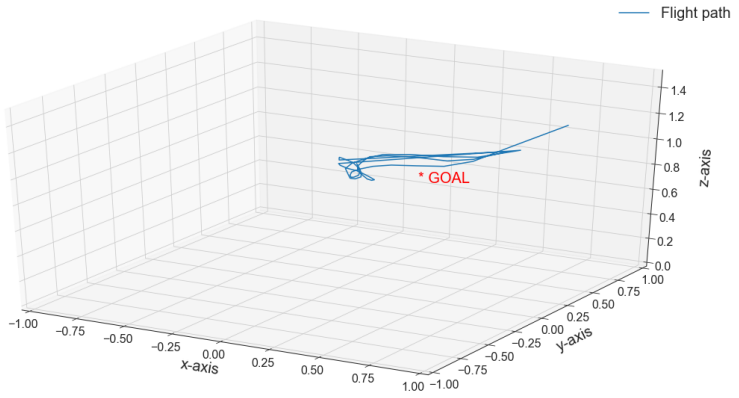


(C)

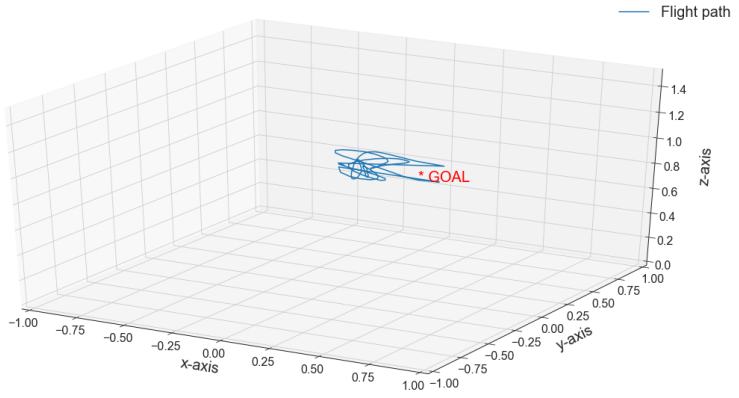
FIGURE A.28: 3D representation of the hexacopter's performance.



(A)

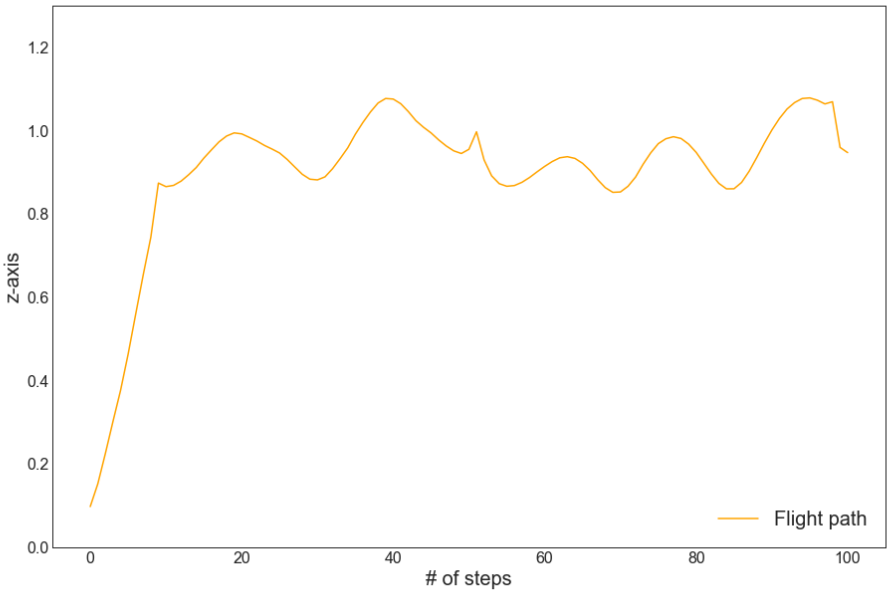


(B)

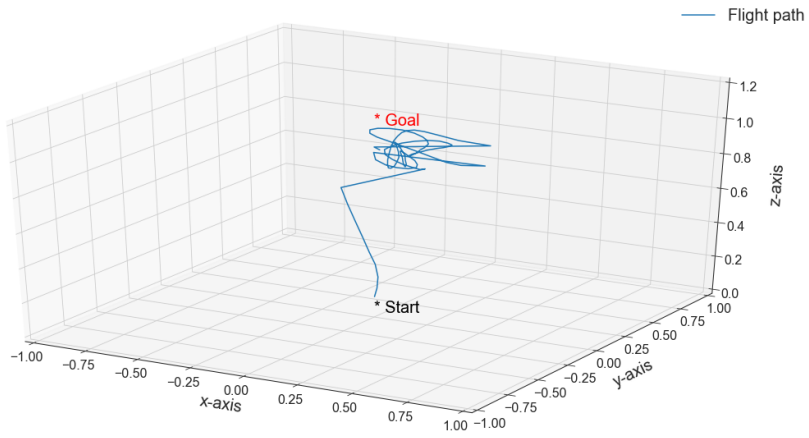


(C)

FIGURE A.29: 3D representation of the hexacopter's performance.



(A)



(B)

FIGURE A.30: 3D representation of the hexacopter's performance.

Bibliography

[1] Pieter Abbeel et al. "An application of reinforcement learning to aerobatic helicopter flight". In: *Advances in neural information processing systems*. 2007, pp. 1–8.

[2] M Jamal Afridi, Ahsan Javed Awan, and Javaid Iqbal. "AWG-Detector: A machine learning tool for the accurate detection of Anomalies due to Wind Gusts (AWG) in the adaptive Altitude control unit of an Aerosonde unmanned Aerial Vehicle". In: *2010 10th International Conference on Intelligent Systems Design and Applications*. IEEE. 2010, pp. 1125–1130.

[3] Abien Fred Agarap. "Deep learning using rectified linear units (relu)". In: *arXiv preprint arXiv:1803.08375* (2018).

[4] Ali Ajdari and Hashem Mahlooji. "An adaptive exploration-exploitation algorithm for constructing metamodels in random simulation using a novel sequential experimental design". In: *Communications in Statistics-Simulation and Computation* 43.5 (2014), pp. 947–968.

[5] Frank Allgöwer and Alex Zheng. *Nonlinear model predictive control*. Vol. 26. Birkhäuser, 2012.

[6] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[7] John E Angus. "Bootstrap one-sided confidence intervals for the log-normal mean". In: *Journal of the Royal Statistical Society: Series D (The Statistician)* 43.3 (1994), pp. 395–401.

[8] G Arleo et al. "Control of quadrotor aerial vehicles equipped with a robotic arm". In: *21St mediterranean conference on control and automation*. IEEE. 2013, pp. 1174–1180.

[9] Minoru Asada et al. "Purposive behavior acquisition for a robot by vision-based reinforcement learning". In: *Journal of the Robotics Society of Japan* 13.1 (1995), pp. 68–74.

[10] Jose Raul Azinheira and Alexandra Moutinho. "Hover control of an UAV with backstepping design including input saturations". In: *IEEE Transactions on control systems technology* 16.3 (2008), pp. 517–526.

[11] Adrià Puigdomènech Badia et al. "Agent57: Outperforming the atari human benchmark". In: *arXiv preprint arXiv:2003.13350* (2020).

[12] Khelifa Baizid et al. "Behavioral control of unmanned aerial vehicle manipulator systems". In: *Autonomous Robots* 41.5 (2017), pp. 1203–1220.

[13] Aryabrata Basu. *Tracking the untracked*. 2019. arXiv: 1909.05327 [cs.HC].

[14] Randal W Beard and Timothy W McLain. *Small unmanned aircraft: Theory and practice*. Princeton university press, 2012.

[15] Neeraj Bhargava et al. "Decision tree analysis on j48 algorithm for data mining". In: *Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering* 3.6 (2013).

[16] Haitham Bou-Ammar, Holger Voos, and Wolfgang Ertel. "Controller design for quadrotor uavs using reinforcement learning". In: *2010 IEEE International Conference on Control Applications*. IEEE. 2010, pp. 2130–2135.

[17] Haitham Bou-Ammar, Holger Voos, and Wolfgang Ertel. "Controller design for quadrotor uavs using reinforcement learning". In: *2010 IEEE International Conference on Control Applications*. IEEE. 2010, pp. 2130–2135.

[18] Alexander de Brebisson and Giovanni Montana. "Deep neural networks for anatomical brain segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2015, pp. 20–28.

[19] Tom Brotherton and Ryan Mackey. "Anomaly detector fusion processing for advanced military aircraft". In: *2001 IEEE Aerospace Conference Proceedings (Cat. No. 01TH8542)*. Vol. 6. IEEE. 2001, pp. 3125–3137.

[20] Tim Brys et al. "Multi-objectivization of reinforcement learning problems by reward shaping". In: *2014 international joint conference on neural networks (IJCNN)*. IEEE. 2014, pp. 2315–2322.

[21] Tim Brys et al. "Reinforcement learning from demonstration through shaping". In: *Twenty-fourth international joint conference on artificial intelligence*. 2015.

[22] Jian Bu et al. "Integrated method for the UAV navigation sensor anomaly detection". In: *IET Radar, Sonar & Navigation* 11.5 (2017), pp. 847–853.

[23] Jian Bu et al. "Integrated method for the UAV navigation sensor anomaly detection". In: *IET Radar, Sonar & Navigation* 11.5 (2017), pp. 847–853.

[24] Fabrizio Caccavale et al. "Adaptive control for UAVs equipped with a robotic arm". In: *IFAC Proceedings Volumes* 47.3 (2014), pp. 11049–11054.

[25] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[26] José M. Cañas and Vicente Matellán. "From bio-inspired vs. psycho-inspired to etho-inspired robots." In: *Robotics and Autonomous Systems* 55.12 (2007), pp. 841–850. DOI: https://gsyc.urjc.es/jmplaza/papers/ras2007-jdec.pdf.

[27] Xie Chen et al. "Pipelined back-propagation for context-dependent deep neural networks". In: *Thirteenth Annual Conference of the International Speech Communication Association*. 2012.

[28] Yafeng Chen et al. "On-line and non-invasive anomaly detection system for unmanned aerial vehicle". In: *2017 Prognostics and System Health Management Conference (PHM-Harbin)*. IEEE. 2017, pp. 1–7.

[29] Melanie Coggan. "Exploration and exploitation in reinforcement learning". In: *Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University* (2004).

[30] Peter Corke. *Robotics, vision and control: fundamental algorithms in MATLAB® second, completely revised*. Vol. 118. Springer, 2017.

[31] Stanford.edu CS231. *Convolutional neural network*. URL: http://cs231n.stanford.edu/. (2020).

[32] Brian J Daiuto, Tom T Hartley, and Stephen P Chicatelli. *The hyperbolic map and applications to the linear quadratic regulator*. Vol. 110. Springer, 1989.

[33] Marc Peter Deisenroth, Carl Edward Rasmussen, and Dieter Fox. "Learning to control a low-cost manipulator using data-efficient reinforcement learning". In: *Robotics: Science and Systems VII* (2011), pp. 57–64.

[34] Sam Michael Devlin and Daniel Kudenko. "Dynamic potential-based reward shaping". In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS. 2012, pp. 433–440.

[35] Wenjie Dong et al. "Command filtered adaptive backstepping". In: *IEEE Transactions on Control Systems Technology* 20.3 (2011), pp. 566–580.

[36] Yong Duan et al. "Unmanned Aerial Vehicle Sensing Data Anomaly Detection by Relevance Vector Machine". In: *2017 International Conference on Sensing, Diagnostics, Prognostics, and Control (SDPC)*. IEEE. 2017, pp. 638–641.

[37] Emlid. *Navio2 - Autopilot HAT for Raspberry Pi Powered by ArduPilot and ROS*. URL: https://docs.emlid.com/navio2/.

[38] Emlid. *Navio2 - Collection of drivers and examples for Navio*.

[39] Manfred Eppe et al. "Combining deep learning for visuomotor coordination with object identification to realize a high-level interface for robot object-picking". In: *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE. 2017, pp. 612–617.

[40] Damien Ernst et al. "Approximate Value Iteration in the Reinforcement Learning Context. Application to Electrical Power System Control." In: *International Journal of Emerging Electric Power Systems* 3.1 (2005).

[41] Hossein Eslamiat et al. "Autonomous Waypoint Planning, Optimal Trajectory Generation and Nonlinear Tracking Control for Multi-rotor UAVs". In: *2019 18th European Control Conference (ECC)*. IEEE. 2019, pp. 2695–2700.

[42] Bruno S Faiçal et al. "Fine-tuning of UAV control rules for spraying pesticides on crop fields". In: *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*. IEEE. 2014, pp. 527–533.

[43] Aleksandra Faust et al. "Automated aerial suspended cargo delivery through reinforcement learning". In: *Artificial Intelligence* 247 (2017), pp. 381–398.

[44] Diego Ferigo et al. "Gym-Ignition: Reproducible Robotic Simulations for Reinforcement Learning". In: *2020 IEEE/SICE International Symposium on System Integration (SII)*. IEEE. 2020, pp. 885–890.

[45] Rafael Figueroa et al. "Reinforcement learning for balancing a flying inverted pendulum". In: *Proceeding of the 11th World Congress on Intelligent Control and Automation*. IEEE. 2014, pp. 1787–1793.

[46] Chelsea Finn et al. "Deep spatial autoencoders for visuomotor learning". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 512–519.

[47] Paul Freeman et al. "Model-based and data-driven fault detection performance for a small UAV". In: *IEEE/ASME Transactions on mechatronics* 18.4 (2013), pp. 1300–1309.

[48] Justin Fu, Katie Luo, and Sergey Levine. "Learning robust rewards with adversarial inverse reinforcement learning". In: *arXiv preprint arXiv:1710.11248* (2017).

[49] Fadri Furrer et al. "RotorS—A modular Gazebo MAV simulator framework". In: *Robot Operating System (ROS)*. Springer, 2016, pp. 595–625.

[50] Fadri Furrer et al. "RotorS—A modular gazebo MAV simulator framework". In: *Robot Operating System (ROS)*. Springer, 2016, pp. 595–625.

[51] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. "Learning to fly by crashing". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3948–3955.

[52] A Gasparetto and L Scalera. "A brief history of industrial robotics in the 20th century". In: *Advances in Historical Studies* 8.1 (2019), pp. 24–35.

[53] Piotr Gawłowicz and Anatolij Zubow. "ns3-gym: Extending openai gym for networking research". In: *arXiv preprint arXiv:1810.03943* (2018).

[54] Amy Greenwald, Keith Hall, and Roberto Serrano. "Correlated Q-learning". In: *ICML*. Vol. 20. 1. 2003, p. 242.

[55] Ivo Grondman et al. "A survey of actor-critic reinforcement learning: Standard and natural policy gradients". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012), pp. 1291–1307.

[56] Marek Grześ and Daniel Kudenko. "Online learning of shaping rewards in reinforcement learning". In: *Neural Networks* 23.4 (2010), pp. 541–550.

[57] Marek Grzes and Daniel Kudenko. "Plan-based reward shaping for reinforcement learning". In: *2008 4th International IEEE Conference Intelligent Systems*. Vol. 2. IEEE. 2008, pp. 10–22.

[58] Shixiang Gu et al. "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 3389–3396.

[59] Shixiang Gu et al. "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 3389–3396.

[60] Aditya Gudimella et al. "Deep reinforcement learning for dexterous manipulation with concept networks". In: *arXiv preprint arXiv:1709.06977* (2017).

[61] Tuomas Haarnoja et al. "Composable deep reinforcement learning for robotic manipulation". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 6244–6251.

[62] Peter Harrington. *Machine learning in action*. Manning Publications Co., 2012.

[63] Hado V Hasselt. "Double Q-learning". In: *Advances in neural information processing systems*. 2010, pp. 2613–2621.

[64] Peter Henderson et al. "Deep reinforcement learning that matters". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[65] Michael A Henson. "Nonlinear model predictive control: current status and future directions". In: *Computers & Chemical Engineering* 23.2 (1998), pp. 187–202.

[66] Rogelio G Hernandez-Garcia and H Rodriguez-Cortes. "Transition flight control of a cyclic tiltrotor uav based on the gain-scheduling strategy". In: *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2015, pp. 951–956.

[67] Gabriel Hoffmann et al. "Quadrotor helicopter flight dynamics and control: Theory and experiment". In: *AIAA guidance, navigation and control conference and exhibit*. 2007, p. 6461.

[68] Jemin Hwangbo et al. "Control of a quadrotor with reinforcement learning". In: *IEEE Robotics and Automation Letters* 2.4 (2017), pp. 2096–2103.

[69] Jemin Hwangbo et al. "Control of a quadrotor with reinforcement learning". In: *IEEE Robotics and Automation Letters* 2.4 (2017), pp. 2096–2103.

[70] Jemin Hwangbo et al. "Control of a quadrotor with reinforcement learning". In: *IEEE Robotics and Automation Letters* 2.4 (2017), pp. 2096–2103.

[71] Ascending Technologies | A Part of Intel. *Ascending Technologies*. URL: http://www.asctec.de.

[72] Artificial Intelligence. *Artificial Intelligence (AI) Robots Market*. URL: https://www.marketsandmarkets.com/Market-Reports/artificial-intelligence-robots-market-120550497.html. (accessed: Feb 2018).

[73] Riashat Islam et al. "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control". In: *arXiv preprint arXiv:1708.04133* (2017).

[74] Nathalie Japkowicz. "Supervised versus unsupervised binary-learning by feedforward neural networks". In: *Machine Learning* 42.1-2 (2001), pp. 97–122.

[75] Ran Jiao et al. "Adaptive robust control of quadrotor with a 2-degree-of-freedom robotic arm". In: *Advances in Mechanical Engineering* 10.8 (2018), p. 1687814018778639.

[76] Antonio E Jimenez-Cano et al. "Control of an aerial robot with multi-link arm for assembly tasks". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 4916–4921.

[77] Sham M Kakade. "A natural policy gradient". In: *Advances in neural information processing systems*. 2002, pp. 1531–1538.

[78] Dmitry Kalashnikov et al. "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation". In: *arXiv preprint arXiv:1806.10293* (2018).

[79] Somasundar Kannan, Miguel A Olivares-Mendez, and Holger Voos. "Modeling and control of aerial manipulation vehicle with visual sensor". In: *IFAC Proceedings Volumes* 46.30 (2013), pp. 303–309.

[80] Somasundar Kannan, Miguel A Olivares-Mendez, and Holger Voos. "Modeling and control of aerial manipulation vehicle with visual sensor". In: *IFAC Proceedings Volumes* 46.30 (2013), pp. 303–309.

[81] Somasundar Kannan et al. "Control of aerial manipulation vehicle in operational space". In: *2016 8th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. IEEE. 2016, pp. 1–4.

[82] Somasundar Kannan et al. "Control of aerial manipulation vehicle in operational space". In: *2016 8th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. IEEE. 2016, pp. 1–4.

[83] Aman Kataria and MD Singh. "A review of data classification using k-nearest neighbour algorithm". In: *International Journal of Emerging Technology and Advanced Engineering* 3.6 (2013), pp. 354–360.

[84] Aman Kataria and MD Singh. "A review of data classification using k-nearest neighbour algorithm". In: *International Journal of Emerging Technology and Advanced Engineering* 3.6 (2013), pp. 354–360.

[85] Kjell Kersandt. "Deep reinforcement learning as control method for autonomous uavs". MA thesis. Universitat Politècnica de Catalunya, 2018.

[86] Eliahu Khalastchi et al. "Online anomaly detection in unmanned vehicles". In: *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems. 2011, pp. 115–122.

[87] Byungchan Kim et al. "Impedance learning for robotic contact tasks using natural actor-critic algorithm". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 40.2 (2009), pp. 433–443.

[88] H Jin Kim et al. "Autonomous helicopter flight via reinforcement learning". In: *Advances in neural information processing systems*. 2004, pp. 799–806.

[89] Kwang In Kim, Keechul Jung, and Hang Joon Kim. "Face recognition using kernel principal component analysis". In: *IEEE signal processing letters* 9.2 (2002), pp. 40–42.

[90] Wonchul Kim et al. "Three-link planar arm control using reinforcement learning". In: *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. IEEE. 2017, pp. 424–428.

[91] Yoon Kim. "Convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1408.5882* (2014).

[92] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[93] Jens Kober, J Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[94] Jens Kober and Jan R Peters. "Policy search for motor primitives in robotics". In: *Advances in neural information processing systems*. 2009, pp. 849–856.

[95] William Koch et al. "Reinforcement learning for UAV attitude control". In: *ACM Transactions on Cyber-Physical Systems* 3.2 (2019), pp. 1–21.

[96] Miroslav Kubat. *An introduction to machine learning*. Springer, 2017.

[97] Thanard Kurutach et al. "Model-ensemble trust-region policy optimization". In: *arXiv preprint arXiv:1802.10592* (2018).

[98] Autonomous Robots Lab. *RotorS Simulator*. URL: https://www.autonomousrobotslab.com/rotors-simulator1.html.

[99] Michail G Lagoudakis and Ronald Parr. "Least-squares policy iteration". In: *Journal of machine learning research* 4.Dec (2003), pp. 1107–1149.

[100] Michail G Lagoudakis and Ronald Parr. "Least-squares policy iteration". In: *Journal of machine learning research* 4.Dec (2003), pp. 1107–1149.

[101] Maximilian Laiacker, Felix Huber, and Konstantin Kondak. "High accuracy visual servoing for aerial manipulation using a 7 degrees of freedom industrial manipulator". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 1631–1636.

[102] Joel Lehman et al. "The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities". In: *Artificial Life* 26.2 (2020), pp. 274–306.

[103] Andrew Levy, Robert Platt, and Kate Saenko. "Hierarchical actor-critic". In: *arXiv preprint arXiv:1712.00948* 12 (2017).

[104] Yilan Li et al. "Autonomous waypoints planning and trajectory generation for multi-rotor UAVs". In: *Proceedings of the Workshop on Design Automation for CPS and IoT*. 2019, pp. 31–40.

[105] Yu-Jhe Li et al. "Deep Reinforcement Learning for Playing 2.5 D Fighting Games". In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE. 2018, pp. 3778–3782.

[106] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[107] Qing Lin et al. "Adaptive flight control design for quadrotor UAV based on dynamic inversion and neural networks". In: *2013 Third International Conference on Instrumentation, Measurement, Computer, Communication and Control*. IEEE. 2013, pp. 1461–1466.

[108] Raz Lin, Eliyahu Khalastchi, and Gal A Kaminka. "Detecting anomalies in unmanned vehicles using the mahalanobis distance". In: *2010 IEEE international conference on robotics and automation*. IEEE. 2010, pp. 3038–3044.

[109] Vincenzo Lippiello and Fabio Ruggiero. "Cartesian impedance control of a UAV with a robotic arm". In: *IFAC Proceedings Volumes* 45.22 (2012), pp. 704–709.

[110] Chi Harold Liu et al. "Energy-efficient UAV control for effective and fair communication coverage: A deep reinforcement learning approach". In: *IEEE Journal on Selected Areas in Communications* 36.9 (2018), pp. 2059–2070.

[111] Cunjia Liu, Wen-Hua Chen, and John Andrews. "Tracking control of small-scale helicopters using explicit nonlinear MPC augmented with disturbance observers". In: *Control Engineering Practice* 20.3 (2012), pp. 258–268.

[112] Liansheng Liu et al. "MEMS Sensor data anomaly detection for the UAV flight control subsystem". In: *2018 IEEE SENSORS*. IEEE. 2018, pp. 1–4.

[113] Ming Liu, Greg K Egan, and Fendy Santoso. "Modeling, autopilot design, and field tuning of a UAV with minimum control surfaces". In: *IEEE Transactions on Control Systems Technology* 23.6 (2015), pp. 2353–2360.

[114] Nestor Gonzalez Lopez et al. "gym-gazebo2, a toolkit for reinforcement learning using ROS 2 and Gazebo". In: *arXiv preprint arXiv:1903.06278* (2019).

[115] A Rupam Mahmood et al. "Setting up a reinforcement learning task with a real-world robot". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 4635–4640.

[116] Anush Manukyan. *Online Degradation Identification of UAV*. URL: https://github.com/anushmanukyan/online-degradation-identification-uav.git. (published: August 2020).

[117] Anush Manukyan. *RotorS Gym Framework*. URL: https://github.com/anushmanukyan/rotors-gym-framework.git. (published: August 2020).

[118] Anush Manukyan. *Running the trained model on a physical hexacopter UAV using the Cyber Gym Robotics platform*. URL: https://youtu.be/tKvtde2oOiE. (published: August 2020).

[119] Anush Manukyan. *Running the trained model on a physical hexacopter UAV using the Cyber Gym Robotics platform*. URL: https://youtu.be/IU0ja_fFwvY. (published: August 2020).

[120] Anush Manukyan. *The learning process of a flying manipulation with 7-DoF for performing an end-effector positioning task*. URL: https://youtu.be/aPpqRoHf_eU. (published: August 2020).

[121] Anush Manukyan. *The learning process of a flying manipulation with 8-DoF for performing an end-effector positioning task*. URL: https://youtu.be/HGUtA9zXOxA. (published: August 2020).

[122] Anush Manukyan. *The learning process of a hexacopter UAV for performing a navigation task using DRL*. URL: https://youtu.be/LtVGceXUDIY. (published: August 2020).

[123] Anush Manukyan. *The learning process of a hexacopter UAV for performing a stable hovering task using DRL*. URL: https://youtu.be/2tLAtk6Tj5E. (published: August 2020).

[124] Anush Manukyan. *The learning process of a hexacopter UAV for performing navigation and stable hovering task*. URL: https://youtu.be/rFr4UTf4qTc. (published: August 2020).

[125] Anush Manukyan et al. "Deep Reinforcement Learning-based Continuous Control for Multicopter Systems". In: *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*. IEEE. 2019, pp. 1876–1881.

[126] Anush Manukyan et al. "Deep Reinforcement Learning-based Continuous Control of a Flying Manipulator". In:

[127] Anush Manukyan et al. *Real time degradation identification of UAV using machine learning techniques*. URL: https://www.dropbox.com/s/tfc6jjvrzqy5l2d/UAV_Good_Worst_Flight.mp4?dl=0.

[128] MatLab. *MatLab*. URL: https://en.wikipedia.org/wiki/MATLAB.

[129] Blakeley B McShane. "Machine learning methods with time series dependence". In: (2010).

[130] R Scott Mitchell. "The application of machine learning techniques to time-series data". In: (1995).

[131] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[132] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[133] Bryan S Morse, Cameron H Engh, and Michael A Goodrich. "UAV video coverage quality maps and prioritized indexing for wilderness search and rescue". In: *2010 5th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE. 2010, pp. 227–234.

[134] Mostafa Moussid, Adil Sayouti, and Hicham Medromi. "Dynamic modeling and control of a hexarotor using linear and nonlinear methods". In: *International Journal of applied information systems* 9.5 (2015), pp. 9–17.

[135] Giuseppe Muscio et al. "Experiments on coordinated motion of aerial robotic manipulators". In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 1224–1229.

[136] Ofir Nachum et al. "Data-efficient hierarchical reinforcement learning". In: *Advances in Neural Information Processing Systems*. 2018, pp. 3303–3313.

[137] Ofir Nachum et al. "Why Does Hierarchy (Sometimes) Work So Well in Reinforcement Learning?" In: *arXiv preprint arXiv:1909.10618* (2019).

[138] Andrew Y Ng, Daishi Harada, and Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping". In: *ICML*. Vol. 99. 1999, pp. 278–287.

[139] Andrew Y Ng, Stuart J Russell, et al. "Algorithms for inverse reinforcement learning." In: *Icml*. Vol. 1. 2000, p. 2.

[140] Andrew Y Ng et al. "Autonomous inverted helicopter flight via reinforcement learning". In: *Experimental robotics IX*. Springer, 2006, pp. 363–372.

[141] Anibal Ollero and Bruno Siciliano. *Aerial Robotic Manipulation*. Springer, 2019.

[142] OpenAI. *OpenAI Gym toolkit*. URL: https://gym.openai.com.

[143] A Pahsa et al. "Integrating navigation & surveillance of Unmanned Air Vehicles into the civilian national airspaces by using ADS-B applications". In: *2011 Integrated Communications, Navigation, and Surveillance Conference Proceedings*. IEEE. 2011, J7–1.

[144] Ivana Palunko et al. "A reinforcement learning approach towards autonomous suspended load manipulation using aerial robots". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 4896–4901.

[145] Jooyoung Park, Jongho Kim, and Daesung Kang. "An RLS-based natural actor-critic algorithm for locomotion of a two-linked robot arm". In: *International Conference on Computational and Information Science*. Springer. 2005, pp. 65–72.

[146] Kui-Hong Park, Yong-Jae Kim, and Jong-Hwan Kim. "Modified uni-vector field navigation and modular q-learning for soccer robots". In: *Proceedings of the 32nd ISR (International Symposium on Robotics)*. Vol. 19. 2001, p. 21.

[147] parrot. *AR.Drone 2.0*. URL: https://www.parrot.com/global/drones/parrot-ardrone-20-elite-edition.

[148] Hector Perez-Leon et al. "An Aerial Robot Path Follower Based on the 'Carrot Chasing'Algorithm". In: *Iberian Robotics conference*. Springer. 2019, pp. 37–47.

[149] Theodore J Perkins and Mark D Pendrith. "On the existence of fixed points for Q-learning and Sarsa in partially observable domains". In: *ICML*. 2002, pp. 490–497.

[150] Jan Peters and Stefan Schaal. "Natural actor-critic". In: *Neurocomputing* 71.7-9 (2008), pp. 1180–1190.

[151] Jan Peters and Stefan Schaal. "Natural actor-critic". In: *Neurocomputing* 71.7-9 (2008), pp. 1180–1190.

[152] Jan Peters and Stefan Schaal. "Policy gradient methods for robotics". In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2006, pp. 2219–2225.

[153] Jan Peters and Stefan Schaal. "Policy gradient methods for robotics". In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2006, pp. 2219–2225.

[154] Jan Peters and Stefan Schaal. "Reinforcement learning of motor skills with policy gradients". In: *Neural networks* 21.4 (2008), pp. 682–697.

[155] Thomas Petsche et al. "A neural network autoassociator for induction motor failure prediction". In: *Advances in neural information processing systems*. 1996, pp. 924–930.

[156] Huy X Pham et al. "Autonomous uav navigation using reinforcement learning". In: *arXiv preprint arXiv:1801.05086* (2018).

[157] Huy X Pham et al. "Autonomous uav navigation using reinforcement learning". In: *arXiv preprint arXiv:1801.05086* (2018).

[158] Huy Xuan Pham et al. "Cooperative and distributed reinforcement learning of drones for field coverage". In: *arXiv preprint arXiv:1803.07250* (2018).

[159] Riccardo Polvara et al. "Toward end-to-end control for UAV autonomous landing via deep reinforcement learning". In: *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2018, pp. 115–123.

[160] Ivaylo Popov et al. "Data-efficient deep reinforcement learning for dexterous manipulation". In: *arXiv preprint arXiv:1704.03073* (2017).

[161] Doina Precup. "Temporal abstraction in reinforcement learning." In: (2001).

[162] Deirdre Quillen et al. "Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 6284–6291.

[163] Ahmed H Qureshi et al. "Composing Task-Agnostic Policies with Deep Reinforcement Learning". In: *arXiv preprint arXiv:1905.10681* (2019).

[164] Mehdi Rahimi et al. "A comparison of various approaches to reinforcement learning algorithms for multi-robot box pushing". In: *International Conference on Engineering Research and Applications*. Springer. 2018, pp. 16–30.

[165] Thanawin Rakthanmanon et al. "Searching and mining trillions of time series subsequences under dynamic time warping". In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2012, pp. 262–270.

[166] Ellis Ratner, Dylan Hadfield-Menell, and Anca D Dragan. "Simplifying reward design through divide-and-conquer". In: *arXiv preprint arXiv:1806.02501* (2018).

[167] Esteban Real et al. "Large-scale evolution of image classifiers". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 2902–2911.

[168] Research and Markets. *Europe Machine Learning Market 2019-2027*. URL: https://www.researchandmarkets.com/reports/4856306/europe-machine-learning-market-2019-2027#pos-5. (accessed: December 2019).

[169] Research and Markets. *Roundup of Machine Learning Forecasts and Market Estimates, 2020*. URL: https://www.forbes.com/sites/louiscolumbus/2020/01/19/roundup-of-machine-learning-forecasts-and-market-estimates-2020/. (accessed: January 2020).

[170] ROS. *Robot Operating System*. URL: http://www.ros.org.

[171] Pengkai Ru and Kamesh Subbarao. "Nonlinear model predictive control for unmanned aerial vehicles". In: *Aerospace* 4.2 (2017), p. 31.

[172] Andrei A Rusu et al. "Sim-to-real robot learning from pixels with progressive nets". In: *arXiv preprint arXiv:1610.04286* (2016).

[173] Atheer L Salih et al. "Flight PID controller design for a UAV quadrotor". In: *Scientific research and essays* 5.23 (2010), pp. 3660–3667.

[174] Andriy Sarabakha et al. "Novel Levenberg–Marquardt based learning algorithm for unmanned aerial vehicles". In: *Information Sciences* 417 (2017), pp. 361–380.

[175] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[176] John Schulman et al. "Trust region policy optimization". In: *International conference on machine learning*. 2015, pp. 1889–1897.

[177] Shital Shah et al. "Airsim: High-fidelity visual and physical simulation for autonomous vehicles". In: *Field and service robotics*. Springer. 2018, pp. 621–635.

[178] Akanksha Rai Sharma and Pranav Kaushik. "Literature survey of statistical, deep and reinforcement learning in natural language processing". In: *2017 International Conference on Computing, Communication and Automation (ICCCA)*. IEEE. 2017, pp. 350–354.

[179] Amritash Shekhar and Abhijeet Sharma. "Review of Model Reference Adaptive Control". In: *2018 International Conference on Information, Communication, Engineering and Technology (ICICET)*. IEEE. 2018, pp. 1–5.

[180] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer, 2016.

[181] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[182] David Silver et al. "Deterministic policy gradient algorithms". In: 2014.

[183] Panda 3D Simulator. *Panda 3D Simulator*. URL: https://en.wikipedia.org/wiki/Panda3D.

[184] Leena Singh and James Fuller. "Trajectory generation for a UAV in urban terrain, using nonlinear MPC". In: *Proceedings of the 2001 American Control Conference.(Cat. No. 01CH37148)*. Vol. 3. IEEE. 2001, pp. 2301–2308.

[185] Ewoud JJ Smeur, Qiping Chu, and Guido CHE de Croon. "Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles". In: *Journal of Guidance, Control, and Dynamics* 38.12 (2016), pp. 450–461.

[186] Ondřej Spinka, Ondřej Holub, and Zdenek Hanzálek. "Low-cost reconfigurable control system for small UAVs". In: *IEEE Transactions on Industrial Electronics* 58.3 (2009), pp. 880–889.

[187] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[188] Cyrill Stachniss, John J Leonard, and Sebastian Thrun. "Simultaneous localization and mapping". In: *Springer Handbook of Robotics*. Springer, 2016, pp. 1153–1176.

[189] Alejandro Suarez et al. "Anthropomorphic, compliant and lightweight dual arm system for aerial manipulation". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 992–997.

[190] Takuya Sugimoto and Manabu Gouko. "Acquisition of hovering by actual UAV using reinforcement learning". In: *2016 3rd International Conference on Information Science and Control Engineering (ICISCE)*. IEEE. 2016, pp. 148–152.

[191] Takuya Sugimoto and Manabu Gouko. "Acquisition of hovering by actual UAV using reinforcement learning". In: *2016 3rd International Conference on Information Science and Control Engineering (ICISCE)*. IEEE. 2016, pp. 148–152.

[192] Ilya Sutskever, James Martens, and Geoffrey E Hinton. "Generating text with recurrent neural networks". In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011, pp. 1017–1024.

[193] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[194] Richard S Sutton and Andrew G Barto. "Reinforcement learning: an introduction Cambridge". In: *MA: MIT Press.[Google Scholar]* (1998).

[195] Richard S Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial intelligence* 112.1-2 (1999), pp. 181–211.

[196] Richard S Sutton et al. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems*. 2000, pp. 1057–1063.

[197] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. "Introduction to multilayer feed-forward neural networks". In: *Chemometrics and intelligent laboratory systems* 39.1 (1997), pp. 43–62.

[198] Kai Sheng Tai, Richard Socher, and Christopher D Manning. "Improved semantic representations from tree-structured long short-term memory networks". In: *arXiv preprint arXiv:1503.00075* (2015).

[199] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. "Reinforcement learning of motor skills in high dimensions: A path integral approach". In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 2397–2403.

[200] Michael E Tipping. "Sparse Bayesian learning and the relevance vector machine". In: *Journal of machine learning research* 1.Jun (2001), pp. 211–244.

[201] Claude F Touzet. "Distributed lazy Q-learning for cooperative mobile robots". In: *International Journal of Advanced Robotic Systems* 1.1 (2004), p. 1.

[202] Kimon P Valavanis and George J Vachtsevanos. *Handbook of unmanned aerial vehicles*. Vol. 1. Springer, 2015.

[203] Hado Van Hasselt. "Reinforcement learning in continuous state and action spaces". In: *Reinforcement learning*. Springer, 2012, pp. 207–251.

[204] Chao Wang et al. "Autonomous navigation of UAVs in large-scale complex environments: A deep reinforcement learning approach". In: *IEEE Transactions on Vehicular Technology* 68.3 (2019), pp. 2124–2136.

[205] Steven Lake Waslander et al. "Multi-agent quadrotor testbed control design: Integral sliding mode vs. reinforcement learning". In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2005, pp. 3712–3717.

[206] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[207] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[208] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". In: (1989).

[209] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". In: (1989).

[210] MJ Willis. "Proportional-integral-derivative control". In: *Dept. of Chemical and Process Engineering University of Newcastle* (1999).

[211] Xindong Wu et al. "Top 10 algorithms in data mining". In: *Knowledge and information systems* 14.1 (2008), pp. 1–37.

[212] DING Xilun et al. "A review of aerial manipulation of small-scale rotorcraft unmanned robotic systems". In: *Chinese Journal of Aeronautics* 32.1 (2019), pp. 200–214.

[213] DING Xilun et al. "A review of aerial manipulation of small-scale rotorcraft unmanned robotic systems". In: *Chinese Journal of Aeronautics* 32.1 (2019), pp. 200–214.

[214] Peng Yang et al. "A tighter lower bound estimate for dynamic time warping". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, pp. 8525–8529.

[215] Zhao Yijing et al. "Q learning algorithm based UAV path learning and obstacle avoidence approach". In: *2017 36th Chinese Control Conference (CCC)*. IEEE. 2017, pp. 3397–3402.

[216] Rakan A Zahawi et al. "Using lightweight unmanned aerial vehicles to monitor tropical forest recovery". In: *Biological Conservation* 186 (2015), pp. 287–295.

[217] Iker Zamora et al. "Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo". In: *arXiv preprint arXiv:1608.05742* (2016).

[218] Guangyu Zhang et al. "Aerial Grasping of an Object in the Strong Wind: Robust Control of an Aerial Manipulator". In: *Applied Sciences* 9.11 (2019), p. 2230.

[219] Guangyu Zhang et al. "Aerial Grasping of an Object in the Strong Wind: Robust Control of an Aerial Manipulator". In: *Applied Sciences* 9.11 (2019), p. 2230.

[220] Tianhao Zhang et al. "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search". In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 528–535.

[221] Andrew Zulu and Samuel John. "A review of control algorithms for autonomous quadrotors". In: *arXiv preprint arXiv:1602.02622* (2016).