Noname manuscript No.
(will be inserted by the editor)

A Deep Reinforcement Learning Strategy for UAV Autonomous Landing on a Moving Platform

Alejandro Rodriguez-Ramos · Carlos Sampedro · Hriday Bavle · Paloma de la Puente · Pascual Campoy

Received: date / Accepted: date

Abstract The use of multi-rotor UAVs in industrial and civil applications has been extensively encouraged by the rapid innovation in all the technologies involved. In particular, deep learning techniques for motion control have recently taken a major qualitative step, since the successful application of Deep Q-Learning to the continuous action domain in Atari-like games. Based on these ideas, Deep Deterministic Policy Gradients (DDPG) algorithm was able to provide outstanding results with continuous state and action domains, which are a requirement in most of the robotics-related tasks. In this context, the research community is lacking the integration of realistic simulation systems with the reinforcement learning paradigm, enabling the application of deep reinforcement learning algorithms to the robotics field.

In this paper, a versatile Gazebo-based reinforcement learning framework has been established and validated with a continuous UAV landing task. The UAV landing maneuver on a moving platform has been solved by means of the novel DDPG algorithm and our reinforcement learning framework. Several experiments have been performed in a wide variety of conditions for both simulated and real flights, demonstrating the generality of the approach. As an indirect result, a powerful work flow for robotics has been validated, where robots can learn in simulation and perform properly in real operation environments. To the best of the authors knowledge, this is the first work that addresses the continuous UAV landing maneuver on a moving platform by means of a state-of-the-art deep reinforcement learning algorithm, trained in simulation and tested in real flights.

Keywords deep reinforcement learning · UAV · autonomous landing · continuous control

Alejandro Rodriguez-Ramos
Calle de Jose Gutierrez Abascal, 2, 28006 Madrid
E-mail: alejandro.rramos@upm.es

1 Introduction

In recent years, considerable research has been conducted regarding the design, development, and operation of autonomous Unmanned Aerial Vehicles (UAVs). Multi-rotor UAVs are potentially useful in a wide variety of scenarios, from natural disasters (conflagrations, earthquakes, etc.) to automation in a broad range of industries (energy, manufacture, construction, etc.). Nevertheless, these fields of application impose enormous constraints for normal operation tasks such as taking-off, navigation, object detection, environment interaction or landing. Thus, due to their level of complexity, researchers have approached these tasks separately in a diverse set of research lines [18,24, 35,23,38] and international competitions [34,12], e.g. International Micro-Air Vehicles competition (IMAV)[1].

In this context, for the last decade, the research community has focused on providing multi-rotor UAVs with the required level of autonomy for every of the previously stated tasks, from navigation in unstructured environments [7] to landing on moving platforms. In particular, the landing maneuver plays a significant role for long-term operations due to the UAV limitation of rapid battery discharge [33]. Moreover, in multi-robot operations, such as in combination with Unmanned Ground Robots (UGVs), landing and/or target following become necessary [9]. Indeed, these facts can be limiting for providing multi-rotor UAVs with the required level of autonomy in long-term missions. Due to this reason, the landing maneuver on a moving platform has been on the focus of several research lines for years [14,3,6,42,5,20,46,15].

Traditionally, the landing maneuver on a moving platform has been approached by means of a wide variety of techniques, which are able to solve the problem in an analytic manner and to perform properly in some specific conditions. Most of these strategies are mainly based on perception and relative pose estimation [6,42,5], as well as trajectory optimization and control [29,1, 20,46,15].

Nevertheless, classical techniques have their limitations, in terms of model design, non-linearities approximation, disturbances rejection and efficiency of computation. In this context, machine learning techniques have proven to increasingly overcome most of these limitations, having generated high expectations in the research community since 1971, when Ivakhnenko [19] trained a 8-layer neural network using the Group Method of Data Handling (GMDH) algorithm. Nowadays, machine learning has evolved to more complex techniques, such as deep learning strategies which are capable of generalizing from large datasets of raw data information. Deep learning has opened up important research and application fields in the context of unsupervised feature extraction, where Convolutional Neural Networks (CNNs) were able to provide outstanding results in comparison to traditional computer vision techniques [26].

In the context of machine learning (and reinforcement learning) for continuous control, there are uprising problems to cope with, such as divergence

[1] `https://imavs.org`

of learning, temporal correlation of data, data efficiency or continuous nature of inputs and outputs. These issues have been limiting machine learning and reinforcement learning strategies for continuous control over the last years. However, recent advances in the reinforcement learning field, such as DeepMind Technologies Deep Q-Network (DQN) [31], have unveiled a new set of possibilities to solve complex human-level problems by means of novel deep reinforcement learning strategies. The key advances of DQN were the inclusion of an experience replay buffer (to overcome data correlation), and a different approach for the target Q-Network, whose weights change with the update of the main Q-Network in order to break the correlation between both networks (in contrast with the targets used for traditional supervised learning, which are fixed before learning begins) [31]. The state of the DQN algorithm is the raw image and it has been widely tested with Atari games. DQN established the base for a novel line of deep reinforcement learning solutions, but it was not designed for continuous states, which are deeply related to robotic control problems.

Based on the key improvements of DQN and the actor-critic paradigm established by Richard S. Sutton and Andrew G. Barto in their renowned reinforcement learning book [43], Lillicrap et al. proposed Deep Deterministic Policy Gradients (DDPG) [28] as an algorithm to solve continuous control problems by integrating neural networks in the reinforcement learning paradigm. DDPG is able to perform remarkably well with low dimensional continuous states and actions, but is also capable of learning from raw pixels [28].

In this work, the novel deep reinforcement learning algorithm (DDPG) has been utilised to solve a complex high level task, such as UAV autonomous landing on a moving platform. This task has been solved in simulation and real flights by means of a Gazebo-based reinforcement learning framework. The training phase has been carried out in Gazebo[2] [48] and RotorS simulator [13], which provide realistic simulations that help to a quick transition to real flight scenarios. The testing phase has been performed in both simulated and real flights.

1.1 Related work

The problem of UAV autonomous landing on both static and moving platforms is of utmost importance for real world applications [33, 9]. Given the complexity of the challenge, a number of previous works focus mostly on specific solutions for components such as perception and relative pose estimation [6, 42, 5] or such as trajectory optimization and control [29, 1, 20, 46, 15]. Other research lines explore coupled methods mostly related to Image-Based Visual Servoing [27] and, in this direction, novel advanced algorithms which also incorporate constant force disturbance estimation have been proposed [39].

[2] `http://gazebosim.org`

Regarding the control maneuvers when the relative state of the vehicles is assumed to be known, popular techniques include different kinds of guidance and rendezvous laws [15,20] which sometimes are augmented with velocity controllers for a faster approaching phase [3]. When a desired meeting point is obtained, incorporating feedforward inputs allows for a faster response against track following errors [29] and the determination of optimal rendezvous trajectories can also take wind disturbances into account [1]. PID controllers are the preferred option for aggressive landing from relatively short distances [47, 5,3], while an adaptive control schema presents enhanced robustness [18,24]. A discrete-time non-linear model predictive controller which optimizes both the trajectories and the landing time was developed to address the difficult problem of landing on top of moving inclined platforms [46].

Even if only tested on static platform landing tasks, innovative bio-inspired strategies have proven to perform well in the real world, employing a time-to-contact (TTC) indicator [22]. Intelligent control and machine learning based methods are very promising too, since they provide the ability to deal with different system dynamics in different environments and landing circumstances [4]. Recent contributions have proposed neural network backpropagation controllers [4] for landing on top of a static platform and classical discrete reinforcement learning approaches have also been used in the literature, such as the approach proposed by Shaker et al. [40], where an LSPI algorithm was used to land on top of a static platform. Both state and actions were part of a discrete space and the main sensor to estimate the state was a camera. The UAV was able to perform a landing maneuver on a static platform in a simulated environment.

The previously mentioned novel reinforcement learning methodologies are strongly related to deep learning strategies, since their theory is intrinsically linked. Concerning deep learning for UAV indoor navigation tasks, recent advances have driven to a successful application of CNNs in order to map images to high-level behaviour directives (e.g. turn left, turn right, rotate left, rotate right) [35,23]. In [35], the Q function is estimated through a CNN, which is trained in simulation and successfully tested in real experiments. In [23], discrete actions are directly mapped from raw images. In all stated methods, the learned model is run offboard, usually taking advantage of a GPU in an external laptop.

In [16], a Deep Neural Network (DNN) model was trained to map image to action probabilities (turn left, go straight or turn right) with a final softmax layer, and tested onboard by means of an Odroid-U3 processor. The performance is later compared to two automated methods (SVM and a method in [38]) and two human observers.

On the other hand, deep learning for low-level motion control is challenging, since dealing with continuous and multi-variable action spaces can become an intractable problem. Nevertheless, some recent advances have proposed novel methods to learn low-level control policies from imperfect sensor data in simulation [49,21]. In [49], a Model Predictive Controller (MPC) was used to generate data at training time in order to train a DNN policy, which was

allowed to access only raw observations from the UAV onboard sensors. In testing time, the UAV was able to follow an obstacle-free trajectory even in unknown situations. In [21], the well-known Inception v3 model (pre-trained CNN) was adapted in order to enable the final layer to provide six action nodes (three transitions and three orientations). After re-training, the UAV managed to cross a room filled with a few obstacles in random locations.

On the side of deep reinforcement learning, some recent algorithms are able to perform slightly better than DDPG, in terms of training time and for low-dimensional continuous tasks. In [17], Normalized Advantage Functions (NAF) or continuous deep Q-learning algorithm is able to solve continuous problems in simulation, by the use of a neural network that separately outputs a value function $V(x)$ and an advantage term $A(x, u)$ [17]. This representation allows to simplify more standard actor-critic style algorithms, while preserving the benefits of non-linear value function approximation [17]. In [30], several agents (from 1 to 16) are run in parallel threads, enabling the possibility of stable training of neural networks with both value-based and policy-based methods, off-policy as well as on-policy methods, and in discrete as well as continuous domains. Also, Asynchronous Advantage Actor-Critic (A3C) shows that stable online Q-learning is possible without experience replay [30]. Both [17] and [30] have been tested in simulated environments, such as MuJoCo [44] and/or TORCs [11].

Finally, concerning the framework for training and testing novel deep reinforcement learning algorithms for robotics, recent developments point to extend the OpenAI Gym[3] reinforcement learning training/test bench to a widely-used robotics simulator, such as Gazebo simulator. In [48], a complete open source test bench is released, with simulation frequency up to real time and meant for an specific model of UAV and UGV.

1.2 Contributions

Our proposed method differs from previous work in the following aspects: (i) A Gazebo-based reinforcement learning framework has been established. This framework is versatile-enough to be adapted to other types of algorithms, environments and robots. (ii) A novel deep reinforcement learning algorithm (DDPG) has been adapted and integrated into our Gazebo-based simulation framework. (iii) The landing maneuver on a moving platform has been solved by means of a deep reinforcement learning algorithm, in both simulated and real flights.

Please note that we address the full problem, with continuous state and actions spaces. Also, as an indirect result, we have demonstrated the feasibility of a powerful work flow, where robots can be trained in simulation and tested in real operation environments. To the best of the authors knowledge, this is the first work that addresses the UAV landing maneuver on top of a moving

[3] Open test bench for reinforcement learning algorithms: `https://gym.openai.com`

platform by means of a state-of-art deep reinforcement learning algorithm, trained in simulation and tested in real flights.

The remainder of the paper is organized as follows: Section 2 presents a brief introduction on the reinforcement learning theory and a short explanation on the basics of DDPG algorithm. Section 3 details the presentation and description of our Gazebo-based reinforcement learning framework and the design of the experiment which meets all the constraints required in the deep reinforcement learning paradigm for autonomous UAV landing on a moving platform. Section 4 presents the simulated and real-flight experiment results. Finally, Section 5 provides conclusions and future work optimizations and research lines.

2 Background

In reinforcement learning, an agent is defined to interact with an environment, seeking to find the best action for each state at any time step. The agent must balance exploration and exploitation of the state space in order to find the optimal policy which maximizes the accumulated reward from the interaction with the environment. In this context, an agent modifies its behaviour or policy with the awareness of the states, actions taken and rewards for every time step. Indeed, reinforcement learning involves an optimization process throughout the whole state space, in order to maximize the accumulated reward. Robotic problems are often task-based with temporal structure. These type of problems are suitable to be solved by means of a reinforcement learning framework [25].

The standard reinforcement learning theory states that an agent is able to obtain a policy, which maps every state $s \in \mathbb{S}$ to an action $a \in \mathbb{A}$, where $\mathbb{S}$ is the state space (possible states of the agent in the environment) and $\mathbb{A}$ is the finite action space. The inner dynamics of the agent are represented by the transition probability model $p(s_{t+1}|s_t, a_t)$ at time t. The policy can be stochastic $\pi(a|s)$, with a probability associated to each possible action, or deterministic $\pi(s)$. In each time step, the policy determines the action to be chosen and the reward $r(s_t, a_t)$ is observed from the environment. The goal of the agent is to maximize the accumulated discounted reward $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$ from a state at time t to time T ($T = \infty$ for infinite horizon problems) [43]. The discount factor γ is defined to allocate different weights for the future rewards.

For a specific policy π, the value function V^π in Eq. 1 is a representation of the expectation of the accumulated discounted reward R_t for each state $s \in \mathbb{S}$ (assuming a deterministic policy $\pi(s_t)$).

$$V^\pi(s_t) = \mathbb{E}[R_t|s_t, a_t = \pi(s_t)] \tag{1}$$

An equivalent of the value function is represented by the action-value function Q^π in Eq. (2) for every action-state pair (s_t, a_t).

$$Q^\pi(s_t, a_t) = r(s_t, a_t) + \gamma \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) V^\pi(s_{t+1}) \tag{2}$$

The optimal policy π^* shall be the one which maximizes the value function (or equivalently the action-value function), as in Eq. (3).

$$\pi^* = \arg\max_{\pi} V^{\pi}(s_t) = \arg\max_{a_t} Q^*(s_t, a_t) \tag{3}$$

A general problem in real robotic applications is that the state and action spaces are often continuous spaces. A continuous state and/or action space can make the optimization problem intractable, due to the overwhelming set of different states and/or actions. Reinforcement learning methods, as a general framework for representation, are enhanced through deep learning to aid the design for feature representation, which is known as deep reinforcement learning.

In the context of state-of-the-art deep reinforcement learning algorithms, DDPG represents one successful application of neural networks to the reinforcement learning paradigm, and it is able to solve continuous control problems. As previously stated, DDPG [28] is a policy-based deep reinforcement learning algorithm designed to work with both continuous state and actions spaces. Policy-based reinforcement learning methods aim towards directly searching the optimal policy π^*, which provides a feasible framework for continuous control. If the target policy π^* is a deterministic policy μ, the Q function (see Eq. 4) can be learned off-policy, using transitions (from an environment E) which are generated from a different stochastic behaviour policy β [28].

$$Q(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))] \tag{4}$$

A function approximator, parametrized by θ^Q, is considered in DDPG to approximate the Q function. It is optimized by minimizing the loss $L(\theta^Q)$ of Eq. 5.

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^{\beta}, a_t \sim \beta, r_t \sim E}[(Q(s_t, a_t|\theta^Q) - y_t)^2] \tag{5}$$

where

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q) \tag{6}$$

The key changes for this large non-linear approximators to converge (in discrete spaces) were: the use of a *replay buffer*, and a separate target network for calculating y_t, as firstly proven by DQN [31]. In order to deal with large continuous state and action spaces, DDPG adapted the actor-critic paradigm introduced in [41], with two neural networks to approximate a greedy deterministic policy (actor) and the Q function (critic). DDPG method learns with an average factor of 20 times fewer experiences steps than DQN [28].

The actor network is updated by following and applying the chain rule to the expected return from the start distribution J with respect to the actor parameters (see Eq. 7).

$$\nabla_{\theta^{\mu}} J \approx \mathbb{E}_{s_t \sim \rho^{\beta}}[\nabla_{\theta^{\mu}} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^{\mu})}] \tag{7}$$

An advantage of the off-policy methods is that exploration can be treated independently from learning. In this case, exploration is carried out throughout autocorrelated Ornstein-Uhlenbeck exploration noise [45].

3 Proposed approach

In this section, a complete explanation of our Gazebo-based reinforcement learning framework is included. Furthermore, the formulation of the UAV landing on a moving platform problem, within the reinforcement learning paradigm, is detailed.

3.1 Reinforcement learning simulation framework

Traditionally, reinforcement learning algorithms have been tested and validated by means of generic tools for simulation (e.g. Matlab, Simulink). The performance of other classic reinforcement learning algorithms was not powerful enough to overcome high-level and complex problems, neither in the context of continuous states and actions spaces, or working with continuous physics in realistic simulators (*e.g.* Gazebo simulator). Nevertheless, since there has been an increasing improvement in the performance of this kind of algorithms, simulation systems have been following this complexity trend, allowing a feasible and stable training of reinforcement learning agents under realistic and continuous simulators.

OpenAI Gym is an open-source toolkit for developing and comparing reinforcement learning algorithms [8]. Currently, most of the available environments for testing in OpenAI Gym are 2D or Atari-like games, with a diverse range of complexity examples, but lacking more realistic environments for robotic simulation. In robotics, this type of training/testing environments are not directly available for the research community.

On the other hand, the Gazebo simulator is often a prime example of a versatile and realistic simulation system for robotics, normally aided by the well-known Robot Operating System (ROS) middleware [32]. Hence, taking advantage of the versatility of this broadly-used simulation system, we have designed and implemented a Gazebo-based reinforcement learning framework. The aim of this framework is to provide a predefined interface between the reinforcement learning algorithm, the environment interface and the Gazebo simulator (or other type of simulators). Also, it has been designed in such a way that it can be extended in the future to different types of reinforcement learning algorithms, environments and robots.

As previously stated, in reinforcement learning, an agent interacts with an environment and tries to maximize the accumulated reward in each time step. In our framework, the communication channel between the agent and the environment is implemented through commonly-used ROS communication tools, which provide a standardized way of communication (see Figure 1). Also, some

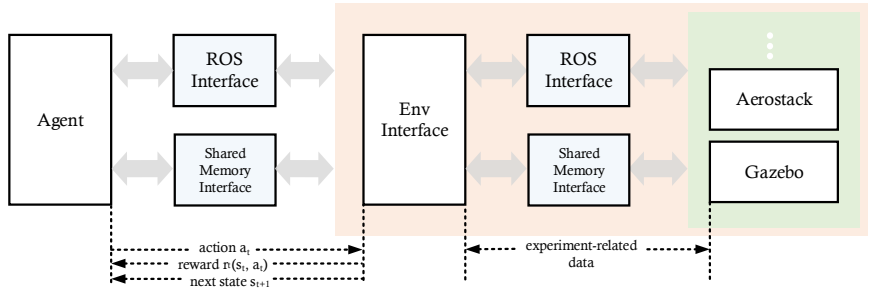


Fig. 1: Diagram of our reinforcement learning simulation framework. The theoretical concept of the reinforcement learning environment is highlighted in pale red. Simulation systems or multi-purpose architectures are highlighted in pale green.

reinforcement learning problems require the simulation time to be deliberately iterated (the clock is paused), in order to complete computationally-expensive training steps. In this scenario, where the Gazebo simulator clock is paused, communication via ROS tools is not possible, since ROS uses Gazebo clock to forward its messages through the network. In order to overcome these issues, an additional shared memory communication channel has been implemented, which can be used in case the Gazebo clock is paused and the simulation is being actively iterated. This functionality provides the developer with full control on the simulation time, allowing the interaction of a Gazebo plugin with Gazebo simulator and the shared memory interface, which can be a requirement in several real world problems.

Furthermore, in our framework, the agent component which represents the classical reinforcement learning agent, is capable of receiving experience vectors and rewards from the environment in order to find the optimum action to be taken in each time step. Our implementation of the agent is also capable of logging representative data which can be used to determine whether it is actually learning or whether the algorithm has otherwise diverged. On the other hand, the environment interface shown in Figure 1 represents a partial implementation of the classical reinforcement learning environment, since in this case, Gazebo simulator and Aerostack architecture [37] constitute an important part of the whole reinforcement learning environment. Aerostack is a system architecture and open-source multi-purpose software framework for autonomous multi-UAV operation. It has been used to enable the operation of the UAVs in both training and testing time, though its full explanation is out of the scope of this work. For further information, please refer to [37].

In this framework, the environment interface shown in Figure 1 implements an interface between Gazebo/Aerostack and the agent, being in charge of parsing all the incoming data, in order to adapt it to an intelligible structure which the agent can use. Furthermore, taking into consideration future

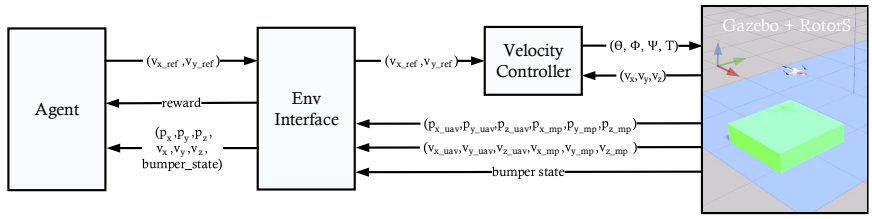


Fig. 2: Architecture of the experiment.

extensions of either agents, environments, robots or simulation systems, the framework has been designed in a versatile manner at a programming level. Since all the communication interfaces are standard and cross-language, both the agent and the environment interface can be implemented in a wide variety of programming languages, such as C++, Python, or Java.

Finally, our framework is designed to be used with Gazebo, but it can be adapted to any other simulation systems (as well as simulated robots), due to the standard nature of its communications. Also, the simulation time can be speeded up or slowed down, in order to reduce training times and to adapt the simulation to computationally-expensive experiments, respectively.

3.2 Reinforcement learning based formulation

In the context of reinforcement learning, the formulation of the experiment can be decisive for the algorithm to converge, since there are an increasing number of possible designs which ideally would lead to the same result. In practice, the formulation of the state and action spaces, as well as the design of the reward function, determines the speed of convergence and even the possibility of divergence of the reinforcement learning algorithm. We have designed the state, action and reward function in a way that it minimizes information passed to the agent, speeds up learning and avoids learning divergence.

As previously stated, a reinforcement learning experiment is defined by the state space $s \in \mathbb{S}$, the action space $a \in \mathbb{A}$ and the reward function r. In our proposed approach, the state space $\mathbb{S}$ is defined by Eq. 8.

$$\mathbb{S} = \{p_x, p_y, p_z, v_x, v_y, C\} \tag{8}$$

Where p_x, p_y and p_z are the positions of the UAV with respect to the Moving Platform (MP) in x, y and z axes respectively at time t, v_x and v_y are the velocities of the UAV with respect to the MP in x and y axes respectively at time t and C is the binary state of a pressure sensor located on the top of the horizontal surface of the MP. All the sensory information is retrieved

from Gazebo simulator and parsed by the environment interface component, as shown in Figure 2. Regarding the action space $\mathbb{A}$, it is defined by Eq. 9.

$$\mathbb{A} = \{a_x, a_y\} \tag{9}$$

Where a_x and a_y are the reference velocities, input of the velocity controller (see Figure 2), in x and y axes at time t. In this paper, the velocity reference in the z axis has not been included in the action space. This is due to the fact that we are tackling a complex problem with continuous state and action spaces and the full behaviour is completely self-learned in simulation, by means of a deep reinforcement learning algorithm not previously tested on this type of robotic tasks. Hence, the inclusion of z axis has been left as future work since it involves a much higher order of complexity out of the scope of this study. Instead, a constant velocity reference is commanded in the z axis in each time step. This fact simplifies the action space, increasing the speed of convergence of the algorithm without losing generality of the approach. The resulting state and action spaces are a continuous 6-dimensional space and a continuous 2-dimensional space respectively, with normalized variables ranging from +1 to -1 values.

The reward function is one of the most important components in the reinforcement learning framework. A proper design of the reward function can lead to a faster convergence of the algorithm and a better performance at testing time. In our proposed approach, where the agent is meant to generate continuous control actions, the reward function shall be designed in such a way that it rewards smooth actions with respect to time. The resulting reward function r is defined by Eq. 10 and Eq. 11.

$$\begin{aligned} shaping_t = -100\sqrt{p_x^2 + p_y^2} - 10\sqrt{v_x^2 + v_y^2} - \sqrt{{a_x}^2 + {a_y}^2} \\ +10C(1 - |a_x|) + 10C(1 - |a_y|) \end{aligned} \tag{10}$$

$$r = shaping_t - shaping_{t-1} \tag{11}$$

As can be inferred from Eq. 10, the reward function explicitly differentiates between the importance of minimizing the position with respect to the MP, the velocity with respect to the MP and the generated actions (each variable is weighted by a different coefficient). Following this fashion, the agent is able to coarsely learn to minimize its position with respect to the MP and to subsequently optimize its behaviour in order to generate smoother velocity references, which leads to a less aggressive movement. Also, the C coefficient rewards the agent as soon as the UAV lands on the MP and the velocity references are decreased to their absolute minimum.

In addition, *shaping* is a popular method for speeding up reinforcement learning in general, and goal-directed exploration in particular [10]. It increases the speed of convergence of a reinforcement learning algorithm by transferring knowledge about the current progress on the task to be solved, in this case,

with respect to the previous state of the agent. Nevertheless, it requires significant design effort, and results in less autonomous agents. Also, it may alter the optimal solution, leading to unexpected final behaviour. In this work, a *shaping* method is applied in an non-invasive trend, by informing the agent about its instantaneous progress and avoiding instability and algorithm divergence.

Also, it is assumed that both the position and velocity of the UAV and the MP are available at training time (ground truth data). Nevertheless, as stated in this section, the agent is only aware of its position and velocity with respect to the MP, enabling this approach to work also in the absence of absolute positioning systems, such as Global Navigation Satellite Systems (GNSS). It has to be noted that even if the agent has been trained with ground truth data, it is capable of performing the landing maneuver with noisy simulated and real data, as shown in section 4.2.

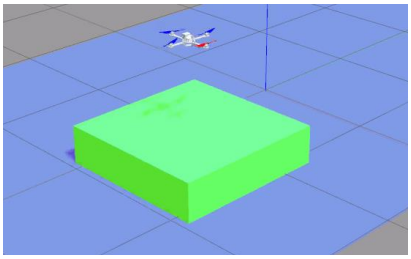
The training procedure is based on an adapted implementation of stated DDPG algorithm included in our framework. In our case, the actor and critic neural networks (and their corresponding target networks) are feed-forward neural networks with two hidden layers of 200 and 100 units each. The activation function of each unit of a hidden layer is a Recified Linear Unit (ReLU). The input and output layer dimensions of the actor network are based on the state and action dimensions (6 and 2 units), respectively (see Eq. (8) and (9)). The activation function of the output layer is a *tanh* function, bounded to the range of [-1,1]. The output layer of the critic network has one unit with a linear activation function in order to provide an estimation of the Q-function.

4 Experiments and results

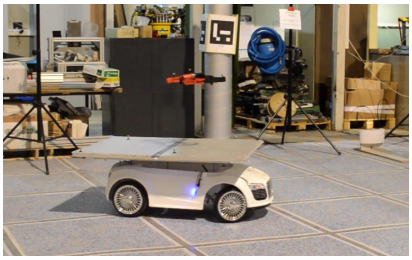
This section aims to provide a full explanation about the experiments designed and implemented to validate the whole training and testing pipeline. A detailed description of the training experiments in simulation, as well as the testing experiments in both simulation and real flights, is included. Results are shown and discussed, as well the hardware and software specifications which have been used to carry out the described experiments. A complete video of the whole set of training and testing experiments can be found in `https://vimeo.com/235350807`.

4.1 Experimental setup

In this section, the proposed experimental setup for simulated and real flights is described. The agent has been implemented in Python 2.7, due to the availability of the most common machine learning libraries. In this work, the Tensorflow library [2] has been used as the main basis of the algorithm and it can run on both Central Processing Unit (CPU) and Graphical Processing Unit (GPU). The GPU involved in the training phase was a Nvidia GeForce GTX970 and in the testing phase was a Nvidia GeForce GTX950M. In the case of the environment interface, it has been implemented in C++ (under the standard



(a) UAV and MP in the training and testing simulation environment.



(b) UAV and MP in the testing real environment.

Fig. 3: Simulation and real environment scenarios.

C++11), in order to take advantage of the benefits of our Aerostack architecture [37]. ROS Kinetic has been used as the communication framework. The operating system utilised for running the processes involved in both simulated and real flights is Ubuntu 16.04 LTS.

4.1.1 Simulated training and testing phases

A simulated environment has been created in Gazebo 7 for both simulated training and testing phases. A UAV model and a minimalistic model of a MP (see Figure 3a) have been included in an adapted version of RotorS simulator [13]. RotorS simulator emulates the autopilot and all the required sensors for a UAV to perform autonomous maneuvers, such as Inertial Measurement Unit (IMU), lidar and/or cameras (RGB or RGB-D). We have selected an AsTec Hummingbird as the UAV to perform the landing maneuver in simulation. A simulated pressure sensor has been included on the surface of the MP, in order to inform the agent about whether the UAV has properly landed or not.

The environment interface, which is in charge of parsing all the incoming data, receives position and velocity ground truth data from the Gazebo simulator and sends the velocity references to the Aerostack velocity controller via ROS topics. The state and reward is sent back to the agent via ROS services. The whole simulated training and testing phases have been carried out in a real-time Gazebo simulation, with an agent frequency of 20 Hz, an asynchronous environment interface, a velocity controller frequency of 30 Hz and Gazebo ground truth frequency of 100 Hz. The main differences between simulated training and testing phases are:

1. *Simulated training phase* The trajectory of the MP is linear and periodic with a maximum velocity of 1 m/s. The measured position and velocity of both the UAV and the MP are ground truth data with no noise. The permitted horizontal area for the UAV to fly is a rectangle of 3 m x 6 m (it has been heuristically set to provide the minimum feasible area which allows to learn the landing maneuver).

2. *Simulated testing phase* The trajectory of the MP can be linear and periodic with a maximum velocity of 1 m/s or non-linear and non-periodic with a maximum velocity of 1 m/s. The measured position and velocity of both the UAV and the MP are ground truth data with Gaussian noise ($\mu = 0$ and $\sigma = 1$) in every variable of the agent state. The permitted horizontal area for the UAV to fly is a rectangle of 5 m x 9 m.

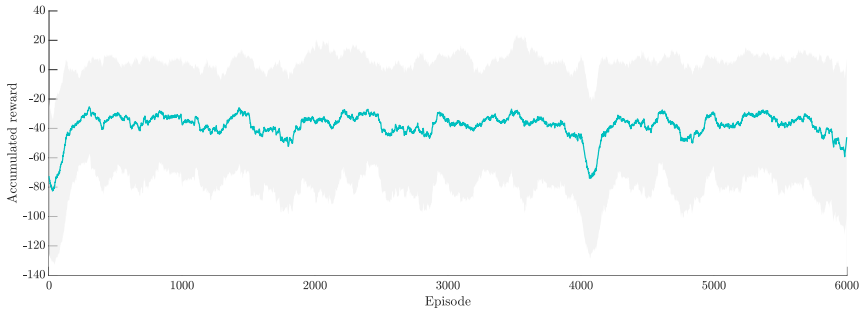
4.1.2 Real-flight testing phase

A replica of the simulated environment has been created for the real-flight testing phase. The MP, which was designed and built for previous works on autonomous landing [34,36], is able to move in linear periodic trajectories (with rails) and in arbitrary trajectories. The selected UAV platform was a Parrot Bebop[4] due to its small size, robust control and higher flight velocity (see Figure 3b). This UAV platform is provided with an on-board autopilot which can be commanded throughout a wireless WiFi channel. The remaining processes, such as the agent, the environment interface and the Aerostack components are run off-board. Tensorflow library calls are computed on a laptop GPU (Nvidia GTX950M). The rest of the required routines are computed on the CPU. The UAV and MP position and velocity information is provided by an Optitrack Motion Capture system (MoCap) which covers an area of approximately 4 m x 6 m. The frequency of the agent is 20 Hz, the environment interface is asynchronous, the velocity controller runs at 30 Hz and the motion capture system frequency is 100 Hz. The communication with the UAV is carried out trough WiFi at 2.4GHz.

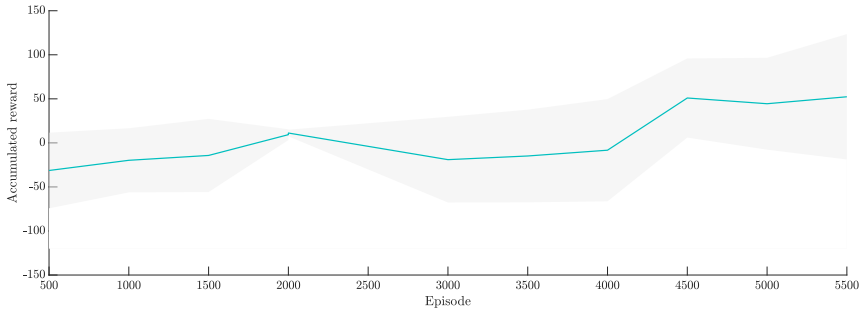
4.2 Results and discussion

In this work, the full landing maneuver has been trained in simulation throughout 4500 episodes (approximately 720k training steps over 10 hours). In this setup, an episode consists of a full landing trial on top of the MP and it is composed by a maximum of 900 training steps. As previously stated, the agent interacts with the environment every 0.05 s (at a frequency of 20 Hz), which corresponds to one training step. In each training step both actor and critic network weights are being optimized by means of Adam optimizer and with a learning rate of 10^{-4} and 10^{-3}, respectively. The selected minibatch size has been 64. In every episode, the UAV and the MP are initialized at a random position of the horizontal plane (x and y axes). The experiment finishes when the UAV touches the ground or the number of training steps exceeds the maximum per episode. Following this trend, the experiment is repeated in a wide variety of conditions to provide a complete range of experiences which the agent can learn from in order to maximize its accumulated reward over time.

[4] `http://global.parrot.com/mx/productos/bebop-drone/`



(a) Moving average and standard deviation of the accumulated reward per episode (moving window of 100 episodes) for the full simulated training phase.



(b) Moving average and standard deviation of the accumulated reward per episode (moving window of 3 episodes) for the full simulated testing phase (within the learning process).

Fig. 4: Average and standard deviation of the accumulated reward, for both simulated training and testing phases.

In Figure 4, the moving average and standard deviation of the accumulated reward for the full simulated training and testing phase are depicted. In the training phase, an Ornstein-Uhlenbeck exploration noise [45] is added to the actions provided by the agent. For this reason, the evolution of the accumulated reward of the training phase is not stable (only its average is stable), as shown in Figure 4a. Nevertheless, the evolution of the accumulated reward shows an improvement after the first 400 episodes, and subsequently a stabilization of the moving average and standard deviation for the rest of the training phase. Thus, the representation of the moving average and standard deviation of the accumulated reward in the training phase is helpful but does not show the real performance of the agent.

In Figure 4b, the moving average and standard deviation of the accumulated reward, for the simulated testing phase, are shown. Every 500 episodes of the training phase, 10 episodes of the testing phase have been carried out. In a test episode, the actions provided by the agent are directly mapped (without adding noise) to the environment as a way of determining its real performance. As shown in Figure 4b, the average accumulated reward increases until episode

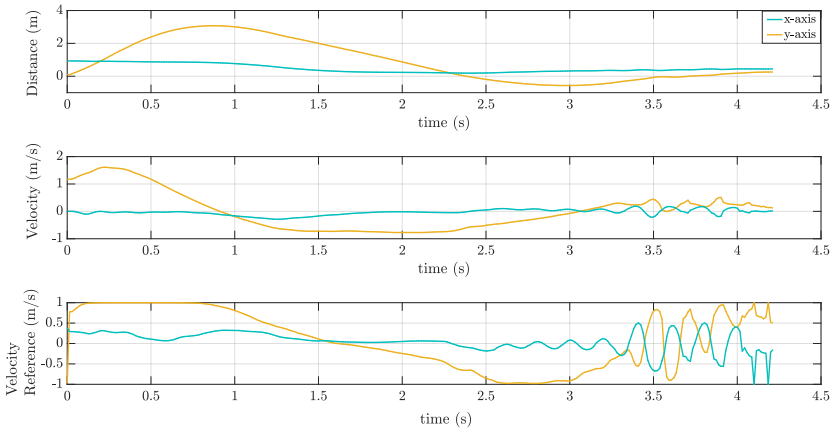
4500, where it remains approximately stable. As explained in the rest of the subsection, episode 4500 has been chosen as the first episode where the agent is fully trained, due to the stabilization of the average accumulated reward and the performance shown in the actions provided.

Every experiment included in this section corresponds to a successful landing trial on top of the MP. In Figure 5a, part of the state and action signals of episode 500 of the simulated testing phase are depicted. The most important components of the state, in terms of representation of the performance of the agent, are the position of the UAV with respect to the MP (p_x, p_y) and the velocity of the UAV with respect to the MP (v_x, v_y). These components allow to infer the performance in the x and y axes (parallel to the ground plane), since the performance in the z axis remained constant in these experiments (refer to Section 3.2). Also, the actions provided by the agent (a_x, a_y) are determinant to validate the performance for a real application. As shown in Figure 5a, the actions generated by the agent in the episode 500 are not optimum, since it keeps on generating velocity reference commands even when landed. Nevertheless, as suggested by Figure 4a, after episode 400, the behaviour of the agent is close to the optimum, being able to perform a full landing maneuver in most of the testing episodes, but with oscillating control actions when touching the MP.

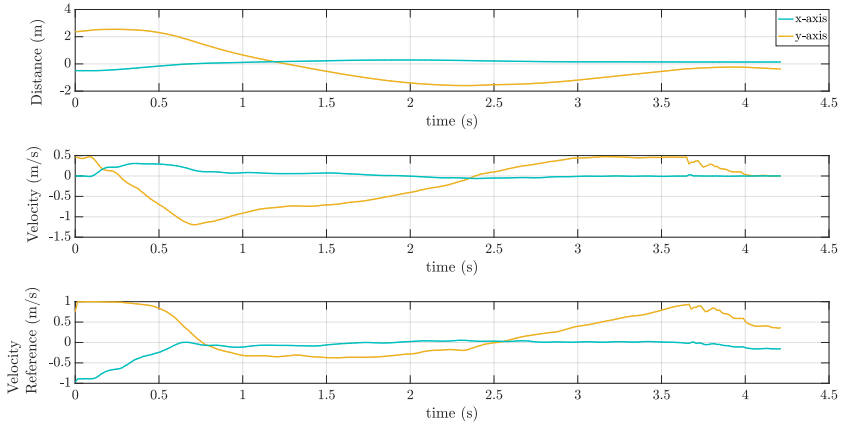
In Figure 5b, both position and velocity with respect to the MP converge to approximately zero (exact zero is not practically possible either in the context of a realistic simulation or in a real flight), and in a continuous and smooth trend, which is the desired behaviour. Furthermore, the velocity reference commands generated by the agent converge approximately to zero as well. Figure 5b represents the optimum performance of the agent in the absence of noise for the setup presented in this work. Note that due to the lack of friction of the simulated MP, the UAV is able to land on approximately the center of the MP, but it slightly slippers from this position over time and the agent has learned to compensate this effect (see supplementary video provided in the beginning of Section 4). Nevertheless, the UAV is still on top of the MP, which is considered a successful landing.

On the other hand, in order to test the capability of generalization and robustness of the DDPG algorithm in simulation, a test experiment with added noise has been performed. In this experiment, a Gaussian random variable ($\mu = 0$ and $\sigma = 1$) has been added to every component of the agent state, resulting in the plots of Figure 6. As shown, both the position and velocity of the UAV with respect to the MP are signals with a high level of gaussian noise, but the agent is still able to perform a proper landing maneuver (the position of the UAV with respect to the MP still converges to the origin). The velocity reference commands generated by the agent are notably noisy, which can be problematic in some other velocity control strategies (it may lead to over oscillation). However, in the context of a linear velocity controller, the final behaviour is more erratic but does not become unstable.

Additionally, in order to further validate our selected network from episode 4500, an extensive evaluation in two different scenarios has been performed



(a) Partial state and actions signals of episode 500 (test). (Top) Position of the UAV with respect to the MP, in x and y axes. The final position of the UAV with respect to the MP is 0.44 m and 0.42 m, in x and y axes respectively. (Middle) Velocity of the UAV with respect to the MP, in x and y axes. (Bottom) Velocity reference commands (actions) generated by the agent, in world coordinates and in x and y axes.



(b) Partial state and actions signals of episode 4500 (test). (Top) Position of the UAV with respect to the MP, in x and y axes. The final position of the UAV with respect to the MP is 0.14 m and -0.37 m, in x and y axes respectively. (Middle) Velocity of the UAV with respect to the MP, in x and y axes. (Bottom) Velocity reference commands (actions) generated by the agent, in world coordinates and in x and y axes.

Fig. 5: Partial state and actions signals of two test episodes (simulated testing phase).

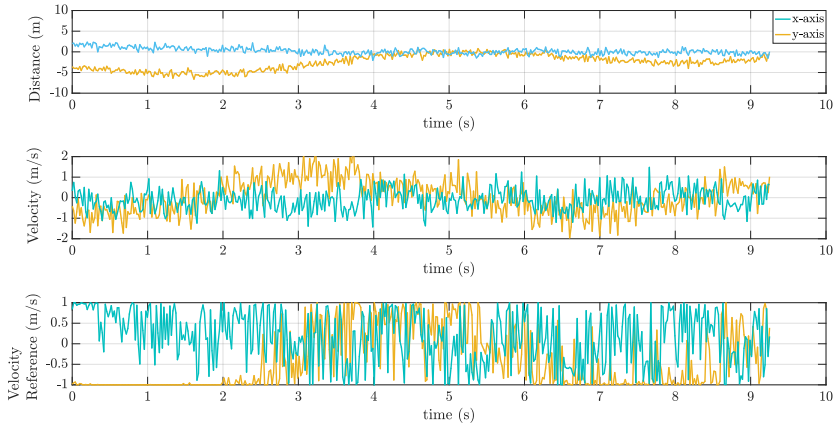


Fig. 6: Partial state and actions signals of episode 4500 (simulated testing phase). (Top) Position of the UAV with respect to the MP with Gaussian noise, in x and y axes. The final position of the UAV with respect to the MP is 0.15 m and 0.42 m, in x and y axes respectively. (Middle) Velocity of the UAV with respect to the MP with Gaussian noise, in x and y axes. (Bottom) Velocity reference commands (actions) generated by the agent, in world coordinates and in x and y axes.

Table 1: Mean and standard deviation metrics over 150 landing trials in two different simulated scenarios (for our selected actor network of episode 4500).

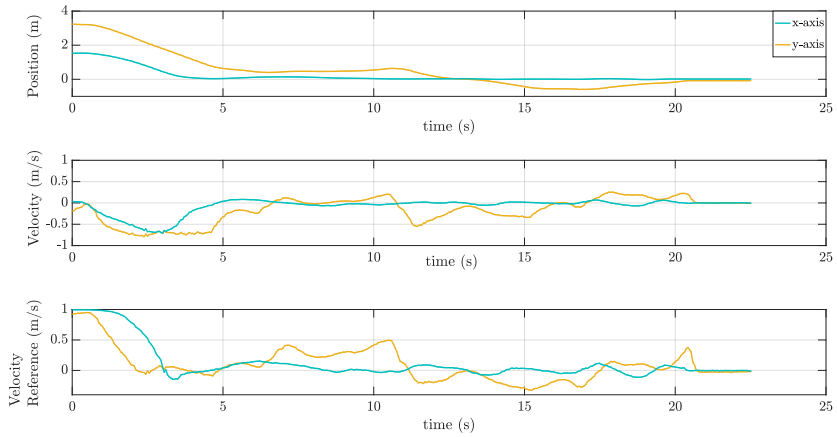
Scenario	t_{land} (s)	x (m) y (m)	SR (%)
Slow	13.5 ± 1.56	-0.01 ± 0.38 0.06 ± 0.47	90.6
Fast	17.76 ± 1.52	0.04 ± 0.42 -0.11 ± 0.49	73.3

over 150 test episodes. Both the UAV and the MP start at a random position in each episode (see testing phase area of Section 4.1). Two scenarios has been designed and several metrics have been provided. *Slow* scenario corresponds to a rectilinear periodic trajectory of the MP with a maximum velocity of 0.4 m/s. *Fast* scenario corresponds to a rectilinear periodic trajectory of the MP with a maximum velocity of 1.2 m/s. t_{land} represents the required time to perform the full landing maneuver (until the UAV touches the MP); x and y represent the final position of the UAV with respect to the center of the MP in x and y axes, respectively; and the Success Rate (SR) represents the percentage of successful landing trials with respect to the whole set (over 150 episodes). A failure in the landing maneuver is mostly due to the fact that

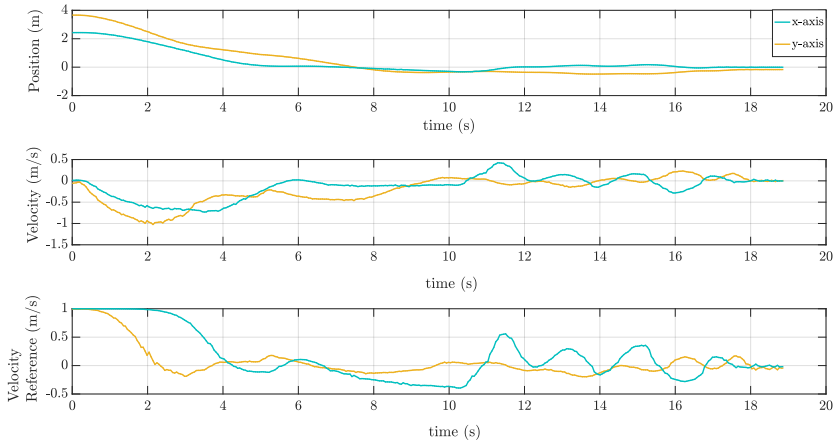
velocity in z axis is constant and the target MP can become out of range, from where the agent is not able to recover the MP position. Nevertheless, the SR suggests that this approach has succeeded in learning the landing maneuver. Also, both the t_{land} and the final position of the UAV with respect to the MP shows a proper performance.

The robustness of DDPG algorithm can be further validated in real flights, due to the difference in sensors and dynamics of the UAV used. It has to be noted that no additional tuning of the actions provided by the actor network was required when moving from simulation to real flights. Furthermore, as previously stated, the AscTec Hummingbird included in simulation training and testing phases, and the Parrot Bebop used in real-flight testing have a similar size but the simulated dynamics differ from the real ones. The dynamics of the AscTec Hummingbird were not required to be adjusted in simulation to fit Parrot Bebop dynamics. The explanation of this fact is twofold. First, our approach aims to prove a powerful workflow, where a robot can be trained in simulation and tested in real flights, even when the robot is not being precisely simulated (*e.g.* different dynamics or autopilot). In this context, the generalization capability of the DDPG algorithm was enough to overcome stated differences. Second, our approach performs high level control (velocity reference control), so that differences in dynamics can be partially absorbed by a proper tuning of the velocity and autopilot controllers.

In Figure 7, two real-flight plots are shown. In these real-flight experiments, the UAV is automatically commanded to land when the altitude with respect to the MP is lower than a certain threshold th ($th = 35$ cm), in order to avoid unsafe maneuvers that do not add value to the final performance. Also, the UAV is commanded a constant velocity in z axis. However, due to the Parrot Bebop autopilot design it can sometimes have sudden altitude changes due to missestimation of the UAV altitude in a certain instant (see supplementary video). Figure 7a, shows the performance of the UAV when the MP follows a linear periodic trajectory. This experiment seeks to replicate the scenario which has been used to train the agent in simulation, in order to prove that it is possible to describe a similar behaviour in a real flight. As seen in Figure 7a, the position of the UAV with respect to the MP converges to the origin and the velocity reference commands generated by the agent are stable with respect to time. Furthermore, in order to test the capability of generalization of the DDPG algorithm, a more complex experiment has been designed. In this new experiment, the MP describes a random trajectory in both x and y axes of the horizontal space (ground plane). This scenario has never been experienced by the agent, so that the approach can be proven to be robust and generic enough to overcome the uprising differences, as shown in Figure 7b. The results shown in Figure 7b depict a similar convergence, compared to previous results, leading to a proper landing of the UAV on the MP and with a stable generation of actions. The final high level performance of the UAV remains stable, smooth and robust against new experiences. Regarding these results, we can conclude that DDPG algorithm is capable of learning a complex and continuous landing maneuver task. In addition, it is feasible for a



(a) Partial state and actions signals of a real flight (test with a periodic trajectory of the MP). (Top) Position of the UAV with respect to the MP, in x and y axes. The final position of the UAV with respect to the MP is 0.01 m and -0.07 m, in x and y axes respectively. (Middle) Velocity of the UAV with respect to the MP, in x and y axes. (Bottom) Velocity reference commands (actions) generated by the agent, in world coordinates and in x and y axes.



(b) Partial state and actions signals of a real flight (test with a random trajectory of the MP). (Top) Position of the UAV with respect to the MP, in x and y axes. The final position of the UAV with respect to the MP is -0.01 m and -0.17 m, in x and y axes respectively. (Middle) Velocity of the UAV with respect to the MP, in x and y axes. (Bottom) Velocity reference commands (actions) generated by the agent, in world coordinates and in x and y axes.

Fig. 7: Partial state and actions signals of two test episodes (real-flight testing phase).

UAV to be trained in simulation and tested in the real world without further parameter tuning in a diverse range of conditions.

5 Conclusions and Future Work

In this paper, the problem of autonomous landing of a UAV on a moving platform has been solved by means of a deep reinforcement learning algorithm. The state-of-the-art DDPG algorithm was integrated and adapted into our novel Gazebo-based reinforcement learning simulation framework, enabling the possibility of training complex continuous tasks in a realistic simulation. The UAV landing maneuver task was trained in simulation, and tested in simulation and real flights. This fact has validated a powerful work flow for robotics, where robots can learn in simulation and properly perform in real flights. The experiments have been run in a wide variety of conditions, demonstrating the generality of the approach. To the best of the authors knowledge, this is the first work that addresses the UAV landing maneuver on top of a moving platform by means of a state-of-the-art deep reinforcement learning algorithm, trained in simulation and tested in real flights. Concerning the complexity of the landing problem (and other type of robotic problems), other continuous deep reinforcement learning algorithms can be integrated in our reinforcement learning simulation framework, since there is an ongoing innovation in this type of algorithms. As an immediate future work, the altitude (z axis) can be included in the states and actions space and some prediction-based solution can be tested. Furthermore, the input to the algorithm can be changed from a continuous space of variables to raw pixels, in order to test its capability of generalization from a higher amount of noisy information.

Acknowledgements This work was supported by the Spanish Ministry of Science (Project DPI2014-60139-R). The LAL UPM and the MONCLOA Campus of International Excellence are also acknowledged for funding the predoctoral contract of one of the authors.

A short version of this paper was presented in the 2017 International Conference on Unmanned Aircraft Systems (ICUAS), held in Miami, FL USA, on 13-16 June 2017.

References

1. A. Rucco P.B. Sujit, A.A.J.S., Pereira, F.L.: Optimal rendezvous trajectory for unmanned aerial-ground vehicles. arXiv preprint arXiv:1612.06100 (2016)
2. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016)
3. Alexandre Borowczyk Duc-Tien Nguyen, A.P.V.N.D.Q.N.D.S.J.L.N.: Autonomous landing of a multirotor micro air vehicle on a high velocity ground vehicle. In: IFAC World Congress (2017)
4. Ananthakrishnan, U.S., Akshay, N., Manikutty, G., Bhavani, R.R.: Control of Quadrotors Using Neural Networks for Precise Landing Maneuvers (2017)
5. Araar, O., Aouf, N., Vitanov, I.: Vision based autonomous landing of multirotor uav on moving platform. Journal of Intelligent & Robotic Systems (2017)

6. Arora, S., Jain, S., Scherer, S., Nuske, S., Chamberlain, L., Singh, S.: Infrastructure-free shipdeck tracking for autonomous landing. In: Robotics and Automation (ICRA), 2013 IEEE International Conference on, pp. 323–330 (2013)
7. Blösch, M., Weiss, S., Scaramuzza, D., Siegwart, R.: Vision based mav navigation in unknown and unstructured environments. In: Robotics and automation (ICRA), 2010 IEEE international conference on, pp. 21–28. IEEE (2010)
8. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)
9. Cantelli, L., Mangiameli, M., Melita, C.D., Muscato, G.: Uav/ugv cooperation for surveying operations in humanitarian demining. In: Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International symposium on, pp. 1–6. IEEE (2013)
10. Dorigo, M., Colombetti, M.: Robot shaping: an experiment in behavior engineering. MIT press (1998)
11. Espié, E., Guionneau, C., Wymann, B., Dimitrakakis, C., Coulom, R., Sumner, A.: Torcs-the open racing car simulator. Available at:¡ ext-link ext-link-type=" uri" xlink: href=" http://torcs. sourceforge. net"¿ http://torcs. sourceforge. net¡/ext-link (2005)
12. Falanga, D., Zanchettin, A., Simovic, A., Delmerico, J., Scaramuzza, D.: Vision-based autonomous quadrotor landing on a moving platform
13. Furrer, F., Burri, M., Achtelik, M., Siegwart, R.: Robot Operating System (ROS): The Complete Reference (Volume 1), chap. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. Springer International Publishing, Cham (2016). DOI 10.1007/978-3-319-26054-9_23. URL http://dx.doi.org/10.1007/978-3-319-26054-9_23
14. Gautam, A., Sujit, P.B., Saripalli, S.: A survey of autonomous landing techniques for uavs. In: 2014 International Conference on Unmanned Aircraft Systems (ICUAS) (2014)
15. Gautam, A., Sujit, P.B., Saripalli, S.: Application of guidance laws to quadrotor landing. In: 2015 International Conference on Unmanned Aircraft Systems (ICUAS) (2015)
16. Giusti, A., Guzzi, J., Cireşan, D.C., He, F.L., Rodríguez, J.P., Fontana, F., Faessler, M., Forster, C., Schmidhuber, J., Di Caro, G., et al.: A machine learning approach to visual perception of forest trails for mobile robots. IEEE Robotics and Automation Letters **1**(2), 661–667 (2016)
17. Gu, S., Lillicrap, T., Sutskever, I., Levine, S.: Continuous deep q-learning with model-based acceleration. In: International Conference on Machine Learning, pp. 2829–2838 (2016)
18. Hu, B., Lu, L., Mishra, S.: Fast, safe and precise landing of a quadrotor on an oscillating platform. In: 2015 American Control Conference (ACC) (2015)
19. Ivakhnenko, A.G.: Polynomial theory of complex systems. IEEE transactions on Systems, Man, and Cybernetics **1**(4), 364–378 (1971)
20. Kai, W., Chunzhen, S., Yi, J.: Research on adaptive guidance technology of uav ship landing system based on net recovery. Procedia Engineering **99**, 1027 – 1034 (2015)
21. Kelchtermans, K., Tuytelaars, T.: How hard is it to cross the room?–training (recurrent) neural networks to steer a uav. arXiv preprint arXiv:1702.07600 (2017)
22. Kendoul, F., Ahmed, B.: Bio-inspired taupilot for automated aerial 4d docking and landing of unmanned aircraft systems. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (2012)
23. Kim, D.K., Chen, T.: Deep neural network for real-time autonomous indoor navigation. arXiv preprint arXiv:1511.04668 (2015)
24. Kim, J., Jung, Y., Lee, D., Shim, D.H.: Landing control on a mobile platform for multi-copters using an omnidirectional image sensor. Journal of Intelligent & Robotic Systems **84** (2016)
25. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. The International Journal of Robotics Research **32**(11), 1238–1274 (2013)
26. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp. 1097–1105 (2012)
27. Lee, D., Ryan, T., Kim, H.J.: Autonomous landing of a vtol uav on a moving platform using image-based visual servoing. In: 2012 IEEE International Conference on Robotics and Automation (2012)

28. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
29. Ling, K., Chow, D., Das, A., Waslander, S.L.: Autonomous maritime landings for low-cost vtol aerial vehicles. In: 2014 Canadian Conference on Computer and Robot Vision (2014)
30. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1928–1937 (2016)
31. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
32. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA workshop on open source software, vol. 3, p. 5. Kobe (2009)
33. Rezelj, A.: Autonomous charging of a quadrocopter by landing at a mobile platform (2013)
34. Rodriguez-Ramos, A., Sampedro, C., Bavle, H., Milosevic, Z., Garcia-Vaquero, A., Campoy, P.: Towards fully autonomous landing on moving platforms for rotary unmanned aerial vehicles. In: Unmanned Aircraft Systems (ICUAS), 2017 International Conference on, pp. 170–178. IEEE (2017)
35. Sadeghi, F., Levine, S.: rl: Real singleimage flight without a single real image. arxiv preprint. arXiv preprint arXiv:1611.04201 **12** (2016)
36. Sampedro, C., Bavle, H., Rodríguez-Ramos, A., Carrio, A., Fernández, R.A.S., Sanchez-Lopez, J.L., Campoy, P.: A fully-autonomous aerial robotic solution for the 2016 international micro air vehicle competition. In: Unmanned Aircraft Systems (ICUAS), 2017 International Conference on, pp. 989–998. IEEE (2017)
37. Sanchez-Lopez, J.L., Fernández, R.A.S., Bavle, H., Sampedro, C., Molina, M., Pestana, J., Campoy, P.: Aerostack: An architecture and open-source software framework for aerial robotics. In: Unmanned Aircraft Systems (ICUAS), 2016 International Conference on, pp. 332–341. IEEE (2016)
38. Santana, P., Correia, L., Mendonça, R., Alves, N., Barata, J.: Tracking natural trails with swarm-based visual saliency. Journal of Field Robotics **30**(1), 64–86 (2013)
39. Serra, P., Cunha, R., Hamel, T., Cabecinhas, D., Silvestre, C.: Landing of a quadrotor on a moving target using dynamic image-based visual servo control. IEEE Transactions on Robotics **32**(6) (2016)
40. Shaker, M., Smith, M.N., Yue, S., Duckett, T.: Vision-based landing of a simulated unmanned aerial vehicle with fast reinforcement learning. In: Emerging Security Technologies (EST), 2010 International Conference on, pp. 183–188. IEEE (2010)
41. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M.: Deterministic policy gradient algorithms. In: Proceedings of the 31st International Conference on Machine Learning (ICML-14), pp. 387–395 (2014)
42. Skoczylas, M.: Vision analysis system for autonomous landing of micro drone. Acta Mechanica et Automatica (2015)
43. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction, vol. 1. MIT press Cambridge (1998)
44. Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pp. 5026–5033. IEEE (2012)
45. Uhlenbeck, G.E., Ornstein, L.S.: On the theory of the brownian motion. Physical review **36**(5), 823 (1930)
46. Vlantis, P., Marantos, P., Bechlioulis, C.P., Kyriakopoulos, K.J.: Quadrotor landing on an inclined platform of a moving ground vehicle. In: 2015 IEEE International Conference on Robotics and Automation (ICRA) (2015)
47. Wenzel, K.E., Masselli, A., Zell, A.: Automatic take off, tracking and landing of a miniature uav on a moving carrier vehicle. Journal of Intelligent & Robotic Systems (2011)

48. Zamora, I., Lopez, N.G., Vilches, V.M., Cordero, A.H.: Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo. arXiv preprint arXiv:1608.05742 (2016)
49. Zhang, T., Kahn, G., Levine, S., Abbeel, P.: Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In: Robotics and Automation (ICRA), 2016 IEEE International Conference on, pp. 528–535. IEEE (2016)