Journal of Intelligent & Robotic Systems (2021) 103:62
https://doi.org/10.1007/s10846-021-01491-2

SHORT PAPER

Quadrotor Path Following and Reactive Obstacle Avoidance with Deep Reinforcement Learning

Bartomeu Rubí[1] · Bernardo Morcego[1] · Ramon Pérez[1]

Received: 2 September 2020 / Accepted: 28 August 2021

Abstract

A deep reinforcement learning approach for solving the quadrotor path following and obstacle avoidance problem is proposed in this paper. The problem is solved with two agents: one for the path following task and another one for the obstacle avoidance task. A novel structure is proposed, where the action computed by the obstacle avoidance agent becomes the state of the path following agent. Compared to traditional deep reinforcement learning approaches, the proposed method allows to interpret the training process outcomes, is faster and can be safely trained on the real quadrotor. Both agents implement the Deep Deterministic Policy Gradient algorithm. The path following agent was developed in a previous work. The obstacle avoidance agent uses the information provided by a low-cost LIDAR to detect obstacles around the vehicle. Since LIDAR has a narrow field-of-view, an approach for providing the agent with a memory of the previously seen obstacles is developed. A detailed description of the process of defining the state vector, the reward function and the action of this agent is given. The agents are programmed in python/tensorflow and are trained and tested in the RotorS/gazebo platform. Simulations results prove the validity of the proposed approach.

Keywords Unmanned aerial vehicles · Obstacle avoidance · Path following · Deep reinforcement learning · LIDAR · Deep deterministic policy gradient

1 Introduction

Recent years are revealing an exponential growth on the research and applications on the deep reinforcement learning (DRL) field. It has been applied to a large number of different computer science, engineering and control problems with outstanding results [1–5]. In [6] a DRL algorithm was implemented to solve the path following (PF) problem with adaptive velocity for a quadrotor obtaining successful experimental results. In this paper, the capabilities of this learning paradigm are taken further by implementing a deep reinforcement learning agent for solving the reactive obstacle avoidance problem. This agent is combined with the one developed in [6] configuring a path following and obstacle avoidance autonomous solution.

The obstacle avoidance (OA) problem has been solved by many different classic techniques including Model Predictive Control (MPC), Potential Field methods or Rapidly-exploring Random Trees (RRT) [7–10]. However, machine learning theory is becoming more popular to solve this problem due to its good results. Most of the approaches used to solve the OA problem on UAVs by means of machine learning theory are based on the use of *Convolutional Neural Networks* (CNN). That is, a type of deep neural network where the inputs may be images. In [11] *CNN*s are used to extract the information of an obstacle from the images of a monocular vision system. Then, this information is used as the state input of an actor-critic RL algorithm that designs the trajectory to avoid the obstacle. In other cases, the obstacle avoidance algorithms based on DRL implement the *Deep Q-Network* (DQN) [12, 13] or

This work has been partially funded by the Spanish Government (MINECO) through the project CICYT (ref. DPI2017-88403-R).

✉ Bartomeu Rubí
tomeu.rubi@upc.edu

Bernardo Morcego
bernardo.morcego@upc.edu

Ramon Pérez
ramon.perez.@upc.edu

[1] Research Center for Supervision, Safety and Automatic Control (CS2AC), Universitat Politècnica de Catalunya (UPC), Rbla Sant Nebridi 22, Terrassa, Spain

Published online: 10 November 2021

the *Double DQN* [14–16]. These algorithms are variants of *Q-Learning* that use *CNN*s. They receive raw images of a camera (or cameras) as the reinforcement learning state. Applications of the *DQN* algorithm for the OA problem are also found in other systems such as marine vessels [17] and mobile robots [18].

Other approaches implementing the *Deep Deterministic Policy Gradient* (DDPG) algorithm for the UAV OA problem are also found in the literature. The main difference between *DQN* and *DDPG* is that in *DQN* the state inputs of the agent are images, while DDPG is especially designed for continuous state-action spaces. In [19] a *DDPG*-based approach for a fixed-wing UAV that performs ground target tracking while avoiding obstacles is presented. This approach uses *Long Short-Term Memory* (LSTM) neural networks to approximate the state of the environment from the obstacle detection distance measures. The *LSTM* networks are a type of recurrent neural networks (RNN) that use a sequence as their input data. That is, the training results at each step are determined by both the current training data and the historical training data. The obstacle detection sensor used in [19] has a field-of-view (FOV) of 180° . The proposed approach is trained and validated in a simulated environment. In [20] an adaptation of the *Recurrent Deterministic Policy Gradient* (RDPG) algorithm, a *DDPG*-like algorithm specially designed for partially observable Markov decision processes, is used to solve the UAV obstacle avoidance problem. The *DDPG* algorithm updates the weights of the networks step-by-step making use of the newest experience. Rather than that, the *RDPG* only updates the weights when an episode ends, using the entire episode experience. Thus, unlike *DDPG*, *RDPG* is not sample-efficient. In this paper a Fast-*RDPG* algorithm is designed in such a way that it permits learning online by updating its weights in terms of history trajectories instead of the entire episode. The proposed algorithm uses the distance measures of a range sensor with a 180° of field-of-view and is validated through simulation results. A LIDAR-based approach for multirotor navigation based on the *DDPG* algorithm is presented in [21]. Their objective is to develop an agent capable of driving the vehicle to a goal position avoiding obstacles in the vehicle's route. They use the measures of a LIDAR with a FOV of 270° as part of the environment state. The reward is designed using an Artificial Potential Field. The agent is trained in RotorS and is validated with real experimental results.

In this paper, a deep reinforcement learning approach is developed to solve the path following and reactive obstacle avoidance problem. Two agents, based on the implementation of the *DDPG* algorithm, are used. The OA training environment is developed in the RotorS/Gazebo framework. The resulting agents are trained in the RotorS simulated environment following a path while dealing with static obstacles and tested with different configuration of obstacles (static, single, multiple, cluttered, different shapes, different sizes, etc). The contributions of this paper are: (i) The reactive obstacle avoidance approach uses the processed measures of a low-cost LIDAR sensor as states of the agent. In contrast to the LIDAR sensors used in the state-of-the-art obstacle avoidance approaches, which usually provide a map of the environment in the 360° field-of-view (or similar FOVs), this LIDAR only provides 8 distance measures in a horizontal space of 48° . This makes the problem significantly more challenging. (ii) Since the obstacle detection sensor has a narrow field-of-view, an approach for providing the agent with a memory of the previously seen obstacles is developed. Though recurrent neural networks are able to interpret historical data, they are not included in the algorithm's structure since they would make the training process slower. Instead, a simpler and effective solution that consists only on mathematical relations is proposed. (iii) The main problem is solved by implementing a novel structure that consists on two DRL agents: One that solves the path following problem and another one for the reactive obstacle avoidance problem. The novelty relies in the fact that the action computed by the OA agent directly modifies the state of the PF agent, converting it into a cascade structure. Compared to traditional DRL methods, it allows to interpret the training process outcomes and is faster. Furthermore, it can be safely trained on the real quadrotor platform.

The rest of the paper is organized as follows: In Section 2 the problem addressed in the paper is defined. Section 3 describes the elements that form the environment of the agent. In Section 4 the designing process of the proposed deep reinforcement learning approach for obstacle avoidance is presented. Section 5 describes how the path following and obstacle avoidance agents are implemented and how they are integrated to solve the proposed problem. The training process of the agents is evaluated in Section 6. Section 7 reports the simulation results of the agents and discusses their performance. Finally, section Section 8 details the conclusions.

2 Problem Statement

The aim of this paper is to develop a reactive obstacle avoidance approach by means of deep reinforcement learning theory. The resultant system must be able to follow a predefined generic path with adaptive velocity. That is, preserving the same functionality of the DRL path following agent presented in [6]. Furthermore, the system must be able to avoid static obstacles that may appear in the vehicle's route. To do so, the agent must use only the local

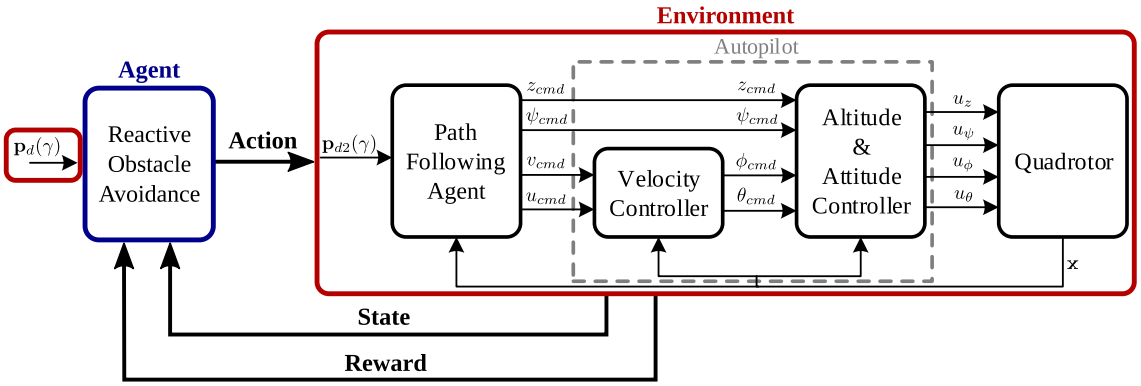

Fig. 1 Reinforcement learning structure employed in this work

information of the environment provided by the obstacle detection sensor, a frontal LIDAR.

Reactive obstacle avoidance is understood as a path planning algorithm that only considers local information of the vehicle's environment to design an alternative route when an obstacle appears in the vehicle's route. That is, it plans online a trajectory that prevents collisions in the last minute.

The algorithm selected to solve the reactive OA problem is the *Deep Deterministic Policy Gradient*, the same algorithm that was used in [6] to solve the path following problem. It is an actor-critic reinforcement learning algorithm; the actor computes the optimal action in function of the environment state and the critic estimates the value function and computes the loss function that is used to train both the actor and the critic. This algorithm is off-policy because the policy that is used to compute the loss function is different from the one that is being improved, and it is model-free since it estimates directly the optimal policy making no effort to learn the dynamics of the environment. It uses particular characteristics of other DRL algorithms, such as the replay buffer, the target network concept [22] or the batch normalisation technique [23]. Details on this deep reinforcement learning algorithm are given in [6, 24, 25].

The structure proposed to solve the described problem relies on maintaining the PF agent developed in [6] and creating a new agent that is in charge of providing the reference path to the PF controller in such a way that the main functionalities of the PF agent are kept while being able to avoid static obstacles. The path following and obstacle avoidance problem might also be addressed by a unique agent. However, considering the amount of training episodes needed to learn to follow a path selecting the optimal vehicle's velocity depending on the path radius, and considering the difficulty of the reactive obstacle avoidance problem itself, the number of training episodes and the size of the network structures could also increase dramatically. This is the main reason to divide the solution into two agents in this work.

The main elements of the reinforcement learning structure employed in this work are shown in Fig. 1. In this case, the agent is the OA algorithm, and the environment includes the rest of elements that interact with the agent, which are the path following agent, the autopilot, the quadrotor, the reference path and the quadrotor's environment. Note that the OA agent receives the original reference path and is in charge of computing the reference path that is commanded to the path following agent, $\boldsymbol{p}_{d2}(\gamma)$. In this paper the autopilot is formed by a set of PID-based controllers.

3 Agent Environment

The quadrotor of the agent's environment is the Asctec Hummingbird (Fig. 2). The agent is trained in a simulated environment to preserve the integrity of the experimental platform. Details of the quadrotor model, the obstacle

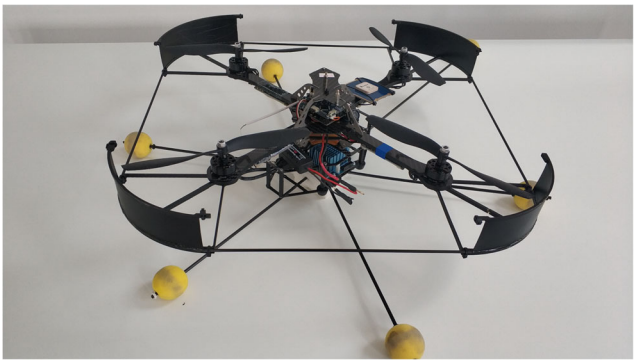

Fig. 2 Asctec Hummingbird Quadrotor

detection system and the training environment of the reactive obstacle avoidance agent are given next.

3.1 Quadrotor Model

The model of the quadrotor employed in this work has twelve states and four inputs. The states are the position in the world frame (x, y and z), the Euler angles (ϕ-*roll*, θ-*pitch* and ψ-*yaw*), the body velocities (u, v and w) and the angular velocities (p, q and r). The inputs are four digital signals that are related to the thrust force (u_z) and torques on each axis of the body frame (u_ϕ, u_θ and u_ψ). More information of the Asctec Hummingbird model in [26].

3.2 Obstacle Detection

The sensor used to perform the obstacle detection task is the Leddar VU8 LIDAR. This LIDAR has a horizontal field-of-view of 48° and a vertigal FOV of 3° . The Leddar VU8 divides its field-of-view in 8 horizontal segments, as it is shown in Fig. 3. That is, this sensor provides a vector of 8 values representing the minimum distance to an obstacle in that segment. Each segment has a FOV of 6° x 3° . Distances are given with a precision of $1cm$.

In this work the data provided by the LIDAR sensor is processed to eliminate the ground detections. To do so, an estimation of the distance to the ground detected by each LIDAR beam ($d_{G,i}$) in function of the vehicle's altitude in the world frame (z) and the pitch (θ) and roll (ϕ) angles is computed. This estimation is shown in Eq. 1, where $\psi_{L,i}$ is the angle of the i-th LIDAR beam with respect to the x axis of the sensor. If the distance measured by the sensor is equal or larger to the estimated ground distance, this value is set to infinite (it is considered that no obstacles are detected). Due to noise or inaccuracies of the sensor, the measured distance to the ground can be slightly different to the estimated one. For this reason, a margin of +20% is applied to the estimated ground distances. Note that Eq. 1 is only valid when both pitch and roll angles are different from zero. Therefore, other simpler formulas are used to estimate the ground distances when pitch or roll are zero.

$$d_{G,i} = \frac{z \sin\left(\arctan\left(\frac{\sin(\theta)}{\sin(\phi)}\right)\right)}{\cos(\theta)\cos\left(\pi/2 - \arctan\left(\frac{\sin(\theta)}{\sin(\phi)}\right) - \psi_{L,i}\right)} \tag{1}$$

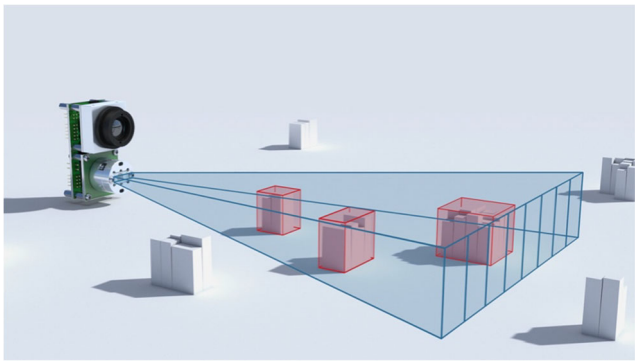

Fig. 3 Leddar VU8 detects obstacles in 8 segments (original image from leddartech.com)

3.3 Path Following Agent

The path following problem [27, 28] is defined as making the vehicle follow a predefined path ($\boldsymbol{p}_d(\gamma) := [x_d(\gamma), y_d(\gamma), z_d(\gamma)]^T$) in the space without any time constraint. That is, unlike the trajectory tracking approach where the trajectory is defined as a function of time, in the path following approach the path is parametrized by a scalar parameter, γ, known as the virtual arc. This approach results in many advantages over the standard trajectory tracking approach [29, 30].

The path following agent used in this paper was developed in a previous work [6]. It also implements the Deep Deterministic Policy Gradient algorithm. The PF agent's state vector is formed by the cross-track error, the angle error between the vehicle and the path, the angle error in a forward point on the path, which provides the ability of anticipate the curves, and the velocity of the vehicle. The PF agent acts on the derivatives of the yaw angle and the vehicle's velocity. The reward function depends on the cross-track error and on the vehicle's velocity projected on the path's tangential frame of reference. A simulation environment combining the Gazebo/RotorS and the python/tensorflow frameworks was build. The PF agents presented in [6] were trained in this environment and tested outdoors experimentally with successful results. The resulting PF agent was able to find a generalized solution of the PF problem following accurately any reference path while selecting the optimal vehicle's velocity according to the path's shape and vehicle's dynamics.

3.4 Autopilot

The autopilot is formed by a set of six PID-based controllers; two for controlling the velocities on the x and y axis, one for controlling the altitude (z) and three for controlling the attitude angles (ϕ, θ and ψ). The performance of these controllers was tested in experimental results [26]. Moreover, real and simulated response were very similar, which proves the reliability of the model.

3.5 Training Environment

RotorS [31], a realistic and complete multirotor simulator built in the Gazebo/ROS platform, is used as the training environment for the agent. A model of the sensors was included and adjusted to resemble the sensors of the actual quadrotor platform. Rubí et al. [6] describes some of the modifications that were made in the RotorS framework to obtain a training environment for the DRL PF agents. However, to perform obstacle avoidance experiments it is necessary to include new elements in this simulator. The LIDAR sensor of the real platform was modelled in the RotorS simulation environment. This model includes the visual part, the dynamics part and the generated ROS topic. Gaussian noise on the LIDAR measurements was included. Furthermore, to perform OA experiments, a key element is necessary: the obstacle/s. In this work, obstacles are modelled as cylindrical objects. The generated obstacle has a radius of 1m and a height of 2m. Figure 4 shows a screen capture of the RotorS/gazebo framework with the Hummingbird quadrotor, the generated obstacle and the modelled LIDAR sensor.

The reactive OA agent, just as the PF agent [6], was programmed in Python 3.5, and it communicates with the ROS framework by means of the *rospy* library. Tensorflow and tflearn libraries were used to generate the neural networks and to train them. Obstacles can be spawned and deleted using the *spawn_sdf_model* and *delete_model* gazebo services. These services are used to define the initial position of the obstacle and to create multiple instances of the same object. In the defined training framework, obstacles are generated by calling those services in the Python script that implements the OA agent.

4 DDPG for Reactive Obstacle Avoidance

This section gives details of each of the elements that form the *DDPG* approach that was developed in this work to solve the reactive obstacle avoidance problem. That includes the employed control structure, the action, state and reward of the agent, the structures of the neural networks and other parameters related to the *DDPG* algorithm.

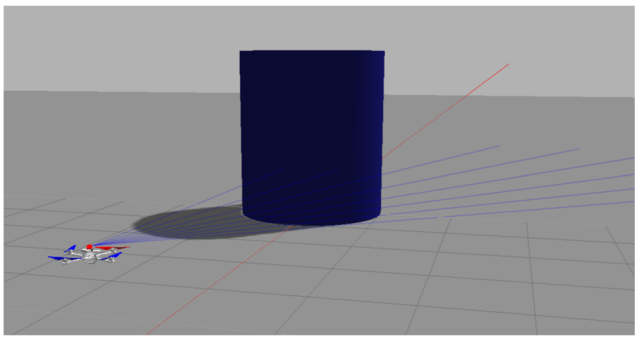

Fig. 4 Capture of RotorS/Gazebo including the Hummingbird quadrotor with the LIDAR sensor and an obstacle

4.1 Action

Figure 1 shows the reinforcement learning structure. In this structure, the OA agent receives the reference path ($\boldsymbol{p}_d(\gamma)$) calculates the obstacle free path ($\boldsymbol{p}_{d2}(\gamma)$) to be sent to the PF agent. The reference path is given by two vectors, one for each component (x_d and y_d), discretized by parameter γ with a precision of 0.01. There are different ways of sending the modified path to the PF agent. For instance, the full path could be sent at each step, however, this would be computationally expensive and inefficient. Otherwise, a small section of the path, corresponding to the part that is being currently followed, could be sent, but it could still become too demanding for the communication channel. Furthermore, making the OA agent compute the full sequence of path points would become a challenging problem. Instead, in this work a simple and efficient way of sending the path to the PF agent was considered; sending only the RL state vector of the PF agent. That is, the real environment's state vector of the PF agent is calculated in the obstacle avoidance script, and the OA agent computes certain variations or increments over this state. In this way, the PF agent is blind about the reference path and only receives the modified RL state vector, and the OA agent can generate the state increments to modify the state/path in such a way that obstacles are avoided.

At this point, it is important to recall that the state vector of the PF agent is formed by four elements [6]; the cross-track error, the angle error, the angle error in a point forward on the path and the velocity of the vehicle (Section 3.3). The vehicle's velocity is an intrinsic state of the vehicle, but the rest of the states depend on the path that is being followed. Thus, modifying these states is equivalent to changing the reference path. In the present work, only the cross-track error state will be modified, leaving the other three states unchanged.

Modifying the cross-track error state is equivalent to having an offset to the reference path, as it is represented in Fig. 5. That is, this action can be interpreted as the original path displaced in the y axis of the path tangential frame, $\{T\}$. Controlling the offset to the path, the agent will generate the corresponding trajectories to avoid the appearing obstacles. It is important to mention that, as the angle error state is computed as the angle between two vectors (x-body axis and x-tangential axis), it remains constant even if the path is translated.

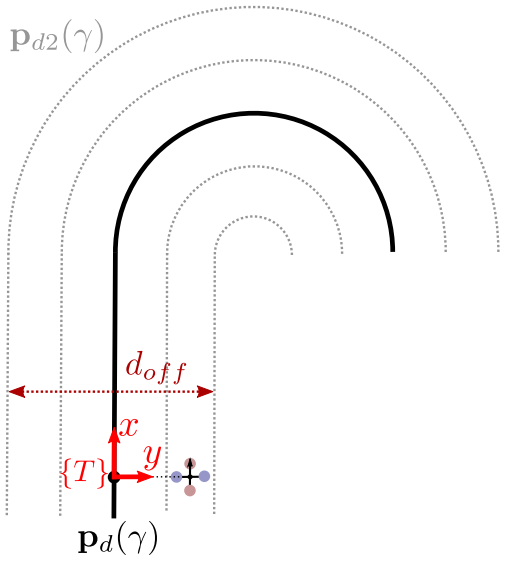

Fig. 5 Action of the agent modifies the path offseet d_{off}

The distance offset to the path is named d_{off}. To maintain a smooth profile of this path offset, the action of the OA agent is set to be an increment over it. That is, the action is set to be the derivative of the path offset. This is represented in Eq. 2. This equation shows, at time step k, how the modified cross-track error state ($e_{d2,k}$) is computed from the distance offset ($d_{off,k}$), and how the distance offset is obtained from the action (a_k) of the OA agent, where Δt is the time step of this agent. The derivative of the path offset is limited to $3^m/_s$, which was considered to be a sufficiently high lateral speed to execute a reactive manoeuvre.

$$e_{d2,k} = e_{d,k} + d_{off,k} \;\mid\; d_{off,k} = d_{off,k-1} + a_k\,\Delta t \tag{2}$$

During the training process, an Ornstein–Uhlenbeck noise function (Eq. 3) is added to the action of the agent for exploration purposes. The modified PF state including this noise function is computed in Eq. 4. The influence of this noise function is decreased continuously with the number of training episodes (j) in such a way that a transition between exploration and exploitation of the policy is achieved during the learning process. Parameter λ regulates the velocity of this transition.

$$n_k = n_{k-1} + \theta_n\,(\mu_n - n_{k-1})\,\Delta t + \sigma_n dW_t \tag{3}$$

$$e_{d2,k} = e_{d,k} + d_{off,k-1} + \left(a_k + \frac{n_k}{j/\lambda + 1}\right)\Delta t \tag{4}$$

4.2 State

The state vector of the OA agent has 14 states and it is represented in Eq. 5. Next paragraphs detail each of the parameters included in this state vector.

$$\boldsymbol{s} = \left\{d_{off},\; L_1 \cdots L_8,\; e_d,\; u,\; \overline{k}(\gamma_{min}),\; n_{L,l},\; n_{L,r}\right\} \tag{5}$$

The state vector (5) includes the path offset, d_{off}. Since the action computed by this agent is the derivative of this parameter, knowing this state is necessary to perform the correct offset modifications over the path. Next, the treated LIDAR measures, $L_1 \cdots L_8$ (see Section 3.2) are included to provide the OA agent with information about the environment and possible obstacles. The cross-track error (i.e. the first state of the PF agent, e_d) is also included in the state vector. The agent needs to know the distance from the vehicle to the reference path to localize the vehicle in space. The velocity of the vehicle in the x axis, u, is included since the avoidance manoeuvre depends on the vehicle's velocity. As the optimal avoiding trajectory depends on the shape of the path too, the curvature of the path, $\overline{k}(\gamma)$, is also set as a parameter of the state vector. Specifically, this state is computed as the average of the path's curvature in a section of 5 meters around γ_{min} (path closest point to the vehicle given by γ), Eq. 6, where $\gamma_{min,i}$ and $\gamma_{min,f}$ are the initial and final points of the average window and n is the number of points used to calculate the average.

$$\overline{k}(\gamma_{min}) = \left(\sum_{\gamma=\gamma_{min,i}}^{\gamma_{min,f}} \frac{\left\| \frac{d\mathbf{p}_d(\gamma)}{d\gamma} \times \frac{d^2\mathbf{p}_d(\gamma)}{d\gamma} \right\|}{\left\| \frac{d\mathbf{p}_d(\gamma)}{d\gamma} \right\|^3}\right) \Big/ n \tag{6}$$

With the introduced set of states, the system is capable of following the path in a correct way and, when the LIDAR detects an obstacle in the vehicle's route, it is able to perform a reactive action to start avoiding it. However, since the LIDAR has a FOV of 48° , as soon as the vehicle starts evading the obstacle, it disappears from the LIDAR's field-of-view. When this happens, the OA agent considers that the vehicle has already avoided the obstacle and moves back to the path provoking a collision. This situation is represented in Fig. 6, where the trajectory of the vehicle is represented with a grey line and the collision instant is marked with a red star. That behaviour occurs because, in the *DDPG* algorithm, the action computed according to this agent's policy only considers the current information provided by the state vector, without taking into account the historical data. Therefore, if this agent notices that the vehicle is out of the path and no obstacles are detected, it will always try to converge back to the reference path.

To deal with that, it is necessary to supply historical information of the obstacles to the OA agent. Other

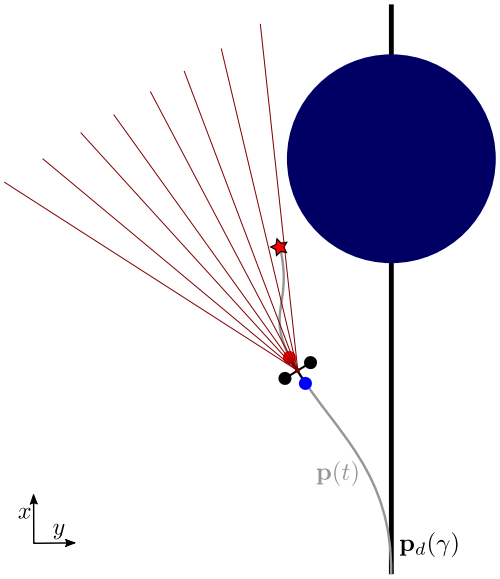

Fig. 6 Vehicle crashes if only instantaneous information of LIDAR is used

works provide the historical information by using recurrent neural networks, such as [19] where *LSTM*s networks are integrated in the *DDPG* algorithm. The issue of RNNs is that convergence is not always achieved and the training process becomes slower. In this work a simpler and functional solution has been chosen: defining two states, $n_{L,l}$ and $n_{L,r}$, that increase when the LIDAR is detecting an obstacle and decrease otherwise. To compute these states, if an obstacle is detected, the inverse of the distance measured by each LIDAR beam is integrated. And when there are no near obstacles in the LIDAR's field-of-view, the states decrease with a predefined rate. That is, these states are equivalent to an integral of the LIDAR detections. Therefore, the agent can learn that positive values of these states mean that some obstacle has been recently detected. Thus, the vehicle will not move back to the path until this state reaches low values.

In this work, two LIDAR integral states ($n_{L,l}$ and $n_{L,r}$) are defined: one state for the detections of the left-side beams of the LIDAR ($n_{L,l}$) and another state for the detections on the right-side beams ($n_{L,r}$). This agent does not only know that an obstacle has been recently detected, but it also can determine whether it was detected on the left or on the right side of the vehicle. With this approach, the trained OA agents acquired the ability of deciding when to avoid the obstacles by the left-side and when by the right-side, depending on the vehicle, path and obstacle's situation.

The formal definition of $n_{L,l}$ and $n_{L,r}$ states is shown in Eqs. 7 and 8, respectively, where $d_{L,i}$ is the distance detected by the ith beam of the LIDAR. To determine whether a detection is occurring or not, a distance threshold, d_T, is used. The values of k_c and k_d constants (charging and discharging rates of the state, respectively) determine the dynamics of this integral state. Note that n_L is constrained to avoid negative values and to have a maximum value of $n_{L,max}$. The maximum value, $n_{L,max}$, permits to regulate the time that the vehicle must take to converge back to the path after the avoiding manoeuvre has started.

$$\begin{aligned} &\text{if} \quad \forall i \in [1,2,3,4]\ \exists!\, d_{L,i} < d_T \quad \text{then} \quad n_{L,l} = \max\left(0,\ n_{L,l} - k_d\right) \\ &\text{otherwise} \quad n_{L,l} = \min\left(n_{L,max}\ ,\ n_{L,l} + \sum_{i=1}^{4} \frac{k_c}{d_{L,i}}\right) \end{aligned} \tag{7}$$

$$\begin{aligned} &\text{if} \quad \forall i \in [5,6,7,8]\ \exists!\, d_{L,i} < d_T \quad \text{then} \quad n_{L,r} = \max\left(0,\ n_{L,r} - k_d\right) \\ &\text{otherwise} \quad n_{L,r} = \min\left(n_{L,max}\ ,\ n_{L,l} + \sum_{i=5}^{8} \frac{k_c}{d_{L,i}}\right) \end{aligned} \tag{8}$$

4.3 Reward

To define the reward function, two virtual zones around the obstacle were created: the banned zone and the safety zone. These zones are represented in Fig. 7, which shows the obstacle in blue surrounded by two circles that represent the two zones. The banned zone is a concentric cylinder of the obstacle with a radius 0.5m larger than it while the safety zone has a radius 0.75m larger than the obstacle. In this figure the vehicle is represented proportionally to the obstacle and to the defined zones.

If the vehicle enters the banned zone, it is considered that a collision has occurred, and the simulation is stopped. Thus, the vehicle is not allowed to enter this zone. If the vehicle enters the safety zone while training, the OA agent will receive a negative reward, but the training episode will continue. Therefore, the trained agent will learn to avoid entering this zone to maximise the received reward. The safety zone is only present while training, so, if the vehicle

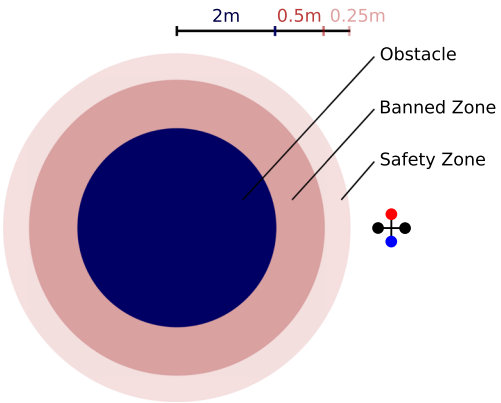

Fig. 7 Obstacle surrounded by the banned zone and the safety zone

enters the safety zone while performing a test, nothing will happen.

Defining a banned zone is always necessary to keep a proper distance to the obstacle when avoiding it. However, when only the banned zone is defined, the trained agents will always learn to avoid the obstacle travelling around the edge of the banned zone. Therefore, due to sensor noise or external disturbances, the vehicle often enters the banned zone and the experiment is stopped. The mentioned safety zone prevents this issue.

The reward function defined in this work is shown in Eq. 9. This function has three terms. The first term, $-10|d_{off}|$, penalizes the path offset distance. This is to force the OA agent to send the real cross-track error state (with no path offset) to the PF agent when no obstacles are present. The second term, $-50SZ$, penalizes the vehicle when it enters the safety zone, where SZ is a boolean parameter that is 1 when the vehicle is in the safety zone and 0 otherwise. And the third term, $-2000BZ$, penalizes the vehicle when it enters into the banned zone, where BZ is another boolean parameter that is 1 when the vehicle enters the banned zone and 0 otherwise. Note that this third term can only be activated once in an episode, since when the vehicle enters the banned zone it is considered that a collision has occurred and the episode is terminated. The reward function also includes a positive reward of 10 units given at each step. This positive reward term doesn't add any new information to the reward function but it helps to the interpretation of the training data, since with it the accumulated episode reward converges to a positive value during the training process of the agent.

$$r = -10|d_{off}| - 50SZ - 2000BZ + 10 \tag{9}$$

It is important to mention that this is the reward that provided the best results in terms of path following error and obstacle avoidance capability. Several other reward functions were also tested with poorer results. For instance, instead of the defined constant penalty when entering the safety zone ($-50SZ$), a penalty proportional to the inverse of the obstacle's distance (similar to an artificial potential field) was tested. However, the agents trained with this reward presented problems for converging to a stable solution. Furthermore, another reward based on the LIDAR measures was also tested. In this case the agents were able to converge faster in the training process, but they showed worse obstacle avoidance capabilities.

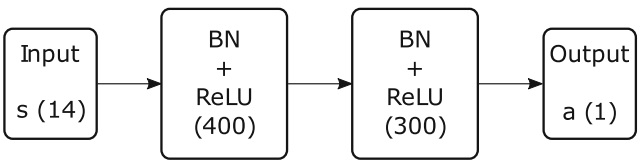

Fig. 8 Actor NN structure: 2 feed-forward hidden layers

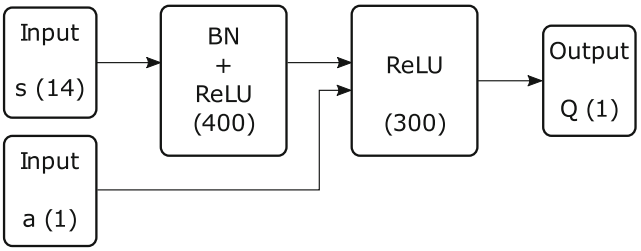

Fig. 9 Critic NN structure: 2 feed-forward hidden layers

4.4 Structure of the *DDPG* agent

The structure of the actor neural network is shown in Fig. 8. The number of elements of each block is given in parenthesis. The input is the state vector (14) and the output is the action (1) of the agent. The critic neural network is shown in Fig. 9, where the inputs are the state vector (14) and the action (1) and the output is the Q-value function (1). Both networks have two hidden layers of 400 and 300 neurons, respectively. They use rectifier linear unit (RelU) as activation function. The batch Normalization (BN) technique [23] is included in some layers to normalise the units, and hence, to help the neural networks to learn properly and to generalize the solution of the problem. Some parameters of the agent, such as the learning rates, discount factor or the minibatch size, were tuned during the first stages of the learning tests, leading to the values given in Table 1. The agent time step is fixed by the frequency of GPS sensor of the real platform (10hz) that is used to compute the position and velocity states.

Table 1 Parameters of the *OA* agent

Symbol	Description	Value
η_θ	Learning rate of actor network.	0.0001
η_ϕ	Learning rate of critic network.	0.001
τ	Soft target update parameter.	0.001
γ	Discount factor for critic updates.	0.99
-	Replay buffer size.	1,000,000
N	Minibatch size.	64
Δt	Agent time step.	0.1 s
θ_n	Mean reversion rate of noise function.	0.15
σ_n	Volatility of noise function.	3
λ	Ratio of exploration-exploitation transition.	500
-	Maximum steps of one episode.	3000

5 Implementation of the PF and Reactive OA

Two python 3.5 scripts are programmed to implement the path following and reactive obstacle avoidance procedure. In Fig. 10 the flowchart of these two scripts is presented, where communication elements are represented in grey. It can be observed that the PF script includes two control modes: the path following mode and the hover mode. In the path following mode, the *DDPG* PF agent is in charge of following the reference path that is received from the OA script in the form of subsequent PF state vectors. In the hover mode a PID controller is used to hover the vehicle around a reference point in space. In both control modes, the path following commands (ψ_{cmd}, z_{cmd}, u_{cmd} and v_{cmd}) are sent to the autopilot controller. The PF script starts in hover mode and waits for the OA script to proceed with the path following task.

The path to be followed and the transition between the two control modes is supervised by the OA script. In the OA script, first, the definition of the reference path is made, and then, the *DDPG* OA agent is launched. This agent computes the path offset action, which is used to obtain the PF state that is sent to the PF script. Once the path following task is finished, the OA script sends the PF script back to hover mode. There are two ways of finishing the PF task: by reaching the end of the path or when a collision is produced. In the simulated framework, it is considered that a collision is produced when the vehicle enters the banned zone (Fig. 7). When this happens, the obstacle is deleted from the gazebo framework and the hover mode is activated. In the simulated framework the OA script is also in charge of generating obstacles and place them randomly along the path and around it. This procedure is made after the reference path is defined.

6 Training process

The training process of the agent was made in the environment described in Section 3.5. The training environment is integrated in a linux Xubuntu virtual machine with a dedication of 8GB RAM and four 1.80GHz processors (i7-8550U CPU), and the training process is performed in real time.

The path following and obstacle avoidance problem can become very complex because many different situations can occur. That is, obstacles can appear at different locations while following diverse path shapes. To obtain a complete and functional OA agent to deal with the stated problem, a very rich training framework must be generated.

The generated training framework consists on making the vehicle follow a straight line of 10m followed by a half asymmetrical lemniscate path (Eq. 10) where the value of

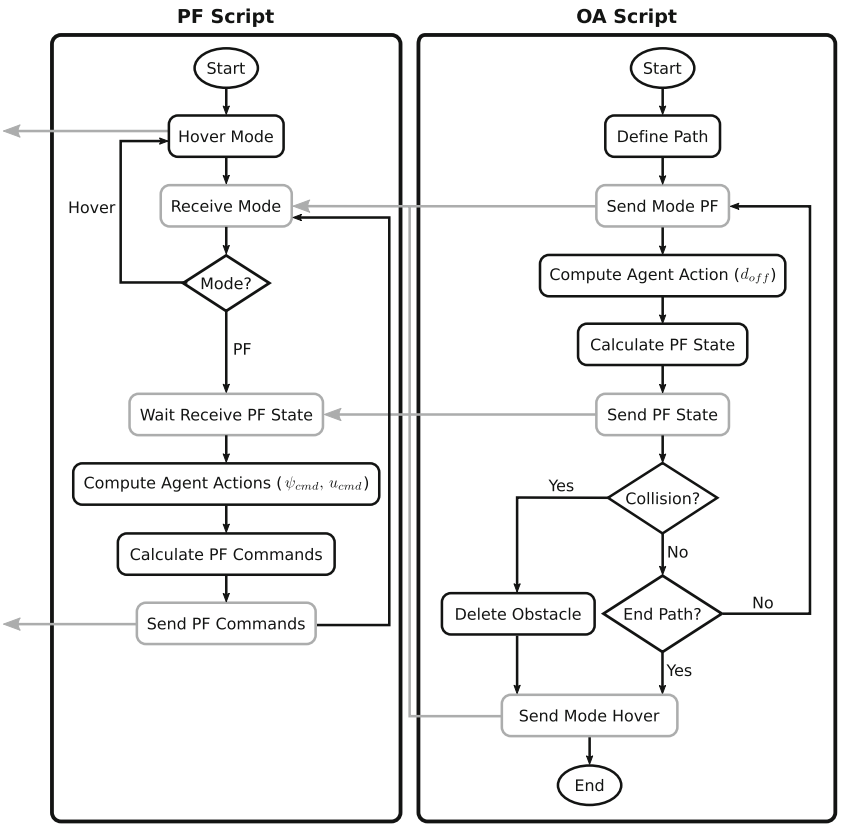

Fig. 10 Flowchart of the two scripts that implement the PF and Reactive OA approach

its two radius (A_1 and A_2) change every episode, taking a random value between $3m$ and $10m$ with a uniform probability distribution. This path, without the straight line path, was used to train the PF agent. Following paths with different curves and with straight lines allows the OA agent to learn the avoiding manoeuvres at different situations.

$$\begin{aligned} x_d(\gamma) &= \begin{cases} 2A_1\cos(\gamma) & \text{if } 0 \le \gamma \le \frac{\pi}{4} \\ 2A_2\cos(\gamma) & \text{if } \frac{\pi}{4} < \gamma \le \frac{\pi}{2} \end{cases} \\ y_d(\gamma) &= \begin{cases} A_1\sin(2\gamma) & \text{if } 0 \le \gamma \le \frac{\pi}{4} \\ A_2\sin(2\gamma) & \text{if } \frac{\pi}{4} < \gamma \le \frac{\pi}{2} \end{cases} \end{aligned} \tag{10}$$

The straight line at the beginning of the path is used to prevent from having an obstacle just in front of the vehicle when starting an episode. Therefore, no obstacles are placed in this line. This is coherent with real experiments, since the human supervisor would not launch the vehicle with an obstacle in front of it. Furthermore, this starting line on the path permits to place obstacles in any point along the Lemniscate path, which allows the OA agent to learn avoiding manoeuvres with different path's shapes. Also, the straight line path permits to reach the cruise velocity before starting the lemniscate path, preventing the avoidance manoeuvres at the transient phase.

Obstacles are generated in each episode with a probability of 75%, so, in average, 1 out of 4 episodes is carried out without an obstacle. That allows the OA agent to learn the policy when no obstacles are present. The obstacles are placed at random locations along the lemniscate path, with a uniform probability distribution, in a range of $\pm$2m of distance to the path. That allows this agent to learn to avoid obstacles that are centred on the path as well as obstacles that are close to it, deciding which side is better to avoid them, depending on the location of the obstacle, the vehicle and the path shape. Furthermore, it is trained with obstacles that are out of the vehicle's route in such a way that the LIDAR detects them but no avoidance manoeuvre is needed.

The OA agent is trained in ideal conditions. That is, the system uses ground truth measurements and the vehicle starts each episode at the initial position of the path with the yaw angle oriented tangentially to it. It is important to recall that the orientation and velocity of the vehicle are controlled by the autopilot and that the path following problem is solved by the PF agent developed in [6]. Therefore, these two control blocks are also included while training the OA agent.

It is interesting to mention that, without the processing of LIDAR measurements to eliminate ground detections, most of the trained OA agents had problems interpreting the distance measurements provided by this sensor. The agents trained in these conditions converged to a solution where the vehicle always followed the path at certain distance to it. That is, the vehicle remained at am enough large distance to elude all the obstacles that appeared. When the LIDAR measurements are treated, this peculiar behaviour disappears.

The results of the training process of the OA agent with the state defined in Eq. 5 and the reward function stated in Eq. 9 are shown in Fig. 11. This agent was trained during 10500 episodes. As can be observed, the average path distance and velocity increase over the episodes, reaching a stable value of around $0.31m$ and $1.24^m/_s$, respectively. The accumulated reward stabilizes around 2053. The moving average of the boolean parameter that represents whether a collision in the episode occurred or not, stabilizes to a value of 0.048. This value can be considered an indicator of the probability of having a collision. This indicator remains close to 0 in the first episodes, since this agent cannot properly follow the path yet, and starts increasing when it improves the path following performance because obstacles are placed near to the path. This indicator reaches a maximum value of around 0.25, meaning that 1 out of 4 episodes ends in a collision. However, the stabilized value of 0.048 can be considered of having a collision every 20 episodes. It is important to mention that the noise function can increase the probability of having a collision.

7 Results

This section presents the results obtained with the OA agent that achieved the best performance in terms of path following error and obstacle avoidance capabilities among the different agents that were trained. The training process of this agent is presented in Fig. 11. Two paths are used to test the approach: a lemniscate and a spiral. Moreover, the approach is tested in situations never seen before during the training phase: multiple obstacles, larger and smaller obstacles and different shaped obstacles. These simulation results are shown in next subsections.

7.1 Lemniscate Path

This section evaluates the OA agent with a lemniscate path (11), where the amplitude, A, is set to 5m, and γ ranges from 0 to 2π, corresponding to a full lap on the path.

$$\begin{aligned} x_d(\gamma) &= 2A\cos(\gamma) \\ y_d(\gamma) &= A\sin(2\gamma) \end{aligned} \tag{11}$$

Table 2 compares the results obtained by the PF agent developed in [6] with the ones obtained by the PF+OA approach presented in this paper. That is, the approach that combines the two DRL agents to solve the path following and obstacle avoidance problem. These results evaluate the performance of both approaches performing one lap of

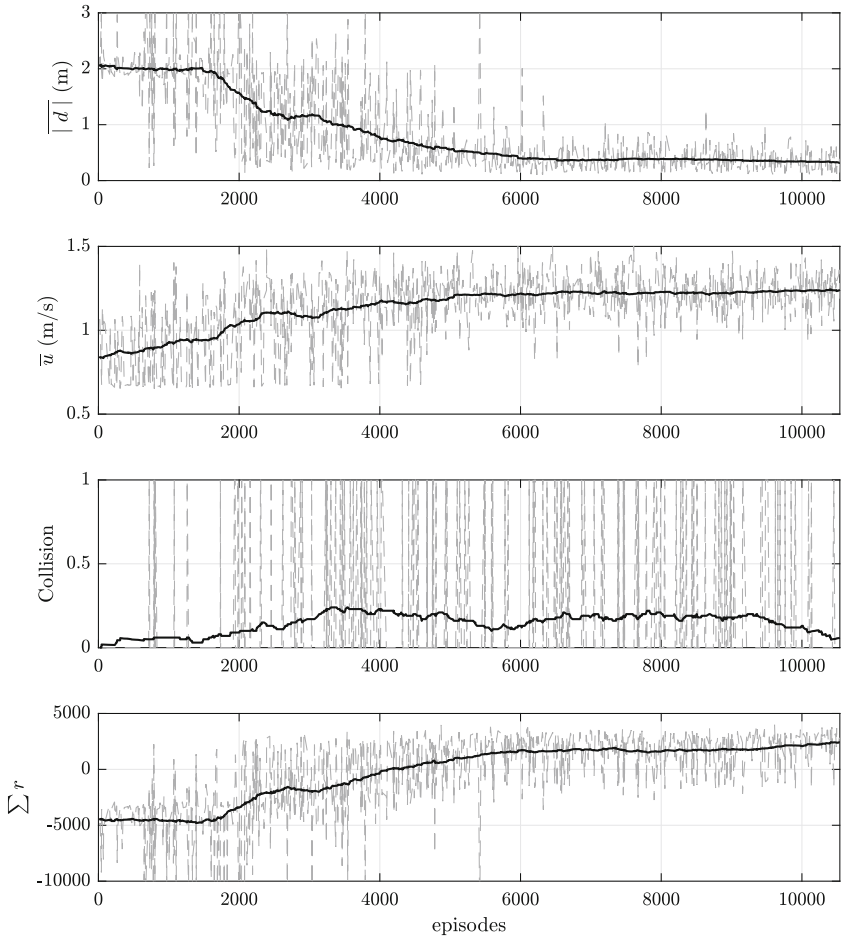

Fig. 11 Average distance error, average velocity, a boolean parameter that indicates if a collision occurred or not (1-collision, 0-no collision) and accumulated reward on each episode during training phase of the OA agent; gray dashed lines are real values and black lines are a 1000-episodes moving average

the lemniscate path without any obstacle. The table shows the average cross-track error, the total time to perform the lap on the path and the average velocity of the vehicle. The PF agent alone achieves a slightly better performance in terms of path distance error. Therefore, it is shown that having the OA agent can affect the PF performance somehow.

Next, the proposed approach is tested performing a full lap of lemniscate with an obstacle at different positions (centred on the path and not centred). Figure 12 shows the trajectory in the xy plane performed by the proposed approach in some of these simulations. In this figure and the rest of the figures of this paper, the initial position of the vehicle is marked with a red arrow pointing to the initial x-body orientation. From left to right and top to bottom, these figures correspond to the simulations with the obstacle placed at $\gamma_{obs} = 1.2$, $\gamma_{obs} = 2.4$, $\gamma_{obs} = 1.6$ and $\gamma_{obs} = 4.4$, respectively. These results show diverse examples of obstacle avoidance manoeuvres: at the long straight line, at the short straight line, at left-sided curves and at right-sided curves.

Table 3 summarizes the results presented in Fig. 12. This table shows the path position of the obstacle, given by γ_{obs}, and the distance from the center of the obstacle to the path, d_{obs}, and presents the same parameters evaluated in Table 2, that is, the average cross-track error, the total time and the average velocity.

In the simulation results presented in Fig. 12, the OA agent always chooses the most convenient side of the obstacle to avoid it. Nevertheless, occasionally, this agent may also choose the wrong side of the obstacle, as it is evident in the simulation of Fig. 13. In this simulation, the obstacle is located on the left of the path at 1.5m, however, this agent unnecessarily avoids it taking the longest route.

Table 2 Simulation results for one lap on the lemniscate path with no obstacles

	$\overline{d}$ (m)	time (s)	$\overline{\|\boldsymbol{v}\|}$ $(^m/_s)$
PF Agent	0.0765	44.6	1.3798
OA Agent + PF agent	0.1191	46.2	1.3223

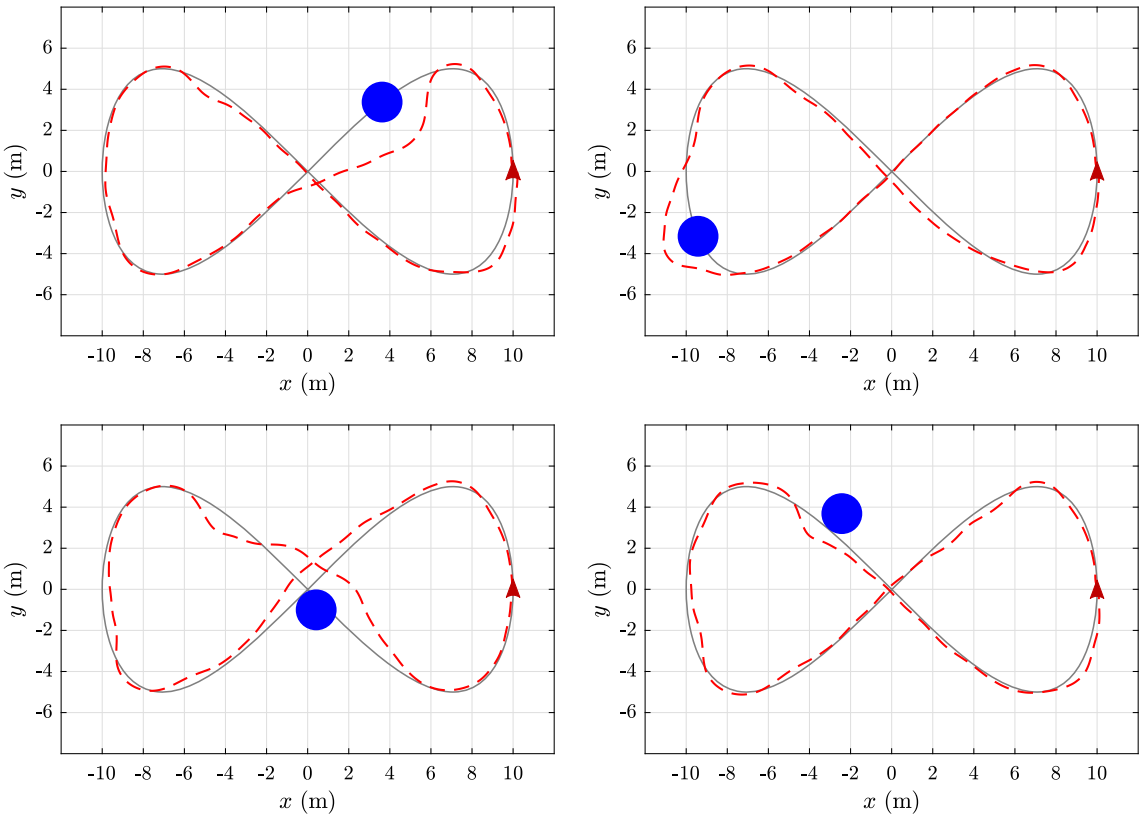

Fig. 12 Simulation results following the lemniscate path. The two figures above have the obstacle centred in the path while in the figures below the obstacle is out of the path

This behaviour may be caused by the fact that when the vehicle sees the obstacle for the first time, it is at its right. Therefore, as this agent cannot predict the upcoming right-sided curve (it only has local information of the path), it starts the avoidance manoeuvre by the left side of the obstacle. Nevertheless, even in the scenario of Fig. 13, this behaviour only happens in very few occasions.

Sometimes, the OA agent can also drive the vehicle to a collision, as shown in Fig. 14. Colliding is really a very rare event and it is more likely to happen on the starting part of the path. That is, when the vehicle is still accelerating and an obstacle suddenly appears in the LIDAR's field of vision.

Table 3 Simulation results for one lap on the lemniscate path with an obstacle

γ_{obs}	d_{obs}	$\bar{d}$ (m)	time (s)	$\overline{\|\boldsymbol{v}\|}$ ($^m/_s$)
1.2	0	0.3839	55.4	1.1265
2.4	0	0.2902	50.6	1.2514
1.6	1	0.3969	56.5	1.0923
4.4	1	0.2124	55.3	1.1235

In addition to the simulations presented in this section, the approach was tested following the same lemniscate path with obstacles in different positions all around the path, either centred or at different distances from it. In more than the 99% of these simulations the vehicle was able to properly avoid the obstacle, having an average distance error

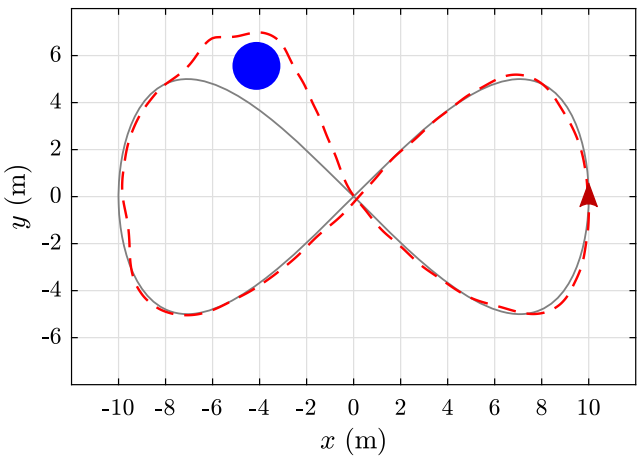

Fig. 13 Trajectory on the xy plane of a simulation where the OA agent avoids the obstacle by the wrong side

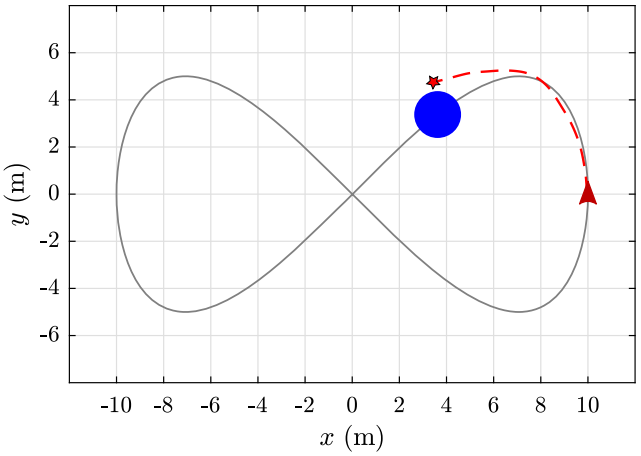

Fig. 14 Trajectory on the xy plane of a simulation that ended in a collision

in each simulation between 0.2 and 0.4m and an average velocity between 1.1 and $1.25^{m}/_{s}$. Having only problems with some obstacles that are located at $\gamma_{obs} \geq 1$ because the OA agent was not trained to avoid obstacles in the transient phase.

7.2 Spiral Path

In this section the PF+OA approach is tested with a spiral path, a path never seen before in the training phase of either the PF agent nor the OA agent. The spiral path is defined in Eq. 12, where A is set to 1.25 and γ ranges from 0 to 3π. Again, the vehicle starts at the initial point on the path, oriented tangentially to it.

$$\begin{aligned} x_d &= -A\gamma \cos(\gamma) \\ y_d &= A\gamma \sin(\gamma) \end{aligned} \tag{12}$$

The trajectories in the xy plane of the proposed approach when following the spiral path with different obstacle positions are shown in Fig. 15. From left to right and top to bottom, these simulations correspond to $\gamma_{obs} = 5.6, d_{obs} = 0$; $\gamma_{obs} = 8, d_{obs} = 0$; $\gamma_{obs} = 4, d_{obs} = 2$ and $\gamma_{obs} = 8.8, d_{obs} = -1.5$, respectively. Note that in a spiral path

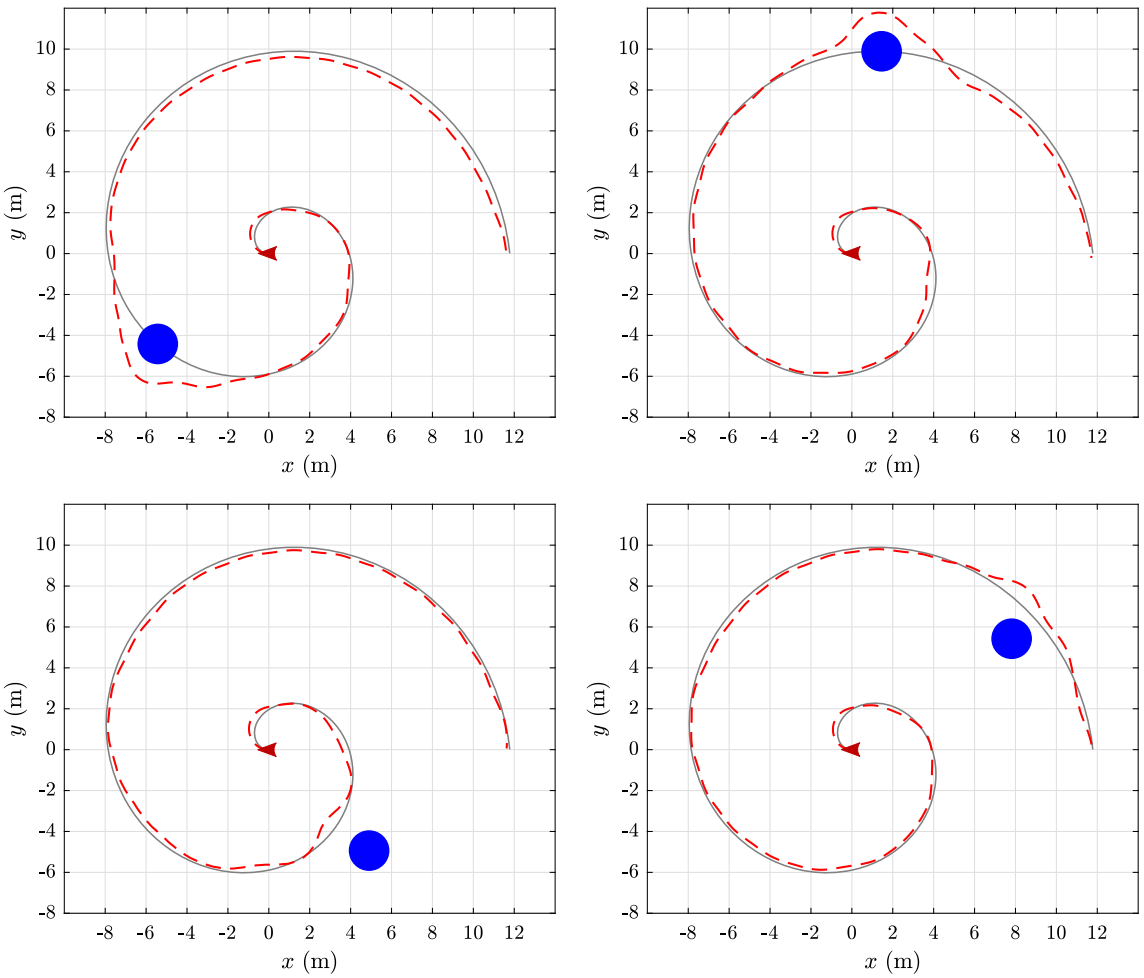

Fig. 15 Simulation results following the spiral path with different obstacle positions

Table 4 Simulation results for one lap on the spiral path with different obstacle positions

γ_{obs}	d_{obs}	$\bar{d}$(m)	time (s)	$\overline{\|\boldsymbol{v}\|}$ ($^m/_s$)
5.6	0	0.3768	49.3	1.1954
8	0	0.3216	52.1	1.1347
4	2	0.1835	48.8	1.1696
8.8	–1.5	0.1779	48.8	1.1951

the preferred side to avoid the obstacle is the outside of the curve. However, if the obstacle is placed shifted to the left of the path from the vehicle's point of view, the OA agent will avoid the obstacle from the inside of the curve, as is the case of the simulation with $\gamma_{obs} = 8.8$, $d_{obs} = -1.5$ (bottom-left in Fig. 15).

The results obtained in the simulations presented in Fig. 15 are shown in Table 4. Other simulations with obstacles located at different positions of the spiral path were also carried out. In these simulations the approach achieves an average cross-track error in each simulation between 0.3 and 0.4m, very similar to the results obtained with the lemniscate path.

The proposed approach is shown, with the previous results, to avoid obstacles following a path different from the trained one. Thus, it is shown how a generalized solution of the problem for different path shapes is achieved.

7.3 Multiple Obstacles

The proposed approach is trained with single cylindrical obstacles of fixed radius. In this section it is evaluated with different obstacle's configurations. That is, with multiple obstacles, cluttered environments, obstacles of different sizes and obstacles of different shapes. The trajectory on the xy plane following a straight line path of 30m with different obstacle configurations is shown in Fig. 16. These simulations and other simulations with similar obstacle's configurations resulted in a successful performance.

7.4 Algorithm Evaluation and Comparison

Following the comparison methodology applied in [9], the algorithm developed in this paper is compared with the ones of the literature review and the result of the comparison is summarized in Table 5. Only reactive algorithms are considered. The evaluation metrics are: velocity constraint (VC), which considers the velocity of the obstacle in the solution; Static and Dynamic Environment (SDC), which is true when obstacles are allowed to move; Swarm Compatibility (SC), which includes some sort of communication means to tackle this issue; Multiple Obstacles (MO), true when the algorithm can handle more than one obstacle; Different Size Obstacles (DSiO) and Different Shape Obstacles (DShO), true when the algorithm can handle obstacles of apparent different sizes and shapes, respectively.

It is difficult to compare a new algorithm with the ones available in the literature since several details condition the comparison. Each paper presents a setup (vehicle speed, sensors, type and dynamics of obstacles, etc.) in which its algorithm avoids the obstacles. Nevertheless, it is interesting to compare the conditions of the present paper's results with the available from the literature review. The usual speeds range between 0.7m/s and 3.5m/s [32–34] while the mean speed in the experiments of this paper is around 1.2m/s. [19, 21, 32] use DRL to solve the avoidance problem, their sensory setup is similar to the one used here but the control structure is different. They use an end-to-end solution, providing the references to the attitude controller while the agent presented in this paper is only responsible for modifying the reference trajectory. In [32] trajectories other than those used for the training are successfully tested like in this paper with the spiral trajectory while some authors focus on straight trajectories [33–35]. Finally, the number of obstacles, the size, shape and dynamics is richer in works like [33, 35] while such study remains as future work in the present paper.

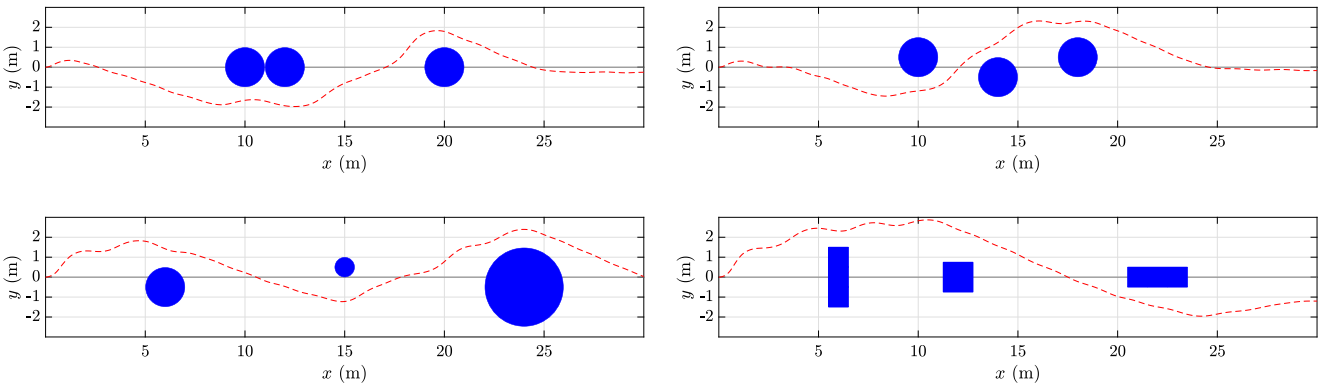

Fig. 16 Simulation results following a straight path with multiple obstacles, with different shapes and sizes

Table 5 Comparison between the DDPG OA approach and state of the art approaches). (∗ denotes metric not explicitly stated in the paper or unclear)

	VC	SDC	SC	MO	DSiO	DShO
[32]	×	✓	×	✓	×	✓
[33]	✓	✓	×	×	∗	∗
[34]	×	✓	×	×	✓	✓
[19]	×	×	×	✓	✓	✓
[21]	×	✓	×	✓	✓	✓
[35]	✓	✓	✓	✓	×	×
Own	×	×	×	✓	✓	✓

8 Conclusions

The aim of this paper is to develop a deep reinforcement learning solution for the path following and obstacle avoidance problem. To this end, a novel structure formed by two agents; the OA agent developed in this paper and the PF agent developed in [6], is proposed. The main novelty of the approach is that the OA agent computes an action that directly modifies the state of the PF agent, converting it into a cascade structure. Specifically, when an obstacle is detected on the vehicle's route the OA agent modifies the path distance error state of the PF agent, which at the same time modifies the reference path to avoid a possible collision. This structure results in many advantages, such as being more interpretable, reducing the training time and permitting to train it safely in the real platform.

A reactive obstacle avoidance agent, based on the application of the *Deep Deterministic Policy Gradient* algorithm, is developed. It receives the data of the obstacle detection system, which consists on a low-cost LIDAR sensor treated to eliminate ground detections. This paper explores different solutions to solve this problem and a detailed explanation is given on how the action, the state vector and reward function are obtained. The main issue addressed in this work is caused by the fact that the low-cost LIDAR sensor has a limited field-of-view of 48° which compromises the detection of an obstacle. This paper proposes a solution to this problem that consists on including information of the historical data of the LIDAR sensor in the state vector. This historical information is generated by integrating the inverse of the range data provided by each LIDAR beam. If no obstacles are near, this state is progressively decreased until it reaches zero. In this way, if the integral LIDAR state is positive, it means that an obstacle has been recently seen. The closer the obstacle is to the vehicle, the larger this state will be. This state is divided in two; one for each side of the LIDAR sensor, providing more information to the agent to know in what side of the vehicle the obstacle is. Its performance is compared with state of the art approaches using a table where the evaluation metrics of the different algorithms are presented. The mean speed in the experiments is within the interval of speeds reported in the literature.

The OA agent is trained in the RotorS/Gazebo environment with the model of the Asctec Hummingbird vehicle. This agent is programmed in pyhton by using the tensorflow library to generate and train the neural networks. A cylindrical obstacle of 1m radius is used in both the training phase and the tests. This agent is trained by following a half asymmetrical lemniscate path with random radius and with obstacles placed randomly on the path in a range of $\pm 2m$ of distance to the path.

The path following and obstacle avoidance approach is tested in the RotorS environment by following a lemniscate path and a spiral path with obstacles at different positions. Moreover, the performance of the approach is evaluated with different obstacle's configurations, such as multiple obstacles, cluttered environments and obstacles of different sizes and shapes. The simulations results show how it successfully avoids obstacles while following a path. The OA agent is able to choose the side to better perform the avoidance manoeuvre depending on the obstacle and vehicle positions and the path's shape. This agent takes into account the vehicle's velocity and the path's shape to generate this manoeuvre. The path following performance without obstacles is very similar to the one obtained by the PF agent. In conclusion, a generalized solution for the path following and obstacle avoidance problem for different path's shapes and obstacle configurations is achieved.

The approach presented in this paper is an initial framework where the main purpose is to solve the PF and OA problem in the simplest manner, extending the work made in [6] with the PF agent, and proving that this problem can be solved by means of DRL. That is the main reason of some of the decisions that were made, as for instance solving the problem with two agents or creating new states to provide historical obstacle information to the OA agent. Nevertheless, the aim is to keep improving this approach to make it more robust and proficient. Therefore, future work is to modify the structure of the *DDPG* algorithm to give the algorithm the intrinsic ability of using historical data. This can be done by using recurrent neural networks, for instance *LSTM* networks, which use a sequence as input data. Other future work is to implement a unique agent for solving the PF and OA problem and compare it with the presented approach. To this end, the state information and reward functions of both the developed PF and OA agents could be combined to generate the new PF and OA agent. Even this would increase the difficulty of the problem, and hence, the amount of training time needed to converge, it would be interesting to see if better results are obtained. Future work is also to obtain an approach that achieves a generalized

solution for different obstacle shapes and scenarios. That may include providing more information of the obstacle to the OA agent as well as training it with different types of obstacles and in multiple obstacle scenarios. In any case, next immediate step is to implement the resulting approach in the real experimental platform. Both the PF and OA agents were already integrated in ROS and the LIDAR system was already installed in the real platform where the PF agent alone has already successfully tested. Therefore, next step would be to build a safe experimental framework containing obstacles, as well as creating a safety protocol to avoid possible collisions.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Declarations

Conflict of Interests The authors declare that they have no conflict of interest.

References

1. Caicedo, J.C., Lazebnik, S.: Active object localization with deep reinforcement learning, in. In: 2015 IEEE International Conference on Computer Vision (ICCV), pp. 2488–2496 (2015)
2. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T.P., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of go with deep neural networks and tree search. Nature **529**, 484–489 (2016)
3. Yu, R., Shi, Z., Huang, C., Li, T., Ma, Q.: Deep reinforcement learning based optimal trajectory tracking control of autonomous underwater vehicle. In: 2017 36th Chinese Control Conference (CCC), pp. 4958–4965 (2017)
4. Tuyen, L.P., Chung, T.: Controlling bicycle using deep deterministic policy gradient algorithm. In: 2017 14TH International Conference on Ubiquitous Robots and Ambien Intelligence (Urai), pp. 413–417 (2017)
5. Li, L., Lv, Y., Wang, F.: Traffic signal timing via deep reinforcement learning. IEEE/CAA J. Automatica Sinica **3**(3), 247–254 (2016)
6. Rubí, B., Morcego, B., Pérez, R.: Deep reinforcement learning for quadrotor path following with adaptive velocity. Auton. Robot. **45**, 119–134 (2020)
7. Goerzen, C., Kong, Z., Mettler, B.: A survey of motion planning algorithms from the perspective of autonomous uav guidance. J. Intell. Robot. Syst. **57**(1), 65 (2009)
8. Huang, S., Teo, R.S.H., Tan, K.K.: Collision avoidance of multi unmanned aerial vehicles: A review. Ann. Rev. Control **48**, 147–164 (2019)
9. Yasin, J.N., Mohamed, S.A.S., Haghbayan, M., Heikkonen, J., Tenhunen, H., Plosila, J.: Unmanned aerial vehicles (uavs): Collision avoidance systems and approaches. IEEE Access **8**, 105139–105155 (2020)
10. Minguez, J., Lamiraux, F., Laumond, J.-P.: Motion Planning and Obstacle Avoidance. In: Springer Handbook Of Robotics, pp. 827–852. Springer, Berlin (2008)
11. Ma, Z., Wang, C., Niu, Y., Wang, X., Shen, L.: A saliency-based reinforcement learning approach for a uav to avoid flying obstacles. Robot. Auton. Syst. **100**, 108–118 (2018)
12. Eslamiat, H., Li, Y., Wang, N., Sanyal, A.K., Qiu, Q.: Autonomous waypoint planning, optimal trajectory generation and nonlinear tracking control for multi-rotor uavs. In: 2019 18TH European Control Conference (ECC), pp. 2695–2700 (2019)
13. Yang, S., Meng, Z., Chen, X., Xie, R.: Real-time obstacle avoidance with deep reinforcement learning three-dimensional autonomous obstacle avoidance for uav. In: Proceeding of the 2019 International Conference on Robotics, Intelligent Control and Artificial Intelligence (2019)
14. Wu, K., Esfahani, M.A., Yuan, S., Wang, H.: Depth-based obstacle avoidance through deep reinforcement learning. In: Proceedings of the 5th International Conference on Mechatronics and Robotics Engineering, pp. 102–106 (2019)
15. Yan, C., Xiang, X., Wang, C.: Towards real-time path planning through deep reinforcement learning for a uav in dynamic environments. J. Intell. Robot. Syst **98**, 297–309 (2020)
16. Han, X., Wang, J., Xue, J., Zhang, Q.: Intelligent decision-making for 3-dimensional dynamic obstacle avoidance of uav based on deep reinforcement learning. In: 2019 11th International Conference on Wireless Communications and Signal Processing (Wcsp), pp. 1–6 (2019)
17. Cheng, Y., Zhang, W.: Concise deep reinforcement learning obstacle avoidance for underactuated unmanned marine vessels. Neurocomputing **272**, 63–73 (2018)
18. Lu, X., Cao, Y., Zhao, Z., Yan, Y.: Deep reinforcement learning based collision avoidance algorithm for differential drive robot. In: Intelligent Robotics and Applications ICIRA 2018 (2018)
19. Li, B., Wu, Y.: Path planning for uav ground target tracking via deep reinforcement learning. IEEE Access **8**, 29064–29074 (2020)
20. Wang, C., Wang, J., Shen, Y., Zhang, X.: Autonomous navigation of uavs in large-scale complex environments: A deep reinforcement learning approach. IEEE Trans. Veh. Technol. **68**(3), 2124–2136 (2019)
21. Sampedro, C., Bavle, H., Rodriguez-Ramos, A., de la Puente, P., Campoy, P.: Laser-based reactive navigation for multirotor aerial robots using deep reinforcement learning. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1024–1031 (2018)
22. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M.A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)
23. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167 (2015)
24. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep

reinforcement learning. In: 2016 International Conference on Learning Representations (ICLR) (2016)
25. 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, pp. 387–395 (2014)
26. Rubí, B., Ruiz, A., Pérez, R., Morcego, B.: Path-flyer: A benchmark of quadrotor path following algorithms. In: 2019 15th IEEE International Conference on Controol and Automation (2019)
27. Aguiar, A.P., Hespanha, J.P.: Trajectory-tracking and path-following of underactuated autonomous vehicles with parametric modeling uncertainty. IEEE Trans. Autom. Control **52**(8), 1362–1379 (2007)
28. Cabecinhas, D., Cunha, R., Silvestre, C.: Rotorcraft path following control for extended flight envelope coverage. In: Proceedings of the 48th IEEE Conference on Decision and Control, Held Jointly with the 28th Chinese Control Conference (CDC/CCC 2009), pp. 3460–3465 (2009)
29. Aguiar, A.P., Hespanha, J.P., Kokotovic, P.V.: Performance limitations in reference tracking and path following for nonlinear systems. Automatica **44**(3), 598–610 (2008)
30. Kaminer, I., Yakimenko, O., Pascoal, A., Ghabcheloo, R.: Path generation, path following and coordinated control for time-critical missions of multiple UAVs. In: Proceedings of the American Control Conference, vol. 1–12, p. 4906 (2006)
31. Furrer, F., Burri, M., Achtelik, M., Siegwart, R.: RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. Springer International Publishing, Cham (2016)
32. Stevšić, S., Nägeli, T., Alonso-Mora, J., Hilliges, O.: Sample efficient learning of path following and obstacle avoidance behavior for quadrotors. IEEE Robot. Autom. Lett. **3**(4), 3852–3859 (2018)
33. Park, J., Cho, N.: Collision avoidance of hexacopter uav based on lidar data in dynamic environment. Remote. Sens. **12**, 6 (2020)
34. Al-Kaff, A., García, F., Martín, D., de la Escalera, A., Armingol, J.M.: Obstacle detection and avoidance system based on monocular camera and size expansion algorithm for uavs. Sensors (Basel, Switzerland) 17 (2017)
35. Amrouche, M., Marinho, T., Stipanović, D.: Vision based collision avoidance for multi-agent systems using avoidance functions. In: 2020 European Control Conference (ECC), pp. 1683–1688 (2020)

Bartomeu Rubí received his PhD degree in Automatic Control, Robotics and Computer Vision from the Universitat Politècnica de Catalunya (UPC) in 2020. Before that, he received the M.S. degree in Automatic Systems and Industrial Electronics Engineering from the UPC in 2015 and the B.S. degree in Industrial Electronics Engineering from the Universitat de les Illes Balears in 2013. Currently, he is working at the Robotics, Vision and Control Research Laboratory (University of Seville) as a postdoctoral researcher. The main research topic is focused on automatic control and machine learning applied to UAVs.

Bernardo Morcego is an Associate Professor at the Universitat Politècnica de Catalunya (UPC). He received a PhD degree in Computer Science from the UPC in 2000. He has been teaching several subjects in the area of automatic control in the schools of Engineering and Aeronautics in Terrassa and Barcelona. He is a member of the Research Center for Supervision, Safety and Automatic Control of UPC. His research interests include UAV control systems, intelligent control and computer vision applications.

Dr. Ramon Pérez. Degree and PhD in Physical Sciences by the Universitat de Barcelona (UB) in 1993 and Universitat Politècnica de Catalunya (UPC) in 2003 respectively. He has been lecturing in UPC since 1994. He has long experience in teaching modelling and control of dynamic systems. His research has been focussed on modelling, simulating, control and supervising water systems and UAV. His research and teaching also includes data analysis.