A Deep Reinforcement Learning Approach for Path Following on a Quadrotor

Bartomeu Rubí[1]* and Bernardo Morcego[1] and Ramon Pérez[1]

***Abstract*— This paper proposes the *Deep Deterministic Policy Grandient* (DDPG) reinforcement learning algorithm to solve the path following problem in a quadrotor vehicle. This agent is implemented using a separated control and guidance structure with an autopilot tracking the attitude and velocity commands. The *DDPG* agent is implemented in python and it is trained and tested in the RotorS-Gazebo environment, a realistic multirotor simulator integrated in ROS. Performance is compared with *Adaptive NLGL*, a geometric algorithm that implements an equivalent control structure. Results show how the *DDPG* agent is able to outperform the *Adaptive NLGL* approach while reducing its complexity.**

I. INTRODUCTION

The goal of this paper consists in developing an artificial intelligent agent to solve the path following problem. The agent must be capable of learning online from real experimental tests and must simplify the training process of [1]. This approach will follow the control structure of geometric algorithms. The agent must be able to work with continuous state-action spaces and also be portable to other multirotor vehicles.

The objective of the path following problem, as its name suggests, is to make a system follow a pre-defined path in space. The main difference of this approach with the trajectory tracking problem is that path following eliminates the time dependence of the path reference, resulting in many advantages [2][3][4]. In this paper, the system of study is an Unmanned Aerial Vehicle (UAV), precisely a quadrotor vehicle. Several path following techniques have been implemented on UAV systems, such as *Backstepping* [5], *Feedback Linearisation* [6], *Model Predictive Control* [7], geometric algorithms, etc. Control-oriented algorithms rely on an accurate model of the UAV, while geometric algorithms depend on the shape of the path.

Geometric path following algorithms were initially described on the missile guidance and control literature, but some of these algorithms have been adapted to UAVs. This is the case of *Nonlinear Guidance Law* (NLGL) [8], *Carrot-Chasing*, *Pure Pursuit* [9] or *Trajectory-Shaping* [10] algorithms. These algorithms are simple, they typically provide feasible solutions with considerably good performance [8] and they have very few parameters to tune. These parameters generally depend on three variables: the vehicle's velocity, the path's curvature and the inner dynamics of the vehicle. This results in the main disadvantage of the geometric algorithms since, if the velocity reference of the vehicle or the path's shape are modified, it is necessary to tune those parameters again to maintain the best possible performance.

In [1] we proposed an adaptive approach of the *NLGL* algorithm. In this approach a neural network (NN) was used to compute the optimal algorithm parameters for any given vehicle's velocity and radius of a path's curve. The data set used to train the NN was obtained from multiple simulations with a quadrotor model following different types of paths in different conditions. Thus, the training process was made offline. Simulation results proved the validity of this approach, which is able to outperform the standard *NLGL*. The main drawback of this approach is that its performance depends on the accuracy of the model used to train the neural nets. Also, if the quadrotor vehicle or the dynamics of the attitude controller are changed, it is necessary to train the NN again. This process involves performing a large number of simulations, extracting the information of interest and adjusting the data [1], therefore, it can become tedious.

The authors considered the emerging field of deep reinforcement learning to be suited for the presented problem. Reinforcement learning (RL), along with deep learning, is becoming a wise solution for a large number of different and complex problems. That is, due to the significant progress made in this field, RL is no longer constrained to discrete and small environments. In [11] the *Q-learning* RL algorithm was combined with deep neural networks to create the *Deep Q-Network* (DQN) algorithm. In this approach the inputs of the RL agent are images. This method was successfully tested in several classic Atari games. Since then, *DQN* algorithm has gained a lot of popularity and has been used to solve problems as diverse as object localization [12], traffic signal timing [13] and in [14] it was combined with tree search algorithm to develop an agent capable to defeat the human European champion in the classic game GO.

Deep Deterministic Policy Gradient (DDPG) is another RL algorithm that is also being implemented with success on different environments [15][16]. This algorithm, based on the actor-critic concept, was first stated in [17]. The authors proved the validity of this approach, which was especially designed for continuous state-action spaces, testing it on diverse classic reinforcement learning environments such as the cart-pole problem, tasks involving contacts, locomotion tasks and other high dimensional tasks. *DDPG* has been also implemented on a quadrotor vehicle to solve the landing problem [18] with successful results.

*Corresponding author

[1] Authors are with the Specific Center of Research at CS2AC in the Universitat Politècnica de Catalunya (UPC). Rbla Sant Nebridi 22, Terrassa (Spain). email: tomeu.rubi, bernardo.morcego, ramon.perez at upc.edu

II. PROBLEM STATEMENT

The aim of this paper is to implement the *Deep Deterministic Policy Gradient* reinforcement learning algorithm to solve the path following problem on a quadrotor vehicle. This algorithm will substitute the geometric algorithm [1], preserving the same control structure.

In the path following problem, the path is stated in function of a scalar parameter known as the virtual arc, represented by γ_p in this paper. Eq. (1) shows the parametric definition of a generic three-dimensional path.

$$\mathbf{p}_d(\gamma_p) := [x_d(\gamma_p), y_d(\gamma_p), z_d(\gamma_p)]^T \tag{1}$$

Geometric algorithms implement a Separated Control and Guidance (SGC) structure. That means that path following algorithms calculate the desired velocity and/or orientation references of the vehicle and an inner controller, commonly known as the autopilot, is in charge of tracking these references and computing the inputs of the system.

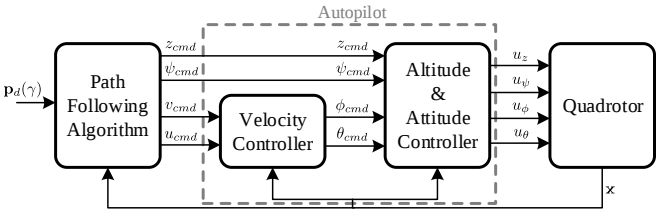

Fig. 1. Separate Guidance and Control structure.

The SGC structure implemented in this paper is shown in Fig. 1. The path following algorithm computes the altitude command (z_{cmd}), the angle command in z axis (ψ_{cmd}) and the velocity commands in x and y axis (v_{cmd} and u_{cmd}). The autopilot is formed by a velocity controller and an altitude & attitude controller. It computes the inputs of the system (u_z, u_ψ, u_ϕ and u_θ) in order to track the commands received by the path following algorithm.

III. THE DEEP REINFORCEMENT LEARNING ALGORITHM

Deep Deterministic Policy Gradient [17] is an actor-critic reinforcement learning algorithm that is model-free and off-policy. Model-free in the sense that it makes no effort to learn the dynamics of the environment, but estimates directly the optimal value function, and off-policy because it uses a policy to generate the action to compute the loss function that is different from the policy that is being improved.

The main idea of the actor-critic algorithms is that the policy is represented independently from the value function. A typical structure for actor-critic RL algorithms is shown in Fig. 2. The policy function ($\mu(s)$) is known as the actor and the value function ($Q(s,a)$) as the critic. The actor computes an action according to the current state of the environment, while the critic is in charge of estimating the value function given the state-action pair. Moreover, the critic computes the loss function or temporal-difference error (TD) and, generally, this TD is then used for the critic and the actor in the learning process. In deep reinforcement learning, actor and critic are approximated by neural networks.

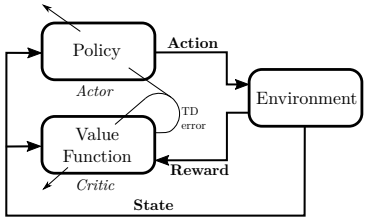

Fig. 2. Actor-Critic agent structure.

DDPG is based on the standard *Deterministic Policy Gradient* [19] and includes some characteristic concepts of *Deep Q-Network*, making the algorithm a deep reinforcement learning one. One of its major advantages, and the reason why this algorithm has been chosen in this work, is that it provides a good performance in large and continuous state-action space environments. *DDPG*, as well as *DQN*, stabilizes the learning of the Q-function by the use of two elements: the replay buffer and the target networks. More information about that in [17].

The weights of the critic and actor are updated from two different gradient functions as shown in Eq. (2) and Eq. (3), respectively. Where, s is the state, a it the action and r the reward, ϕ are the set of weights of the critic network and θ the weights of the actor, η_ϕ and η_θ are the learning rates of the critic and actor, B represents the minibatch of transition tuples and N its size. The prime symbol is used to represent a target network. The target Q-values (not to be confused with target networks) are represented by y_k (Eq. (4)) and are used to compute the loss function or TD error. The weights of the critic are updated to minimize this loss function. γ is the discount factor, a value between 0 and 1 that tunes the importance of future rewards to the current state.

$$\Delta\phi = \eta_\phi \nabla_\phi \left(\frac{1}{N} \sum_{i \in B} \left(Q(s_i, a_i \mid \phi^{Q'}) - y_i \right)^2 \right) \tag{2}$$

$$\Delta\theta = \eta_\theta \nabla_\theta \left(\frac{1}{N} \sum_{i \in B} Q(s_i, \mu(s_i \mid \theta^\mu) \mid \phi^Q) \right) \tag{3}$$

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} \mid \theta^{\mu'}) \mid \phi^{Q'}) \tag{4}$$

Note that the target Q-Values (Eq. (4)) are obtained from the outputs of the actor and critic target networks, following the target network concept.

Eqs. (5-6) show the equations used to update the weights of the target networks from the trained networks, where parameter τ indicates how fast this update is carried on. This soft update is made each step after training the main networks.

$$\phi^{Q'} \leftarrow \tau\phi^Q + (1-\tau)\phi^{Q'} \tag{5}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{6}$$

Fig. 3. Asctec Hummingbird Quadrotor (original image from wiki.asctec.de).

IV. DDPG FOR PATH FOLLOWING

This section explains the particularities of the *Deep Deterministic Policy Gradient* algorithm proposed in this paper to solve the path following problem and the environment that has been employed to train the RL agent.

A. Agent Environment

The vehicle employed in this work is the Asctec Hummingbird (Fig. 3), a quadrotor with a mass of 0.698 kg and a maximum airspeed of 15 $^{m}/_{s}$. To maintain the integrity of the experimental platform it is necessary to have a simulation environment to perform the first training steps safely. In this paper the simulation environment has been built on the Gazebo-ROS (Robot Operating System) platform, by means of the RotorS simulator [20].

RotorS is a UAV simulator programmed in the Gazebo environment. It provides different multirotor vehicle models, such as the Asctec Hummingbird. This simulator implements a complete model of the vehicle and its environment. Sensors and noise can be included too. Also, it has a graphic unit interface, inherent to gazebo, where the user can observe the vehicle's trajectory. One of the most important advantages of building this environment in ROS is that the same code of the agent can be transferred to the real Hummingbird platform, since it is also implemented in ROS.

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 real experimental results [21]. Moreover, real and simulated response were very similar which proves the reliability of the model.

The *DDPG* agent has been implemented in Python 3.5 by means of Tensorflow and Tflearn libraries. Since ROS is only able to work with version 2 of Python, the program cannot be run as a regular ROS node. That is, it can not be launched with the standard *rosrun* command or inside a *roslaunch* file, as the other nodes of the autopilot and the simulator. Nevertheless, *rospy* library is used in order to publish and subscribe to ROS topics.

This environment is integrated in a linux Xubuntu virtual machine with 8GB RAM and four 1.80GHz processors (i7-8550U CPU) dedicated. With these specifications, while training the agent, it is possible to work at ratios near to 1 between gazebo time and real time, that is, almost real time. However, if the PC is not well cooled the time ratio can decrease to a minimum of 0.8. This means that simulation time is slowed down to be able to perform all the operations at a given rate.

B. DDPG Proposed Approach

The path following algorithm, as shown in Fig. 1, computes four control commands (z_{cmd}, ψ_{cmd}, u_{cmd} and v_{cmd}). However, the DDPG agent will be only in charge of calculating the reference of yaw, i.e. the angle in z axis (ψ_{cmd}). As in [1], altitude (z_{cmd}) and velocity (u_{cmd}) commands are considered user specifications, and the velocity reference on the y axis (v_{cmd}) is fixed to 0. Nevertheless, the action obtained by the deep reinforcement learning agent is not directly the yaw angle command but a correction of it. That is, the action (a) is a desired correction over the current yaw angle. Thus, yaw command at step k is obtained as shown in Eq. (7), where Δt is the time of one step of the agent (correction is given in $^{rad}/_{s}$).

$$\psi_{cmd,k} = a_k \, \Delta t + \psi_k \tag{7}$$

The idea of choosing the correction action is because computing directly the yaw angle command results in undesired fast angle changes. On the other hand, using an incremental control action results in an unstable behaviour, since it is equivalent to adding a new integral to the plant. Thus, computing the correction action over the current yaw angle results in a smooth movement while keeping the stability of the system.

To enhance exploration of the *DDPG* agent an Ornstein–Uhlenbeck noise is added to the action at training time. The Ornstein–Uhlenbeck noise function is stated in Eq. (8), where n_k is the value of the noise at the kth iteration, θ_n is a parameter related to the speed rate of mean reversion, μ_n is the drift term which affects the asymptotic mean, Δt is the time step and dW_t is the standard Wiener process scaled by volatility σ_n. During training, the ψ_{cmd} command will be computed as shown in Eq. (9), where noise is added to the obtained agent action. The power of this noise is reduced with the number of episodes (j) in such a way that exploring rate is reduced as the agent keeps learning. Parameter κ indicates the speed transition between exploration and exploitation.

$$n_k = n_{k-1} + \theta_n \left(\mu_n - n_{k-1}\right) \Delta t + \sigma_n dW_t \tag{8}$$

$$\psi_{cmd,k} = \left(a_k + \frac{n_k}{j/\kappa + 1}\right) \Delta t + \psi_k \tag{9}$$

The state vector of the proposed *DDPG* agent is formed by two states, the distance error (e_d) and the angle error (e_ψ). These errors are calculated with respect to the path's closest point to the vehicle and the vehicle. The state vector equations are shown in Eq. (10), where y_T represents the y coordinate of the vehicle position with respect to the path tangential frame of reference, and ψ_T represents the yaw angle of the vehicle referred to the same frame. This tangential frame of reference is placed on the path closest point to the vehicle with x pointing to the tangential direction, z pointing up and y pointing to the resultant direction of $x \times z$.

$$s = \{e_d, \, e_\psi\} \;\; | \;\; e_d = y_T, \;\; e_\psi = \psi_T \tag{10}$$

Several types of rewards (r) were tested (i.e. continuous or discrete, penalizing bad behaviour or rewarding good path following performance, and mixed strategies), being the reward defined in Eq. (11) the one that achieves the best performance and fastest convergence. The term $-20|e_d|$ is for penalizing the distance error from the vehicle to the path. v_T is the velocity of the vehicle projected to the path tangential frame of reference. This last term gives a positive reward if the vehicle is moving forward on the path and a negative reward if it is moving opposite of the desired direction.

$$r = -20|e_d| + 10v_T \tag{11}$$

The structure proposed for the actor neural network is formed by two feed-forward hidden layers of 400 and 300 neurons, respectively. Both layers have Rectified Linear Unit (ReLU) as activation function and use the batch normalisation technique [22] for a faster convergence. The structure of the critic network is also composed by two feed-forward hidden layers of 400 and 300 neurons, but in this case the state input vector is connected to the first hidden layer while the action vector is connected directly to the second hidden layer. That is, action input skips the first layer, which has been proved to be beneficial. Again, all layers have ReLU as activation function and, this time only the state input layer of the critic uses the batch normalisation technique.

The relevant parameters of the proposed *DDPG* agent and its values are shown in table Table I.

TABLE I
PARAMETERS OF THE *DDPG* 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
-	Maximum steps of one episode.	300
Δt	Agent time step.	0.1 s
κ	Ratio of exploration-exploitation transition.	200

The defined state vector along with the stated network structure, reward and the rest of the algorithm's parameters make the proposed *Deep Deterministic Policy Gradient* algorithm able to solve the path following problem properly (see Sec. V). Moreover, it is important to mention that this is the best agent setup with only two states, in terms of PF performance, that we were able to find among numerous and diverse agent setups that were tested. Nevertheless, notice that this state vector (Eq. (10)) only gives instantaneous information about the path since states are referred to the closest point on the path to the vehicle. That is, the agent does not have any information of the path shape to come and that makes it impossible for the agent to anticipate the curves.

To cope with this issue we propose another state vector that, in addition to the two states defined before, includes states of errors of future points in the path. Specifically, it is proposed to use the angle error between the vehicle and the tangential frame, but in this case not placed in the closest path's point to vehicle but forward points on the path. That is, the agent receives information of future orientations of the path with respect to the vehicle and it is possible for it to correct the vehicle's angle to anticipate future curves. With this approach, new design factors appear, such as the number of future error points to be included in the state and the distance of those points. This issue is analysed in Sec. V, proving that adding at least one future orientation error state improves substantially the path following performance of the *DDPG* agent.

V. RESULTS

This section shows and analyses the results obtained in the training and the testing phase of the agent. Both phases are performed in RotorS-Gazebo environment with Hummingbird quadrotor. Sensor noise and wind disturbances were not included.

A. Training

A lemniscate path has been used to train the agents. This path is defined in Eq. (12), where A is fixed to 4m and γ_p ranges from 0 to 2π, corresponding to a half lemniscate path. The path has been discretized with a precision of 0.01m between each path point. The velocity command (u_{cmd}) is $1^m/_s$ and the altitude (z_{cmd}) command is 1m.

$$\begin{aligned} x_d &= 2A\cos(\gamma_p) \\ y_d &= A\sin(2\gamma_p) \end{aligned} \tag{12}$$

The learning evolution of the agent described is shown in Fig. 4. This agent corresponds to the version of only two states: the distance error and the angle error. This figure shows the average distance error ($\overline{|d|}$) and the accumulated reward ($\sum r$) in each episode during training phase. As observed, $\overline{|d|}$ decreases over episodes and r increases, and both converge around the 120th episode. The value of convergence is around 0.1m and $2,000$, respectively.

It is important to mention that the training process has a stochastic component. Thus, with the same agent parameters the convergence can be obtained after different number of episodes. In the case of this particular agent, the convergence is typically obtained between episode 70 and 150. However, the values of average distance error and accumulated reward at which it converges are almost the same in each case.

As shown in the zoomed area of figure Fig. 4 the learning performance starts getting worse around episode 350. This is due to the overfitting effect, that makes the agent to learn features that are specific of each episode and are not generalizable to solve the PF problem.

Fig. 5 shows the learning process followed by an agent with 4 extra states of angle errors in points forward on the path. This new 4 angle error states are computed at distances of 40cm, 60cm 80cm and 100cm to the closest point to the

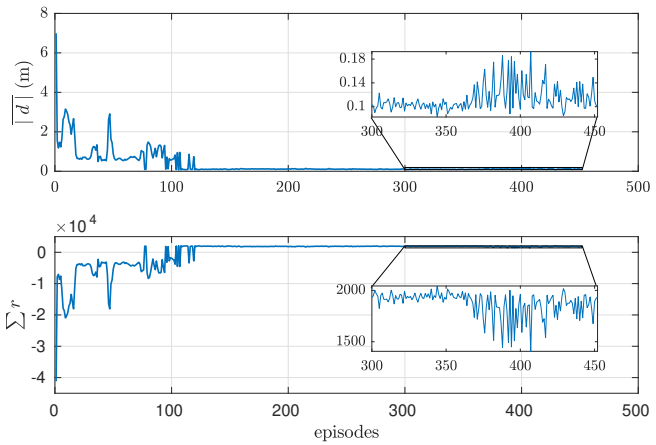

Fig. 4. Average distance error and accumulated reward on each episode during training phase of agent with 2 states.

path, respectively. Again, the agent is trained with the half lemniscate path defined in Eq. (12).

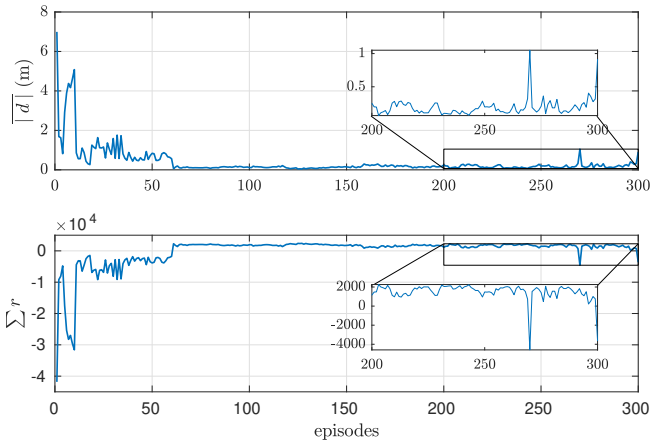

Fig. 5. Average distance error and accumulated reward on each episode during training phase of agent with 2 standard states plus 4 states of future angle errors.

Apparently, the agent with 2+4 states converges around episode 60. Nevertheless, the zoomed area of Fig. 5 denotes that this agent does not converge in a stable manner since the values of the average distance error and the accumulated reward do not remain stable. This issue is produced by underfitting. It has been verified that adding more neurons (+50 neurons) in the first layer of the critic and actor networks (and their respective target networks) makes the learning process stable, and the agent outperforms the standard 2 states agent.

Summarizing, the stated agent is able to have a stable convergence with a maximum of 5 states (2 standard states plus 3 extra states of future angle errors). But, with more than 3 future angle error states the convergence of the system with the given NN structures is almost impossible to achieve. However, as mentioned before, this problem is solved by adding more neurons in the first layer of the neural networks.

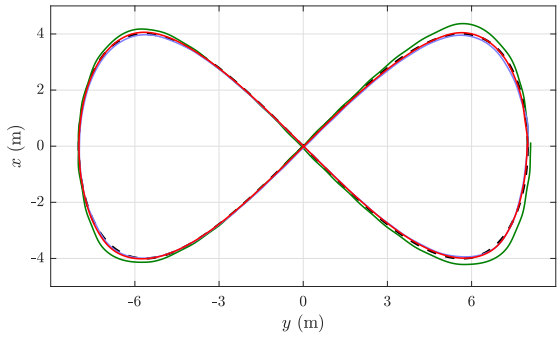

Fig. 6. Trajectory on xy of *DDPG* agent: 2 states in green, 2+1 states in red and 2+2 states in blue. (Lemniscate path, $u_{cmd} = 1^m/_s$)

B. Test

Several agents with different sets of states were trained and then the ones that achieved a stable convergence of learning were saved. Agents are saved when the average distance error achieves its lower value during training such that overfitting issues are avoided. The most prominent saved agents of each set of states are tested in this section.

Fig. 6 shows the trajectory on the xy plane of three different *DDPG* agents while following a full lap of Lemniscate path of $A = 4m$. That is, the agents are tested with the same path that was used to train them. Agents correspond to the standard 2 states agent, in green color in the figure, the 2+1 states agent in red and the 2+2 states agent in blue. Future angle error state of 2+1 agent is at 60cm of the closest path point, which has been proved to be the best distance to place this state at the particular velocity of $1^m/_s$. In the 2+2 states agent the future angle errors are placed at distances of 60cm and 40cm.

The results of Fig. 6 are summarized in Table II, where the average distance to the path, the total time to perform the full lap and the average velocity of the vehicle are presented. Results obtained with the Adaptive NLGL of [1] (without velocity reduction term) are also included. The 2 states standard agent is able to follow the path properly (average distance error of around 10cm), however it presents errors of almost 40cm in the curves. That is because this agent does not have any information to predict those curves. The agents that have information of future angle errors are able to reduce significantly the distance error. The Adaptive NLGL performs slightly better that the 2 states agent, but it is not able to achieve the performance of the agents with future states.

TABLE II

RESULTS FOR ONE LAP ON THE LEMNISCATE PATH ($V_{ref} = 1^m/_s$).

	$\overline{\|\mathbf{d}\|}$ (m)	time (s)	$\overline{\|\|\mathbf{v}\|\|}$ ($^m/_s$)
DDPG 2 States	0.1041	67.10	0.8707
DDPG 2+1 States	0.0189	56.79	0.8620
DDPG 2+2 States	0.0308	55.86	0.8552
Adaptive NLGL	0.0708	55.56	0.8862

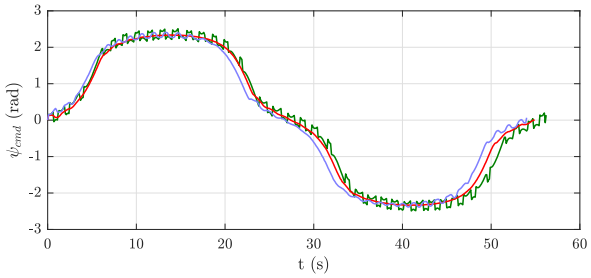

Fig. 7. ψ_{cmd} Lemniscata.

The computed yaw command (ψ_{cmd}) by each tested DDPG agent while following the lemniscate path is shown in Fig. 7. As observed, 2 states and 2+2 states agents present oscillations, while the command obtained by the 2+1 states agent exhibits a clean and stable behaviour, which may explain the results of Table II.

The trajectory performed by the three agents while following a spiral path (Eq. (13), with $A = 4$) is shown in Fig. 8. Again, results are summarized in Table III, which also includes the results of the *Adaptive NLGL* algorithm. Results show how the 2 states agent presents almost the double of distance error than the other two agents. Once again, *Adaptive NLGL* is able to outperform the standard *DDPG* agent, but the agents that receive future path information perform slightly better.

$$\begin{aligned} x_d &= A\gamma_p \cos(\gamma_p) \\ y_d &= A\gamma_p \sin(\gamma_p) \end{aligned} \tag{13}$$

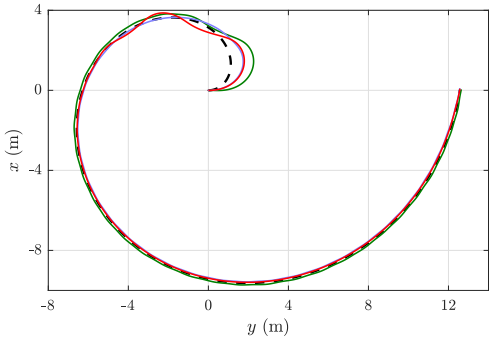

Fig. 8. Trajectory on xy of *DDPG* agent: 2 states in green, 2+1 states in red and 2+2 states in blue. (Spiral path, $u_{cmd} = 1^m/_s$)

TABLE III
RESULTS FOR ONE LAP ON THE SPIRAL PATH ($V_{ref} = 1^m/_s$).

	$\overline{\mathbf{d}}$ (m)	time (s)	$\overline{\|\mathbf{v}\|}$ ($^m/_s$)
DDPG 2 States	0.1651	52.60	0.8535
DDPG 2+1 States	0.0823	51.03	0.8560
DDPG 2+2 States	0.0955	50.59	0.8573
Adaptive NLGL	0.1184	49.48	0.8837

From the obtained results, it is shown that 2+1 and 2+2 states agents perform very similar, being the 2+1 version even better. Therefore, it could be considered that the best state vector information to solve the path following problem is formed by only 1 future angle error state. Nevertheless, this is only possible if this state is placed in the proper distance given the vehicle's velocity. That is, the distance of 60cm, where the future error angle was placed, is the best distance for the particular velocity of 1 $^m/_s$ in this model. But if vehicle's velocity is changed, this distance needs to be recomputed and the agent needs to be trained from scratch. Thus, having more future angle error points may overcome this problem resulting in a more adaptable agent to different velocity references of the vehicle.

VI. CONCLUSIONS

In this paper *Deep Deterministic Policy Grandient*, a reinforcement learning algorithm that is able to work in continuous state-action environments, has been proposed to solve the path following problem in a quadrotor vehicle. This approach implements a separated guidance and control structure, and the *DDPG* agent is in charge of computing the yaw angle command.

The proposed agent has been implemented in python and has been trained and tested in the RotorS-Gazebo environment, which implements a realistic model of the Asctec Hummingbird vehicle. Several agents with different state configurations have been trained and the most prominent are presented in this paper. From the obtained results, it is shown that the agents that have states of future angle errors are able to outperform the standard 2 states *DDPG* agent and also improve significantly the performance of the *Adaptive NLGL* algorithm.

In conclusion, the proposed approach reduces substantially the training complexity of the *Adaptive NLGL* algorithm while achieves a better performance. Nevertheless, the aim of this paper is not only to show that the path following problem can be solved by means of DRL theory, but to present an initial framework that can be upgraded to solve more challenging problems such as wind disturbance rejection, adaptive velocity selection, adaptability to different models, obstacle avoidance, etc. Our immediate objective is to improve the approach to make it able to compute the optimal velocity of the vehicle depending on the path's shape and to implement the agent in the real quadrotor platform.

ACKNOWLEDGMENTS

This work has been partially funded by the Spanish State Research Agency (AEI) and the European Regional Development Fund (ERDF) through the SCAV project (ref. MINECO DPI2017-88403-R), and by SMART project (ref. EFA 153/16 Interreg Cooperation Program POCTEFA 2014-2020). Bartomeu Rubí is also supported by the Secretaria d'Universitats i Recerca de la Generalitat de Catalunya, the European Social Fund (ESF) and AGAUR under a FI grant (ref. 2017FI_B_00212).

REFERENCES

[1] B. Rubí, B. Morcego, and R. Pérez, "Adaptive nonlinear guidance law using neural networks applied to a quadrotor," in *2019 15th IEEE International Conference on Control and Automation*, 2019.

[2] A. P. Aguiar, J. P. Hespanha, and P. V. Kokotovic, "Performance limitations in reference tracking and path following for nonlinear systems," *Automatica*, vol. 44, no. 3, pp. 598–610, 2008.

[3] I. Kaminer, O. Yakimenko, A. Pascoal, and R. Ghabcheloo, "Path generation, path following and coordinated control for time-critical missions of multiple UAVs," in *2006 AMERICAN CONTROL CONFERENCE*.

[4] S. Su, "Path following control of non-minimum phase VTOL aircraft via minimum distance projection method," in *26TH Chinese Control and Decision Conference (CDC)*, 2014, pp. 708–712.

[5] D. Cabecinhas, R. Cunha, and C. Silvestre, "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.

[6] A. Akhtar, S. L. Waslander, and C. Nielsen, "Fault Tolerant Path Following for a Quadrotor," in *2013 IEEE 52ND ANNUAL CONFERENCE ON DECISION AND CONTROL (CDC)*.

[7] T. Faulwasser and R. Findeisen, "Nonlinear model predictive control for constrained output path following," *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, vol. 61, no. 4, pp. 1026–1039, 2016.

[8] P. B. Sujit, S. Saripalli, and J. B. Sousa, "Unmanned aerial vehicle path following: A survey and analysis of algorithms for fixed-wing unmanned aerial vehicless," *IEEE Control Systems*, vol. 34, no. 1, pp. 42–59, 2014.

[9] A. Gautam, P. B. Sujit, and S. Saripalli, "Application of guidance laws to quadrotor landing," in *International Conference on Unmanned Aircfraft Systems (ICUAS)*, 2015, pp. 372–379.

[10] A. Manjunath, P. Mehrok, R. Sharma, and A. Ratnoo, "Application of virtual target based guidance laws to path following of a quadrotor uav," in *2016 INTERNATIONAL CONFERENCE ON UNMANNED AIRCRAFT SYSTEMS (ICUAS)*, 2016, pp. 252–260.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[12] J. C. Caicedo and S. Lazebnik, "Active object localization with deep reinforcement learning," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015, pp. 2488–2496.

[13] L. Li, Y. Lv, and F. Wang, "Traffic signal timing via deep reinforcement learning," *IEEE/CAA Journal of Automatica Sinica*, vol. 3, no. 3, pp. 247–254, July 2016.

[14] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016.

[15] R. Yu, Z. Shi, C. Huang, T. Li, and Q. Ma, "Deep reinforcement learning based optimal trajectory tracking control of autonomous underwater vehicle," in *2017 36th Chinese Control Conference (CCC)*, July 2017, pp. 4958–4965.

[16] L. P. Tuyen and T. Chung, "Controlling bicycle using deep deterministic policy gradient algorithm," in *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, June 2017, pp. 413–417.

[17] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *2016 International Conference on Learning Representations (ICLR)*, 2016.

[18] A. Rodriguez-Ramos, C. Sampedro, H. Bavle, P. de la Puente, and P. Campoy, "A deep reinforcement learning strategy for uav autonomous landing on a moving platform," *Journal of Intelligent & Robotic Systems*, vol. 93, no. 1, pp. 351–366, Feb 2019.

[19] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*.

[20] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *RotorS—A Modular Gazebo MAV Simulator Framework*. Cham: Springer International Publishing, 2016, pp. 595–625.

[21] B. Rubí, A. Ruiz, R. Pérez, and B. Morcego, "Path-flyer: A benchmark of quadrotor path following algorithms," in *2019 15th IEEE International Conference on Control and Automation*, 2019.

[22] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *ArXiv*, vol. abs/1502.03167, 2015.