Autonomous Drone Takeoff and Navigation Using Reinforcement Learning

Sana Ikli[1] and Ilhem Quenel[2]
[1]*Hybrid Intelligence, Capgemini Engineering, 4 Avenue Didier Daurat, 31700 Blagnac, France*
[2]*Applied AI program pilote, Capgemini Engineering, 4 Avenue Didier Daurat, 31700 Blagnac, France*

Keywords: Drone Navigation, Reinforcement Learning, Machine Learning, Drone Simulators.

Abstract: Unmanned Aerial Vehicles, also known as drones, are deployed in various applications such as security and surveillance. They also have the key benefit of being able to operate without a human pilot, which make them suitable to access difficult areas. During autonomous flights, drones can crash or collide with an obstacle. To prevent such situation, they need an obstacle-avoidance solution. In this work, we are interested in the navigation with obstacle avoidance of a single drone. The goal is to autonomously navigate from an origin to a destination point, including takeoff, without crashing. Reinforcement learning is a valid solution to this problem. Indeed, these approaches, coupled with deep learning, are used to tackle complex problems in robotics. However, the works in the literature using reinforcement learning for drone navigation usually simplify the problem into 2-D navigation. We propose to extend these approaches to complete 3-D navigation by using a state-of-the-art algorithm: proximal policy optimization. To create realistic drone environments, we will use a 3-D simulator called *Pybullet*. Results show that the drone successfully takes off and navigates to the indicated point. We provide in this paper a link to our video demonstration of the drone performing navigation tasks.

1 INTRODUCTION

In our previous work (Ikli, 2022), we consider the scheduling of flying taxi operations, which is similar to the scheduling of drone missions. The goal was to transport a parcel from an origin to a destination point, while satisfying some operational constraints. However, we simplified the problem by considering that the drone (or the flying taxi), after the vertical takeoff, follows a straight line to the destination point, and then it lands vertically. Moreover, we didn't consider any obstacles between the origin and the destination points. In our current work, we are interested in the navigation problem of the drone in a realistic environment. More specifically, we want the drone to autonomously navigate between the origin and the destination points, while avoiding obstacles. To simplify the autonomous navigation, we consider the takeoff and the navigation to the destination point as two separate tasks. Indeed, we first focus on the drone's takeoff to reach a predefined target takeoff point. Then, we consider the navigation from this target point to our destination point. Finally, we combine the two models (takeoff and simple navigation) to obtain a complete navigation model. The particularity of our work is that we consider 3-D navigation, unlike the majority of works in the literature that perform lateral navigation only.

Reinforcement Learning (RL)[1]) techniques are used to tackle a wide variety of control problems, such as aircraft autonomous sequencing and scheduling (Brittain and Wei, 2021). For this reason, we intend to use these algorithms to solve the drone takeoff and navigation problems. In the next section, we present a brief overview of RL concepts and algorithms. We also provide a synthesis of the most relevant articles in the literature that use RL for drone navigation problems.

2 STATE-OF-THE-ART

Reinforcement learning is a Machine Learning (ML) paradigm concerned with how an agent can optimally behave in an unknown environment. Unlike supervised learning, where the agent learns from labeled

[1]Acronyms and their meaning are outlined in Table 3.

Ikli, S. and Quenel, I.
Autonomous Drone Takeoff and Navigation Using Reinforcement Learning.
DOI: 10.5220/0012296300003636

In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 3*, pages 63-71
ISBN: 978-989-758-680-4; ISSN: 2184-433X

input and output data, RL agents learn by interacting with the environment through trials and errors. We present in this section a brief introduction to reinforcement learning concepts, and a "tour d'horizon" of articles using these algorithms for drone navigation.

2.1 Overview of Reinforcement Learning

The basic framing of a reinforcement learning problem is the *Markov Decision Process* (MDP), defined by a quadruplet (S, A, T, r) where:

- S is a set of states that represent the system at a given time. It can be discrete or continuous.
- A is a set of actions, *i.e.,* the possible interactions with the environment.
- T is a transition function. Given a state and an action, T defines the transition probability to the possible resulting states.
- r is the immediate reward after transitioning to the resulting state.

A basic RL agent interacts with its environment in discrete time steps. At each time t, the agent receives the current state s_t and reward r_t. It then chooses an action a_t which is executed in the environment. The agent then moves to a new state s_{t+1} and the reward r_{t+1} associated with the transition (s_t, a_t, s_{t+1}) is determined (See Figure 1). The goal of a RL agent is to learn the optimal policy: $\pi : A \times S \rightarrow [0,1], \pi(a,s) = Pr(a_t = a | s_t = s)$ that maximizes the sum of discounted future rewards.

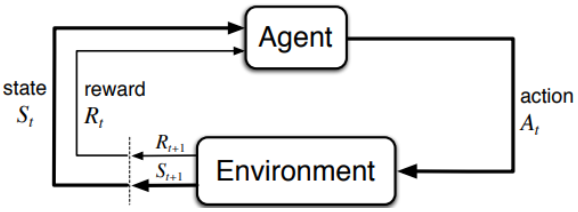

Figure 1: Agent-environment interaction in an MDP. *Source: (Sutton and Barto, 2018).*

Various methods are available in RL to approximate this optimal policy. The main approaches are value-function-based algorithms, such the Q-Learning (Watkins and Dayan, 1992), where the sum of rewards (known as *return*) is repeatedly estimated and the policy π is improved. Another category of RL algorithms are the *model-free* algorithms, that simply learn from the samples, gathered from interactions with the environment. Finally, we can cite the policy-gradient algorithms, that we will use in this work. As opposed to value-function-based algorithms, policy-based algorithms directly estimate the parameters of the optimal policy, usually modeled as an artificial neural network. Readers may refer to the book of (Sutton and Barto, 2018) for an in-depth description of all the reinforcement learning algorithms categories.

For the drone takeoff and navigation tasks, no representation of the environment is provided; the only information the drone has about its environment are obstacles locations and the destination point. Hence, model-free RL algorithms are the most adapted to this context. We chose a policy-gradient-based algorithm called the Proximal Policy Optimization (PPO). It was proposed by (Schulman et al., 2017), and is one of the most successful deep reinforcement learning algorithms in solving various tasks (Wang et al., 2020). It benefits from trust-region policy optimization methods, and it (empirically) has better sample efficiency (Schulman et al., 2017).

We choose this algorithm for mainly two reasons: (i) It is suited to MDPs with continuous state and action spaces, which is our case (see Section 3). (ii) It has good convergence properties, and showed good results in other control problems in robotics (Schulman et al., 2017).

2.2 Drone Navigation in the Literature

Despite its success in solving complex control problems, reinforcement learning is rarely used in the literature to tackle the autonomous drone navigation problem. The majority of the works use standard optimization or path planning methods (Yasin et al., 2020). In this section, we overview some of the most relevant RL techniques for drone navigation problems. We summarize in Table 1 the approaches discussed in this section. For an in-depth and comprehensive review, readers may refer to the recent survey of (AlMahamid and Grolinger, 2022).

In order to use reinforcement learning, the problem of drone navigation must be expressed as an MDP. In (Wang et al., 2017), the following MDP model was proposed:

- States contain the distance between the drone and the obstacles and the angle between the current position and the destination.
- Actions consist only of a change in the drone's heading.
- The reward is composed of four components: an environment penalty, a transition recompense, a constant step penalty, and a direction reward.

Table 1: A summary of RL techniques for drone navigation.

Source	Description	Sensor	Solution method	Simulator	Remarks
(Wang et al., 2017)	Drone navigation between two points	GPS	RDPG, Fast-RDPG	Simulated environment	Need to test on a simulator or more realistic environments
(Sampedro et al., 2018)	Drone navigation in unknown environment	Laser	DDPG	Gazebo with ROS	Consideration of dynamic obstacles
(Dankwa and Zheng, 2019)	model and train a 4-ant-legged robot	Camera	DDPG	Pybullet	Long training phase
(Beeching et al., 2021)	3-D control of a robot	Camera	A2C	ViZDoom	Separate test scenario configuration
(Morad et al., 2021)	Drone navigation using RGB Camera	Camera	Deep RL	Simulated environment	Test on real-world drones

To solve the model, the authors use a deep RL algorithm called RDPG (Recurrent Deterministic Policy Gradient). They also improved this algorithm to make it faster.

Another deep RL framework is proposed in (Sampedro et al., 2018) to solve the autonomous navigation of a drone in an unknown environment, using a laser as a sensor. The proposed method is a mapless reactive approach, based on the Deep Deterministic Policy Gradient (DDPG). Simulation experiments are conducted in the Gazebo environment (Furrer et al., 2016) with ROS (Robot Operating System) (Quigley et al., 2009). The training is performed on a $8m \times 8m$ square. Results show the reactive navigation capability of the drone, especially in the presence of dynamic obstacles.

The authors of (Dankwa and Zheng, 2019) propose a Twin-Delayed DDPG (TD3) algorithm to model and train a 4-ant-legged virtual robot. The TD3 algorithm consists of three RL methods: Continuous Double Deep Q-learning, policy gradient, and actor-critic. The virtual robot is modeled in the environment called "AntBulletEnv-v0", from Pybullet[2] package in Python. Thus, the components of the MDP (states, actions, and reward) are defined in the environment. The robot was trained by performing 500000 iterations of TD3 algorithm, and succeeded in obtaining comparable results with state-of-the-art RL algorithms.

A number of works have addressed the drone navigation problem, but the majority oversimplify it, by only considering *turning right* or *left* as possible actions for the drone. In our work, we propose to extend these works to complete 3-D navigation. We present in the next section our MDP formulation for the two drone tasks: takeoff and simple navigation.

[2]Pybullet: https://pybullet.org/wordpress/

3 PROBLEM MODELING

To autonomously navigate while avoiding obstacles, the drone needs to extract information from its environment using sensors (*e.g.* RGB camera). Then, it generates actions such as "Go to position (x,y,z)" or "Decrease the speed" to avoid the obstacles. This problem is formulated as a Markov decision process, composed of states, actions and rewards.

- A state in our MDP model represents a vector (x,y,z) that defines the position of the drone in the 3-D environment. In our work, this position is directly obtained from the API (Application Programming Interface) of the simulator.
- Actions consist in changing the position of the drone, in the 3D environment according to the X,Y, and Z axes. The state and action spaces are thus continuous.
- Rewards: The ultimate goal is to make the drones navigate to a destination point. This goal is decomposed into two tasks: (i) Takeoff to join a target point, (ii) navigate from this point to the destination. This second task is referred to as "simple navigation" task. In the next subsection, we explain the different rewards we designed for each task.

3.1 Reward Design

Reward design was very challenging. We first tried classical and simple rewards that recompense the drone with a positive value if it correctly performs the task (takeoff or simple navigation), and penalize it with a negative value if it crashes or hits an obstacle. This kind of reward causes the drone to crash after a few steps. We then designed some rewards based

on the simple Euclidean distance between the drone's position and the target point (for takeoff or destination point for navigation). An example of these rewards is: $-\|(x^{target}, y^{target}, z^{target}) - (x, y, z)\|_2^2$. This formulation simply encourages the drone to go to the target point $(x^{target}, y^{target}, z^{target})$. Using these rewards causes the drone to either crash on the ground or fly away from all obstacles.

Fortunately, we designed a new reward that is positive, bounded in $[0, 1]$, and differentiable. This formulation rewards the drone when it gets closer to a target point $(x^{target}, y^{target}, z^{target})$, using the Gaussian function (1).

$$r_{\text{gauss}} = e^{-\|(x^{target}, y^{target}, z^{target}) - (x,y,z)\|_2^2} \quad (1)$$

For the takeoff task, the drone is located in the initial position $(0, 0, 0)$. For the simple navigation, the drone is located in the target take-off point. The target point in the equation (1) is:

- $(x^{target}, y^{target}, z^{target}) = (0, 0, 1)$ for the takeoff task,
- $(x^{target}, y^{target}, z^{target}) = (1, 1, 2)$ for the simple navigation.

We add several terms to the reward (1) to further improve the drone's behavior and encourage smooth movements. These terms will be detailed in Section 4.2.

4 SIMULATION RESULTS

This section presents the computational results of implementing the above-mentioned MDP model. The implementation of the PPO algorithm is adapted from the Stable-Baselines (Hill et al., 2018) library with the default parameters. Simulation results are reported in terms of several performance indicators, such as the drone's position, its velocity, and the reward it obtains. For a more visual result, we record a video demonstration of the drone after the training is completed. The first video is entitled `drone-take-off-and-hover-demo-pybullet` and shows the drone's takeoff and hovering near the target point $(0, 0, 1)$. It is available at: `https://www.youtube.com/shorts/h2TYtB2MYtA`. The second video `drone-combined-takeoff-navigation-pybullet` features the drone performing the complete navigation task. It is available at: `https://www.youtube.com/watch?v=ev5XTAAt73s&ab_channel=Demo-Drone`.

4.1 Simulation Setup

We use a simple yet popular drone simulator based on *Pybullet* (Erwin and Yunfei, 2019), called ***gym-pybullet-drones***. It is an open-source simulator, and can be directly installed from this GitHub link: `https://github.com/utiasDSL/gym-pybullet-drones`. To simulate the drone and its environment, we adapt the *TakeoffAviary* class (provided in the simulator) to our MDP model defined in Section 3. Moreover, we include obstacles with different shapes and configurations in the environment. Figure 3 shows a screenshot of the graphical interface of our simulated environment. We can see in this figure the drone is in the initial position $(0, 0, 0)$. We can also notice that all the obstacles have the same texture and are all colored blue. This is one inconvenient

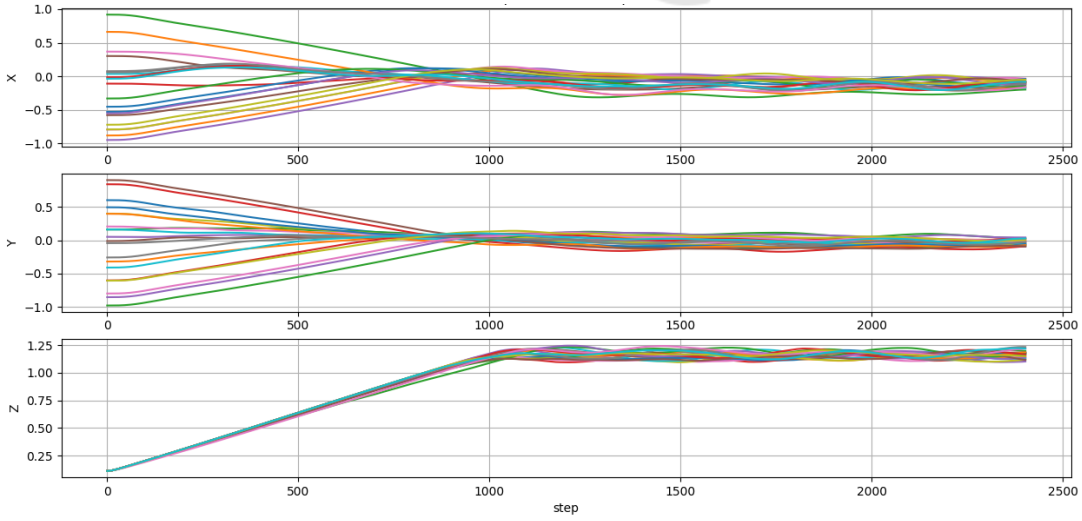

Figure 2: Drone positions in the X-Y-Z axes for 20 episodes with the trained PPO algorithm.

feature of this simulator, because it does not allow the customization of objects included in Pybullet environments.

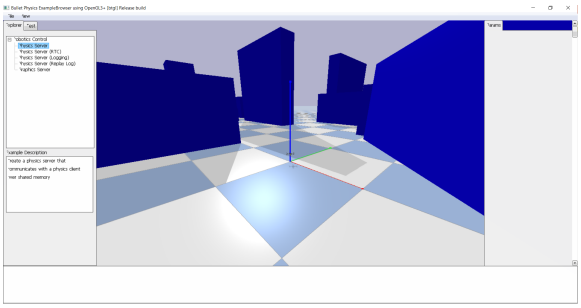

Figure 3: Simulated environment in gym-pybullet-drones.

In this work, we use Python 3.9 as a programming language. The experiments are run on a personal computer under the Windows operating system, with a processor of the 11th Gen of Intel(R) Core(TM) i5-11500H with 16 GB of RAM.

4.2 Takeoff Results

We first analyze the results for the takeoff task, in which the drone should navigate and hover above the target takeoff point $(0,0,1)$. Then, we present the results for the simple navigation task, from the target takeoff point to the destination point $(1,1,2)$. Finally, we combine the two model to obtain the complete navigation, from the initial point $(0,0,0)$ to the destination position $(1,1,2)$.

4.2.1 Training and Testing Results

We train the PPO algorithm for one million time steps, which required around one hour to complete the training. Training metrics are monitored live via Tensorboard (Abadi et al., 2015).

We present in Table 2 some training parameters. We change the initial position in the X-Y axes to be randomly generated in $[-1,1]$, to further encourage exploration of the environment. However, we maintain the remaining parameters of each algorithm as the default ones provided in the Stable-Baselines library.

After training is completed, we test the trained PPO agent for 20 episodes. Figure 2 shows the results in terms of drone positions in the X-Y-Z axes. As depicted in this figure, the drone succeeds in taking-off and hovering above the target point $(0,0,1)$. Indeed, after 1000 steps, the x and y positions are stabilized around 0, while the z coordinate is stabilized around 1. We can conclude that with the reward formulation (1), the PPO agent succeeds in accomplishing the takeoff task and hovering above the target point.

Table 2: Training parameters.

Parameters	Values
Origin point	$(0,0,0)$
Initial drone point	$-1 \leq (x,y) \leq 1$
Target take-off point	$(0,0,1)$
Destination point	$(1,1,2)$
Total training steps	1 Million
Learning rate	0.0003
Maximum episode length	5 seconds
Other training parameters	Default in the Stable-Baselines3

We further analyze the drone behavior right after takeoff, by looking at different indicators: The drone velocity in the three axes X-Y-Z and the drone rotation (pitch and yaw). We present in Figure 4 the drone velocity in the X-axis, and the pitch and yaw angles of the drone during testing. As depicted in this figure, the drone's velocity on the X-axis is very unstable. Furthermore, the drone's pitch and yaw angles oscillate around 0, but these oscillations tend to diverge, which may lead to the loss of its control.

In order to improve the drone's behavior and encourage smooth movements, we propose to include the following sub-rewards to the formulation (1).

- *Altitude term*: that encourages the drone to hover at the given altitude:

$$r_{\text{altitude}} = e^{-(1-z)^2} \tag{2}$$

- *Smoothness term*: that rewards smooth movement. It can, for instance be based on the velocity V of the drone:

$$r_{\text{smooth}} = e^{-\|V\|_2^2} \tag{3}$$

- *Collision term*: that penalizes the drone whenever it comes close to the obstacles.

$$r_{\text{collision}} = \begin{cases} 1 & \text{if } d_{\text{obstacle}} \geq 0.2, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

  Where d_{obstacle} denotes the distance between the drone and the closest obstacle3.

- *Loss of control term*: that helps to avoid dangerous or unstable states. It is based on the roll θ_r and pitch θ_p of the drone:

$$r_{\text{control}} = \begin{cases} 1 & \text{if } |\theta_r| \leq 30^\circ \text{ or } |\theta_p| \leq 30^\circ, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

The final reward is the weighted sum of the main reward (1), and the above-mentioned sub-rewards. Figure 5 shows the new results regarding the velocity along the X-axis, the pitch and the yaw angles, using this new weighted-sum reward. It can be seen in this figure that the drone's velocity is more stable. Moreover, its pitch and yaw angles oscillate

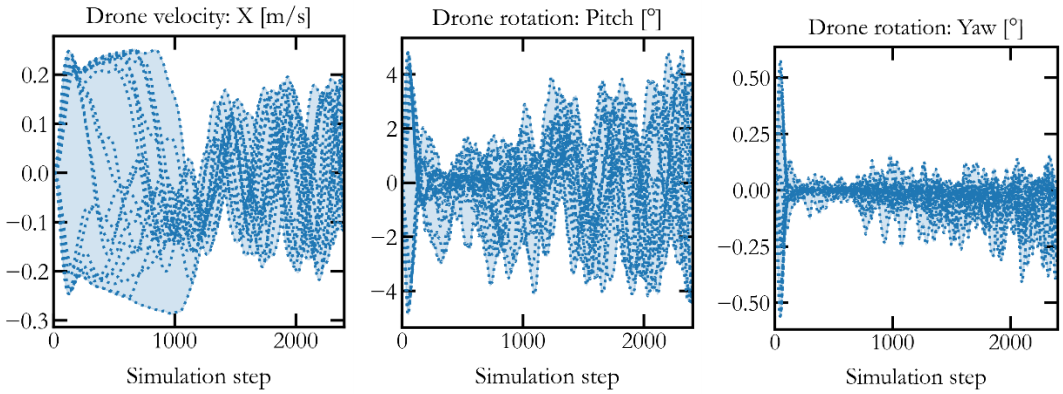

Figure 4: Drone's velocity and rotations.

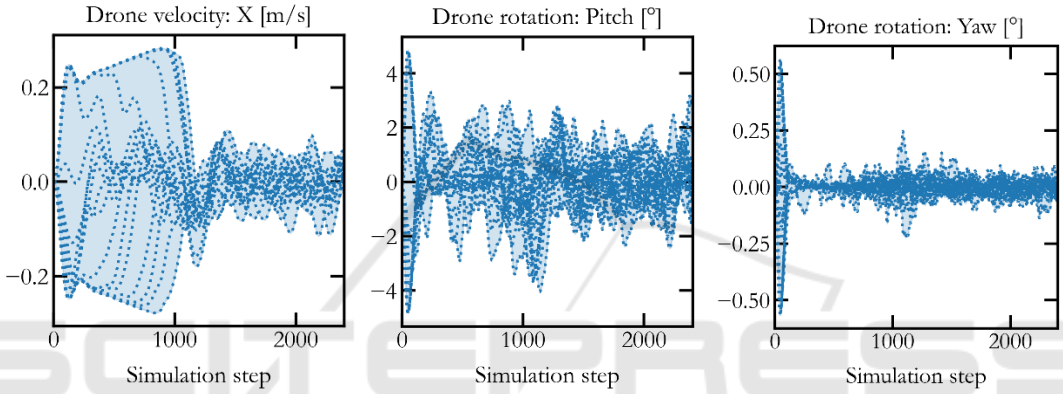

Figure 5: Drone velocity and rotations with the improved reward.

less, resulting in smooth drone movements. We provide a video demonstration of the drone performing the takeoff task, with smooth movements, available from the link: `https://www.youtube.com/shorts/h2TYtB2MYtA`.

4.2.2 Parallelization

In order to decrease the training time, we can parallelize the training. Indeed, instead of training the drone in a single environment during 1 million iterations, we can lunch 10 trainings for 100 000 iterations in parallel, thanks to a functionality in the Stable-Baselines3 library. We show in Figure 6 the computation times using up to 35 training environments. We can observe in this figure that by using ≈ 10 parallel environments, the training time is divided by 2 for the PPO algorithm. Using more parallel environments does not necessarily yield significant gains. Hence, for the next simulations, we define 10 parallel environments for the training.

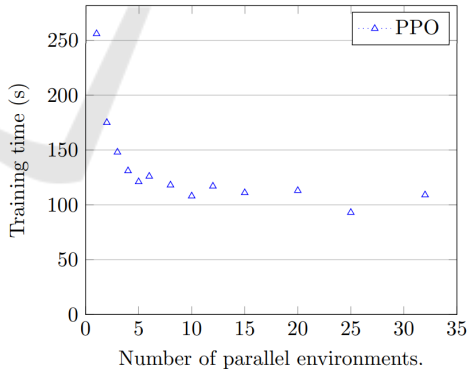

Figure 6: Training time for 100000 iterations.

4.3 Combined Takeoff and Navigation

In this part, we use the same simulation setup as in the takeoff task, and we define 10 parallel environments for the training. The only changes with respect to the takeoff task are:

- The total training steps is now 2 millions steps, which took around 3 hours.
- The initial point is the target take-off point

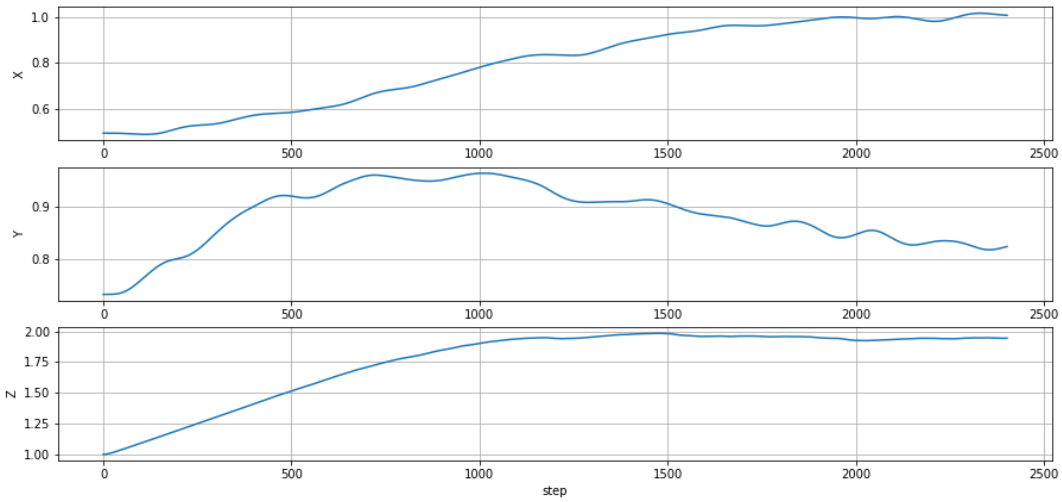

Figure 7: Drone positions in the X-Y-Z axis.

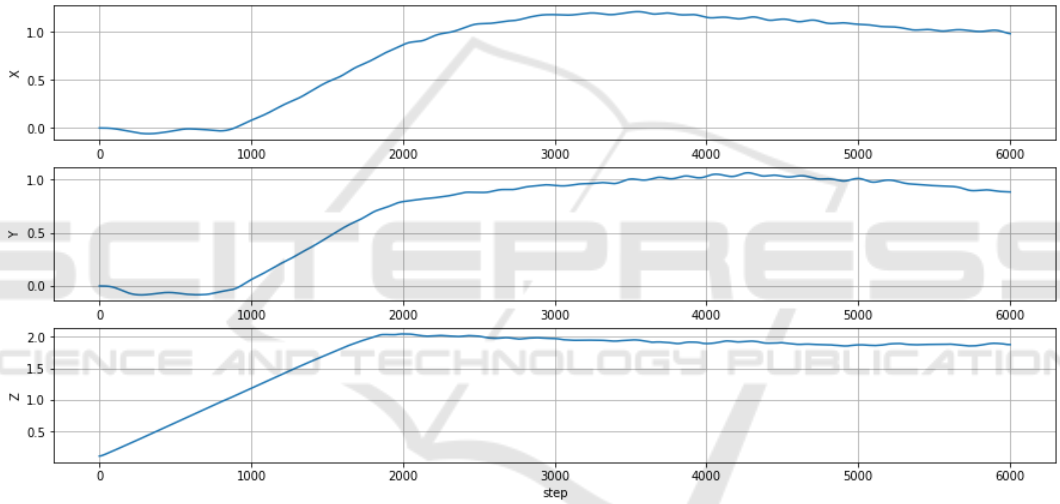

Figure 8: Drone positions in the X-Y-Z axis during one episode, with the combined model.

$(0,0,1)$.

- The destination point is $(1,1,2)$
- The episode length is set to 15 seconds, in order to have enough time for the drone to perform the navigation task. Increasing the episode duration further may cause longer training times.

For the simple navigation, the goal is to make the drone navigate from the target takeoff point to the destination point. The reward we use for this task is the weighted sum reward, in which the target point is now the destination point $(1,1,2)$. We show in Figure 7 the 3D positions of the drone. As depicted in this figure, the drone succeeds in navigating to the destination position $(1,1,2)$, after around 1000 steps.

Finally, we combine the trained takeoff model with the simple navigation to perform the complete navigation task: takeoff from the ground and navigation to the destination point $(1,1,2)$. **Figure 8** features the 3D position of the drone using the combined model. We can observe in the figure that after 2000 steps, the x and y coordinates of the drone oscillate around 1, and the z coordinates oscillate around 2. We also noticed in all the simulations that whenever the drone reaches a target point, it decides to hover above it. A possible explanation is that the fact that the reward is maximized at the target point. Thus, the drone navigates near this point to not decrease its reward.

For this combined model, we also provide another video demonstration of the drone performing the complete navigation task. The video is available at this link: `https://www.youtube.com/watch?v=ev5XTAAt73s&ab_channel=Demo-Drone`.

5 CONCLUSION AND PERSPECTIVES

In this work, we consider the autonomous navigation of a single drone using a state-of-the-art reinforcement learning algorithm called proximal policy optimization. We split the navigation goal into two separate tasks: takeoff to join a target takeoff point and a simple navigation task. We then combine the two tasks to perform complete and autonomous navigation from the ground to a destination point. To model these problems, we propose an adapted Markov Decision Process with a new reward, that enables the drone to accomplish each task. Moreover, we improve the reward formulation to encourage the drone to perform smoother and more stable movements.

The numerical simulations are conducted in the *Pybullet* simulator for drones, and using a well-known reinforcement learning library called Stable-Baselines3. Results show that the successful reward that enables the drone to accomplish the takeoff and the navigation tasks is non-negative, differentiable and bounded. We also learn from this numerical study that the training time can be significantly decreased when training the drone in parallel in different environments. We conclude from this study that reinforcement learning approaches are promising techniques for drone navigation. However, they have two main drawbacks: (i) the challenging problem of designing relevant rewards that include all the objectives in one formulation, (ii) the need for millions of interactions with the simulator in order to learn meaningful policies. The latter is called *sample inefficiency*, and it is a well-known problem in reinforcement learning.

Future tracks of research include the extension of this work to consider the landing on a specific point. Designing a reward formulation that can integrate all these objectives (takeoff, navigation, landing) together with obstacle avoidance can also be a perspective of this work.

ACKNOWLEDGEMENTS

The authors would like to thank Jimmy Debladis, Besma Khalfoun and Fadi Namour From Capgemini Engineering, for the technical discussions that greatly improved the quality of this study.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. `https://www.tensorflow.org/tensorboard`.

AlMahamid, F. and Grolinger, K. (2022). Autonomous unmanned aerial vehicle navigation using reinforcement learning: A systematic review. *Engineering Applications of Artificial Intelligence*, 115:24 pages.

Beeching, E., Debangoye, J., Simonin, O., and Wolf, C. (2021). Deep reinforcement learning on a budget: 3D control and reasoning without a supercomputer. In *25th International Conference on Pattern Recognition (ICPR)*.

Brittain, M. and Wei, P. (2021). Autonomous aircraft sequencing and separation with hierarchical deep reinforcement learning. In *Learning-based decision making for safe and scalable autonomous separation assurance*.

Dankwa, S. and Zheng, W. (2019). Twin-delayed DDPG: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent. In *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing*.

Erwin, C. and Yunfei, B. (2019). Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`.

Furrer, F., Burri, M., Achtelik, M., and Siegwart, R. (2016). Rotors—a modular gazebo MAV simulator framework. In *Robot operating system (ROS)*.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. (2018). Stable baselines. `https://github.com/hill-a/stable-baselines`.

Ikli, S. (2022). A rolling horizon approach for the dynamic scheduling of flying taxis. In *Proceedings of the IJCCI 2022 Conference*. SCITEPRESS.

Morad, S. D., Mecca, R., Poudel, R. P., Liwicki, S., and Cipolla, R. (2021). Embodied visual navigation with automatic curriculum learning in real environments. *IEEE Robotics and Automation Letters*, 2:683–690.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., et al. (2009). ROS: An open-source robot operating system. In *ICRA workshop on open source software*.

Sampedro, C., Bavle, H., Rodriguez-Ramos, A., de La Puente, P., and Campoy, P. (2018). Laser-based reactive navigation for multirotor aerial robots using deep reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. In *arXiv preprint arXiv:1707.06347g*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press, 2nd edition.

Wang, C., Wang, J., Zhang, X., and Zhang, X. (2017). Autonomous navigation of UAV in large-scale unknown complex environment with deep reinforcement learning. In *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*.

Wang, Y., He, H., and Tan, X. (2020). Truly proximal policy optimization. In *Uncertainty in Artificial Intelligence*.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8:279–292.

Yasin, J. N., Mohamed, S. A., Haghbayan, M.-H., Heikkonen, J., Tenhunen, H., and Plosila, J. (2020). Unmanned aerial vehicles (UAVs): Collision avoidance systems and approaches. *IEEE access*, 8:105139–105155.

ACRONYMS

The following table defines the acronyms we used in this article.

Table 3: Acronyms definition.

Acronym	Meaning
DDPG	Deep Deterministic Policy Gradient
MDP	Markov Decision Process
ML	Machine Learning
RDPG	Recurrent Deterministic Policy Gradient
RL	Reinforcement Learning
UAV	Unmanned Arial Vehicles