Deep Reinforcement Learning based Robust Quadcopter Stabilization System
Advanced Deep Learning Models and Methods Research Project, 2022

Lorenzo Poretti
Politecnico di Milano
lorenzo.poretti@mail.polimi.it

Alessandro Restifo
Politecnico di Milano
alessandro.restifo@mail.polimi.it

Abstract

Reinforcement learning has been steadily growing in popularity for few years. Notable applications are in the fields of finance, healthcare, natural language processing and, especially, robotics. In particular, deep reinforcement learning excels in the automatic development of control systems, with the advantage — compared to classical techniques — of being able to model non-linear dynamics and kinematics.
The goal of this work is to explore and test the applications of deep reinforcement learning in the autonomous control systems field. Specifically, the project aim is to develop a deep-RL based controller for quadcopters. Different RL agents are developed and evaluated leveraging both on-policy and off-policy learning, within a simulated environment. Our work focuses on off-policy learning, mainly due to the much better sample-efficiency of the process. Multiple actor-critic network structures are evaluated. The agents are trained to stabilize the quadcopter given its state in the world, by moving and keeping it in the defined target position.

1. Introduction

Drone stabilization and control is a vital application, especially in commercial drones, as it allows anyone to pilot a quadcopter with an extremely easy abstraction of the underlying kinematics, while focusing only on the movements of the drone and not on compensating its inherent instability.
A brief introduction to quadcopters and previous efforts in developing control systems are here presented.

1.1. Quadcopters

Quadcopters are a category of drones which in the last years has seen an exponential gain in interest all over the world [Fig. 1].

These drones are flying machines equipped with four different propellers, spinning in opposite directions, so that the overall moment and propellers torque in the system is null and the drone itself inherently does not spin when hovering in the air [Fig. 2].

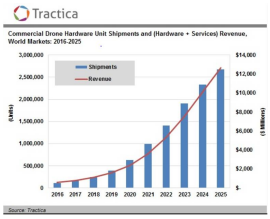

Figure 1. Exponential increase in drone sales in the last decade.

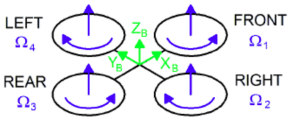

Figure 2. Propellers rotate in opposite directions to decrease the angular momentum and the overall propellers torque.

At the lowest level, the drone movement is controlled via the thrust of each propeller. In the same way, the thrust of each propeller is governed via PWM (pulse-width modulation). PWM is a method of reducing the average power delivered to the propellers, by dividing it into multiple discrete parts. The average value of voltage (and current) fed to the load is controlled by turning the switch between supply and load on and off at a fast rate [1]. The more frequently the switch is on, the more power is delivered to system, eventually leading to faster rotational speeds of the propellers and therefore more thrust. Instead of being presented with an interface to the PWM duty cycle for each propeller, the user is presented either with speed controls, or with throttle, roll, pitch and yaw controls. This can be done through modeling the underlying kinematics of the quadcopter, as in [2].

1.2. Traditional PID controller

Once the kinematics of the drone are resolved, the drone must be properly controlled. Traditionally, this has been

done with a PID controller. The general idea of the controller is to compute the error $e(t)$ between the process set value $r(t)$ and the measured process value $y(t)$ output from the system [Fig. 3]. The contributions from the error derivation and integration are then summed to the proportional contribution and used to set the system input.

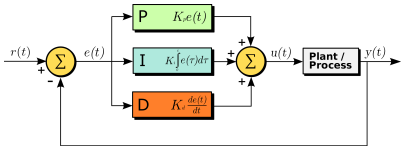

Figure 3. PID controller schema.

An interesting example of the application of a PID controller to a quadcopter can be found in [2], where the authors apply the concept to control the propellers thrust via the set tuple $< throttle, roll, pitch, yaw >$.

One of the disadvantages of PID controllers though, is that they are not very robust against external disturbances. If a system encounters challenging changes in the environment which lead to a big error $e(t)$, the PID controller may deliver poor performance.

1.3. Reinforcement learning controller

For the stated reasons, many efforts have been put in the development of deep reinforcement learning controllers. The advantage of modeling the best action to perform given the state through neural networks is clear: neural networks allow to map input and output variables through non-linear operations and, while they are parametric models, the number of parameters can be chosen case-by-case assessing the performance of the network itself. Moreover, no kinematic model needs to be specified, since the agent is able to directly model it through data.

In theory, a neural network controller should be able to learn the optimal behavioral policy in every challenging environment with a feasible control solution. The reinforcement learning framework typically defines the environment in the form of a Markov Decision Process (MDP). The basic components of this characterization are:

- a set states, S
- a set of actions, A, that can be performed by the agent
- the transitions probability

$$P_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a) \quad (1)$$

- the reward after transiting from one state to the next

$$R_a(s, s') \quad (2)$$

The reinforcement learning agent then tries to maximize the cumulative discounted reward by learning the optimal policy that maps states to actions. The deep reinforcement learning framework differs from the standard RL via the usage of neural networks. These can be used for different specific purposes, but the general idea is to use their interpolation capability to map the useful part of a very high-dimensional - or even continuous - input space to the action space; such task would require enormous storage and computational resources in classical RL.

Mainly, in deep RL the neural networks approximate the Q-value function (which expresses how good or bad an action is given a state) or, in actor-critic frameworks, directly approximate the action given a state. The latter mapping is called policy:

$$\pi(a, s) = Pr(at = a | st = s) \quad (3)$$

Reinforcement learning agents can learn either on-policy (directly improving the optimal policy learned that is used to make decisions) or off-policy (using a different policy for the exploration of the space, called behavioral policy).

An example of on-policy learning for quadcopter control system can be found in [3], which focuses predominantly on a simple PPO controller. PPO, as all the deep RL controllers used to control drones, is an actor-critic algorithm. It uses an actor network to estimate the action given the state (which represents the policy), and uses a critic network to map states and actions together to their discounted future rewards value, therefore evaluating the action "quality" given a state. Being the objective to stabilize the drone in place, the reward function used in the paper depends both on the distance from the set point, and from the squared linear and angular velocities.

A work by Lynch further extends the previous case. In [4] the authors still focus on the Proximal Policy Optimization algorithm, and extend the work - among the others - by proposing a number of different possible reward functions and evaluating their effect on the agents performance change during training. Their best reward function includes terms from the square of the angular and linear velocity, as well the squared positioning error. Moreover, it integrates the input torque generated by the propellers thrust.

In [5] the authors better explore the off-policy learning possibilities both with Deep Deterministic Policy Gradient and with other custom hybrid RL approaches. Both for the actor and the critic, they employ a symmetric structure with two networks made with two hidden layers of 64 units each, activated via tanh.

Finally, [6] presents a robust controller made of a RL actor and a compensator. Their actor network uses two hidden layers of 32 relu-activated units. They test their system with a wind disturbance of up to $3.6m/s$.

In every work, the input to the network is generally represented by the IMU output, hence the three linear and three angular accelerations, the respective velocities, and the pose

of the drone.

1.4. Contributions

The objective of this work is to build an effective and robust deep reinforcement learning controller for a quadcopter. The fundamental idea is to control the drone position via the offset from the target position, ensuring at the same time the robust stability of the drone itself. Although the bias and the differential control action supplied by the network have been decoupled for better control, best results require the drone to be in flight and hovering, as the ground effect generated by the propellers might cause changes in the thrust bias necessary to keep the drone steady. The policy network is highly flexible and can be applied to different drones with no changes, since the output is the delta in the thrust to apply to the thrust bias. The bias is drone-specific and should be calibrated as the propellers power necessary to keep the drone hovering steady in ideal conditions, squashed in $(0, 1)$. Our main contributions can be summarized as follows:

- New interpretation of the actor network output (scaled delta thrust), which allows for finer-grained control and makes the solution a drop-in component compatible with most drone once calibrated.
- New networks architecture, which strikes a balance between learning speed and representational capability.
- New controller, resistant to up to 5x stronger wind compared to previous works, and to 2x stronger wind compared to most commercial solutions.
- Completely off-policy agent training: the process is more sample-efficient than on-policy learning algorithms, and reaches good reward values with as low as 200000 steps. Longer training further increases the reward. No particular exploration policies are needed to converge.
- Robustness to extreme perturbations in initial pitch, roll and yaw angles. The controller can even recover from upside down states.
- Moreover, the neural network controller can sustain high deltas between the initial pose and the set pose (or can be used to control the drone at a lower level).

2. Approach

The approach proposed is based, as stated, on a deep reinforcement learning controller, trained with off-policy learning given the much higher sample efficiency of the process. The agent chosen is Twin Delayed Deep Deterministic Policy Gradient (TD3), as it represents one of the latest and most effective off-policy algorithms. In section 2.1 an overview of the simulation environment is given, section 2.2 exposes details on the thrust management, sections 2.3 and 2.4 detail the agent and the networks used.

2.1. Simulation environment

AirSim simulator [7] has been used in order to train and evaluate the Reinforcement Learning algorithms. It is an open-source simulator built on Unreal Engine and developed by Microsoft. It is mainly used in simulations of autonomous quadcopters and ground vehicles. AirSim not only constitutes a simulator, as it also exposes interfaces between the python code and the PhysX simulator itself. The great advantage of AirSim is that it allows, using an API, to receive data from the sensors, to process them using variety of languages and to provide commands to the actuators. From the API it is also possible to change the state of the environment such as setting a constant wind. The Reinforcement Learning algorithms can be deployed on a variety of simulated flight controllers such as PX4 and ArduPilot. AirSim adopts a Z-down reference system. The pose of the quadcopter is defined by its position and orientation. The position is used to identify the center of the quadcopter in the 3D space using $< x, y, z >$ coordinates, expressed in meters. The orientation is given by a quaternion $< x, y, z, w >$ from which roll pitch and yaw angles are derived. The initial configuration of the quadcopter is the following:

$$position :=< 0, 0, -100 > \tag{4}$$

$$orientation :=< 0, 0, 0, 1 > \tag{5}$$

The framework adopted for the reinforcement learning agents training is tensorflow agents, a new library developed by Google. Training and evaluation have been carried out using the following hardware: Intel Core i7-11800H, Nvidia RTX3070; Intel Core i7-4790K, Nvidia GTX970.

2.2. Thrust management

In the majority of papers, the network output directly control the four motors. In this paper we present an innovation to this approach. The network does not directly control the four motors thrust values (between 0.0 and 1.0). Instead, for each rotor i we supply $\phi[i]$ thrust defined as:

$$\phi[i] = clip(\beta + \gamma(\sigma[i] - 0.5), 0, 1) \quad \forall i \in [0, 3] \tag{6}$$

In (6), $\phi[i]$ is the clipped sum of the constant β, which is the thrust bias, and the expression $\gamma(\sigma - 0.5)$. The thrust bias is equal to the thrust value of the rotors which stabilizes the drone in ideal conditions. It is a constant and can be calculate given the specific quadcopter considering the weight of the drone, its power output, and the propellers characteristics. An important role in the equation is the parameter $\sigma[i]$, which represents the output of the Neural Network for the propeller i. γ is a scaling factor which allows to reduce

the overall freedom of the network output, limiting it to a narrower interval. This allows the network to have finer-grained control over the final thrust values, which eases the stabilization task. Being the quadcopter an inherently unstable system, slight modifications to the rotors thrust produce macroscopic effects in the behavior and stability of the drone. In the simulator we calibrate and set the parameters as $\beta = 0.59$ and $\gamma = 0.5$.

Since this type of RL controller is parametric, in addition to achieving better performance (compared to directly controlling all rotors) as its output is limited in thrust range, it is also more easily portable to different types of quadcopters with different weights and propellers.

2.3. Agents evaluation

As stated in the introduction, the objective of the work is to learn an optimal policy for the control of the quadcopter. Being both the state and action spaces continuous, as better explained in paragraph 2.4, two main techniques are available:

- Discretization of the action space and usage of DQN (deep Q-network) related methods.
- Usage of actor-critic methods.

The drawback of the discretization method is clear: in a critical application as the drone stabilization, the continuity of the action space for a smooth control is essential. Discretization would lead to an extremely high number of actions. Actor-critic methods are therefore chosen. Inside the actor-critic category, multiple algorithms exist.

PPO (Proximal Policy Optimization) is an on-policy learning algorithm, like TRPO (Trust Region Policy Optimization). TRPO works by defining a "trust region" for updating the policy. Inside such region the local approximations of the policy function (whose differentiation gives the policy gradients for the policy update) are accurate. The algorithm then iteratively improves the policy with new on-policy samples. While TRPO performs well, PPO is based on similar concepts but is both easier to implement and better working. PPO updates the policy and the value function via gradient ascent. The algorithm substitutes the Trust region of TRPO by imposing a limit on the ratio between the new policy and old policy, which is supposed to be in the range $[1 - \epsilon, 1 + \epsilon]$, and therefore clipping the objective function so that the updates via gradient computation are kept small enough.

In the off-policy category instead, one of the first algorithms that had been proposed was DDPG (Deep Deterministic Policy Gradients). DDPG can be considered deep Q-learning for continuous action spaces. It uses off-policy data to learn the Q-function (represented by the critic network), and uses the Q-function to learn the policy (actor network). DDPG interleaves updating the critic network $Q^*(s, a)$ with learning the actor network that approximates $a^*(s)$. To improve the learning process, it uses a few solutions as "replay buffers" and "target networks". The first allows to keep in memory batches of observations from which to randomly sample during the network updates. The second allows to stabilize the learning by updating the networks and only performing a weighted copy of their weights to the target networks after "target-update-period" updates. While DDPG is a good agent, improvements have been proposed. TD3 extends it with a few solutions:

- Learning two Q-functions approximators instead of one, using the smaller to form the targets in the Bellman error loss functions (Clipped Double-Q Learning).
- Less frequent updates of the policy (and target networks) compared to the Q-function ("Delayed" Policy Updates).
- Addition of noise to the target action, to prevent the exploitation of Q-function errors (Target Policy Smoothing).

Finally, Soft Actor Critic (SAC) is another off-policy algorithm, similar in principle to TD3, with few key differences: for instance, it adds entropy regularization, while lacking the target smoothing.

In our testing, TD3 performed the best overall, as [Fig. 4] suggests, with overall faster learning and better visual results:

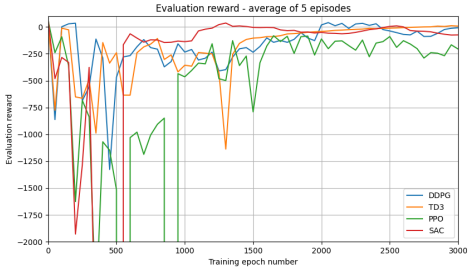

Figure 4. Confront of different agents performance on the drone stabilization problem. TD3 is more stable and after 3000 epochs (4 hours of computation) is outperforming the other algorithms.

Every test was executed with the same settings (section 2.4) and reward function, chosen after many different trials. The best results were obtained with the following reward:

$$R_a(s, s') = max(0, 1 - \|x'\|) - 0.1\,\|\theta'\| - 0.1\,\|w'\| \quad (7)$$

where x' represents the new position of the drone, θ' the new orientation, w' the new angular velocity, after having applied the action.

Such reward places special attention on the orientation and the angular velocity, which represent the main factors in the instability of the drone.

2.4. Neural networks structure

The structure of the actor and critic networks is different, as their function. While the actor has to estimate the best action to perform given a state, the critic has to estimate the Q-value of a state-action pair.

The state input to the two networks is the same, and is composed of 19 different floating-point (fp32) values. In order:

- 3 values for Δx, the difference between the current position and the set point.
- 4 values for $\Delta\theta$, between current orientation and set point orientation.
- 3 values for $< \dot{w_x}, \dot{w_y}, \dot{w_z} >$ angular accelerations.
- 3 values for $< w_x, w_y, w_z >$ angular velocities.
- 3 values for $< \dot{v_x}, \dot{v_y}, \dot{v_z} >$ linear accelerations.
- 3 values for $< v_x, v_y, v_z >$ linear velocities.

As for the actions, they are composed of 4 fp32 values which represent the thrust to apply to each propeller, as explained in section 2.2.

The actor network is composed, excluding the input and tanh-activated output layers, of 2 hidden layers of 128 units each. All the layers are activated via the hyperbolic tangent function [Fig. 5]. From internal tests, the tanh activation performs generally better than other activations, as sigmoid or even ReLU. This might be due to the symmetry of the function, which previous analysis have found to be useful for convergence [8], and the "pseudo-regularization" of the layers output, byproduct of the squashing of the output in the $(-1, 1)$ range.

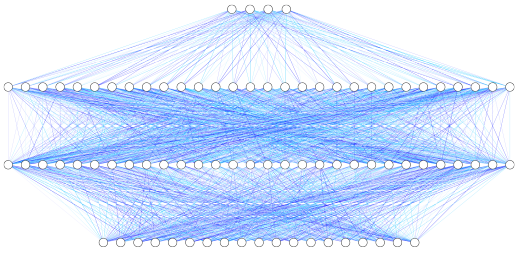

Figure 5. Actor network final architecture. Input Layer $\in R^{19}$, Hidden Layers $\in R^{128}$, Output Layer $\in R^{4}$.

On the other hand, the critic network is activated via ReLU functions, with the output layer being made of a simple Dense with linear activation. This setup was chosen because the critic does not have a bounded output.

In order to choose the best network architecture, multiple structures have been evaluated. Following are the results.

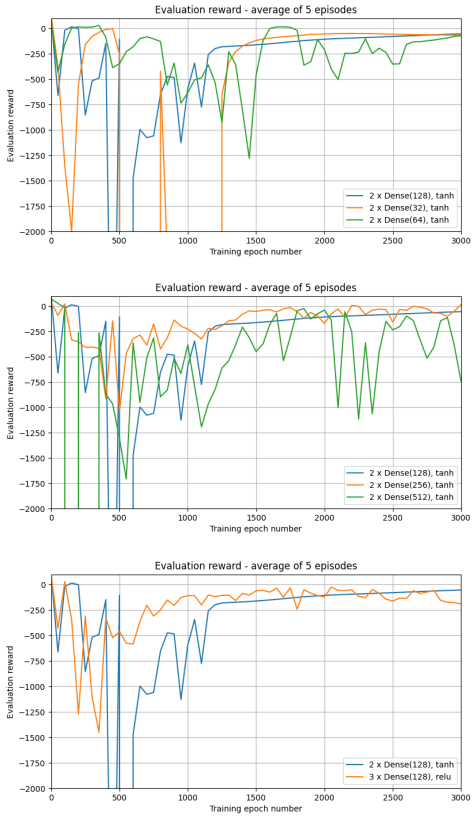

Figure 6. Different actor network architectures and relative performance. Agent trained: TD3, batch size: 512, learning rate: $3e-4$.

As [Fig. 6] shows, the best architecture is the one proposed in the work which, while enabling fast convergence, is still capable enough to correctly map the high dimensional state and action spaces.

Moreover, other important details considered in the training process were the batch size, set to 512, as we found that larger batch sizes guarantee a much better and more consistent training, and the learning rate. For off-policy agents we set a learning rate of $3e{-}4$, while for on-policy algorithms we set it to $1e{-}3$ in an effort to speed up the learning process. A lower learning rate should overall ensure a better but slower convergence. The optimizer used is Adam.

We also found a degree of randomness in the learning process, even with random seeds set to the same value, probably due to the non-deterministic nature of the AirSim environment.

3. Results

To evaluate the performance and response of the developed controller, a sensitivity analysis has been performed on

each one of the input variables, pushed until system failure. Similar testing has been performed to evaluate the system response to different wind conditions.

Experimental setup and dataset collection. TD3 Reinforcement Learning algorithm obtained the best policy to stabilize the quadcopter. The best model has been obtained after 5500 epochs of training. In order to make the controller robust to the wind action, a gaussian random wind with average value of $0m/s$ and standard deviation of $2.5m/s$ is added.
Two environments are used, in particular one for training and another one for evaluation. During training, multiple timesteps are collected. When the steps limit of 200 timesteps is reached, the environment is reset. During our experiments we use a batch-size of 512, since smaller batch-sizes decrease training performance. For TD3 we set the learning rate to $3e{-}4$ during the initial trials, increasing it to $1e{-}3$ in the final training, in order to speed up the longer learning process.

The entire replay buffer capacity is $1M$ timesteps. The initial data collection, executed with a random policy, bootstraps the learning process with 5000 steps. During each training epoch, 200 new time-steps are collected using the default collection policy of the agent. Such policy is the actor policy to which some exploration noise is added. As TD3 algorithm is an off-policy algorithm, during evaluation the learned policy is used as-is, differing from the collection policy as it lacks any added noise.
Every 50 epochs a set of 5 evaluation episodes is started, and the average reward is collected in order to evaluate the current agent policy and plot the learning curve. During each evaluation 400 time-steps are collected.

After the training process ends, further analysis on the learned policy are executed, in order to assess its performance and robustness. Multiple parameters in the evaluation environment are changed, such as the different initial orientation, the different initial positions and different wind speeds. During each each experiment, multiple datapoints are collected and dumped. They are later analyzed and their plots are saved.

Analysis and discussion. During the analysis of the wind effect, the wind has been gradually increased in steps of $2m/s$ for each experiment. With a margin of safety, the drone can stabilize and withstand wind in the range $[-20m/s, 16m/s]$ along the x axis, $[-22m/s, 20m/s]$ along the y axis and $[-6m/s, 6m/s]$ along the z axis. The performance is extremely remarkable, especially considering that most commercial drones nowadays cannot sustain winds stronger than 10 m/s. Only the best commercial drones, for instance "DJI Matrice 300 RTX", can currently sustain winds of up to 15 m/s [9]. Even then, the presented solution outperforms such numbers by a lot, in the simulations executed.

In [Fig. 7] the linear and angular velocity and acceleration tend to zero when the quadcopter reaches stability. The pitch angle approaches $-15deg$ to compensate for the wind, while roll and yaw angles tend to $0deg$.

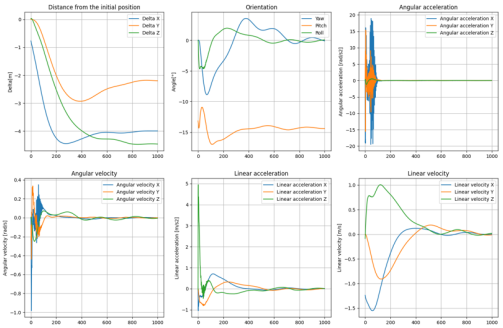

Figure 7. Constant wind of $-20m/s$ along x axis.

We then set the wind to zero and execute the sensitivity analysis on the drone orientation. The drone is again able to stabilize itself. The quadcopter was exceptionally good also in critical situations like the one in [Fig. 8, 9], with an initial roll of $120deg$. The drone was still able to recover gracefully from an initial roll of $180deg$, upside down. This goes to show the effectiveness of the controller. The controller can recover from a maximum initial roll of $180deg$, a maximum pitch of $90deg$ and a maximum yaw of $120deg$.

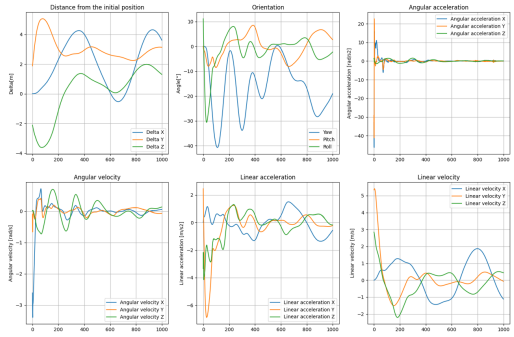

Figure 8. Initial roll, $120deg$. Analysis of the collected states.

While the objective of the work is to obtain the stability of the drone, the controller has been trained to reach a given position in 3D space as well. Since the reward function employs the Δx difference in current and initial position, and the off-policy learning uses a random policy to sample the state space - indirectly mapping the surrounding space as well -, the drone is able to move between 3D key-points even if such points are far. From the sensitivity performed, the drone is able to move autonomously between $[-20m, 10m]$ along the x axis, between $[-10m, 10m]$ along the y axis and $[-25m, 15m]$

Figure 9. Initial roll, $120deg$. Frames of the simulation: the drone is able to restore the orientation and position.

along the z axis.

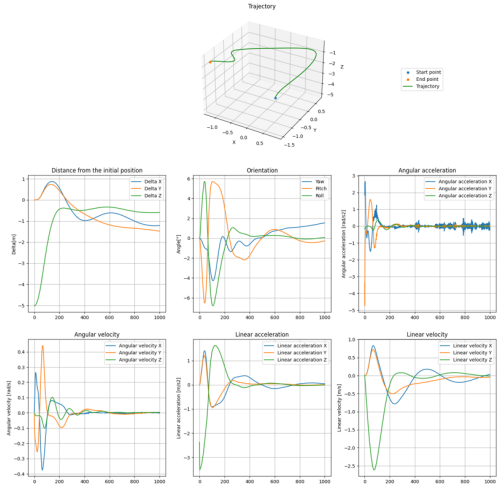

Figure 10. Behavior of the drone when initialized with a distance of $5m$ from the origin. Proper convergence of velocities and pose.

Overall, the learned policy far exceeds the expectations. In addition to being able to stabilize in normal conditions, the drone is able to recover the stability even with strong winds and high roll and pitch angles. The results obtained suggest that the controller can withstand stronger winds than the leading quadcopters in the market.

4. Limitations

While the controller is able to stabilize the quadcopter in most circumstances, the sensitivity analysis highlighted in few occasions high frequency components in the velocities and accelerations which could hinder the stabilization. Future works could address the issue implementing, for instance, an LSTM architecture which should be able to better account for past information, or by adding the previous action executed to the observation state or to the reward function as well, in an effort to minimize high-frequency adjustments in the propellers thrust.
Moreover, the final position of the drone is not always accurate: while this is partially expected, as the reward focuses on the orientation and angular velocity of the drone (that most influence its stability), such behavior might be attenuated via longer training or reward tuning.
While the controller is able to move the drone autonomously for several meters given a target, it is still limited to a certain distance from the target set point. This limitation is easily overcome by continuously updating the end position based on the current one, approaching the endpoint in sequential steps.

5. Conclusion

The work presented a robust deep RL based controller which can stabilize in extreme wind and even recover from upside down orientations. The controller satisfies the objectives of the project and shows the potential of reinforcement learning in the control field, especially for very complex systems as quadcopters. The approach allows not to specify any kinematic model, as it is directly learned by the agent, and to still perform better than traditional counterparts and previous reinforcement learning attempts. The controller is resistant to up to 5x stronger wind compared to previous works, and to 2x stronger wind compared to most commercial solutions. It ensures good performance with high deltas between initial pose and set pose.

Overall, the presented controller - with adequate real world testing - could play an important role in the stability recovery of a new generation of commercial solutions, with the safety of the individuals being a primary concern. Solid solutions are at the heart of the development in the field, as technology progress must also constantly improve the reliability of its applications.

Acknowledgements

Thanks to our teachers, Matteo Matteucci and Giacomo Boracchi, for inspiring our interest towards such interesting topics.
Thanks to the lecturers, Alessandro Giusti, Alessandro Lazaric, Jonathan Masci, Luigi Malago', who taught us with extreme competence.

A. Supplementary Material

All the source code, the presentation, the plots and videos are freely accessible and published in a GitHub repository. In the repository are the instructions to setup the simulation environment and perform both training and testing of the reinforcement learning algorithms. Following is the link to the online repository.
GitHub Repository

References

[1] Wikipedia. Pulse-width modulation. 1

[2] M. Eatemadi. Mathematical dynamics, kinematics modeling and pid equation controller of quadcopter, 2016. 1, 2

[3] Alvin Hou Fang-I Hsiao, Cheng-Min Chiang. Reinforcement learning based quadcopter controller. 2

[4] Alan F. Lynch Zifei Jiang. Quadrotor motion control using deep reinforcement learning. *Journal of Unmanned Vehicle Systems*, 2021. 2

[5] Roland Siegwart Marco Hutter Jemin Hwangbo, Inkyu Sa. Control of a quadrotor with reinforcement learning. 2

[6] Stone Cheng Chen-Huan Pi, Wei-Yuan Ye. Robust quadrotor control through reinforcement learning with disturbance compensation, 2021. 2

[7] Chris Lovett Ashish Kapoor Shital Shah, Debadeepta Dey. Airsim: High-fidelity visual and physical simulation for autonomous vehicles, 2017. 3

[8] Timo Stöttner. Why data should be normalized before training a neural network, 2019. 5

[9] Paul Posea. Can drones fly in strong winds?, 2022. 6