Self-Learning Drone Navigation Using Deep Reinforcement Learning

Per Blåwiik[1]

Abstract
This report covers the theory and the implementation of training an AI-agent to navigate a quad-copter drone through an obstacle course. The drone and the course is represented in a virtual 3D environment and the agent was trained using deep reinforcement learning methods. A curriculum learning strategy was utilized to accelerate and optimize the training by dividing it into several lessons that gradually becomes more complex. The proximal policy optimization algorithm together with a curriculum strategy containing six difficulty levels was implemented to successfully train the agent to consistently solve the specified problem. The results indicate that the environment can be further developed by simply adding more difficulty levels to the curriculum strategy, letting the agent resume its training from where it ended.
Source code: https://github.com/perblawiik/drone-ai

Video: https://www.youtube.com/watch?v=WaMEi7R5w8Y

Author
[1] *Media Technology Student at Linköping University, persj146@student.liu.se*

Keywords: Deep Reinforcement Learning — Proximal Policy Optimization — Curriculum Learning

Contents

1. Introduction

The aim of the project presented in this report was to train an AI-agent to navigate a quad-copter drone through an obstacle course while performing a simple task, using reinforcement learning methods. The drone is a coarsely simplified model based on a remote controlled quad-copter and the course is represented by a virtual 3D environment.

The reinforcement learning method chosen to train the AI-agent is called *proximal policy optimization* (PPO), first introduced by the *OpenAI* research team Schulman et al. [1] in 2017. The PPO algorithm is one of the most popular and efficient reinforcement learning approaches for training AI-agents to solve tasks in game-like environments as well as control problems in the field of robotics. PPO is an on-policy algorithm which can be used in environments with both discrete and continuous action spaces, and supports parallelization for accelerated training.

To increase the efficiency and optimization of the training, a *curriculum learning* strategy was also implemented. The idea of curriculum learning is to break down a complex task into several lessons, which gradually becomes more advanced.

This report first covers the theory behind reinforcement learning, PPO and curriculum learning, followed by the details specific to the implementation of the project. The results are then presented together with training statistics, and finally, the report is concluded with a discussion and conclusion of the work.

2. Theory

In this section, the theoretical foundation of the project is presented, including an introduction to reinforcement learning, and the theory behind *proximal policy optimization* (PPO) algorithm as well as the curriculum learning strategy.

2.1 Reinforcement Learning

The idea of reinforcement learning is to program (train) an agent to solve a task by using positive and negative rewards, without specifying any details on *how* to do it. The standard form of reinforcement learning applied in the field of artificial intelligence, involves an agent that is connected to an environment via *observations* and *actions* [2].

Observations are the agent's perception of the environment, which can range between partial knowledge received from specific sensors, to full knowledge of everything. The observations are used to represent the state of the environment.

Via *action* inputs, the agent can interact with the environment. Each action input changes the current state of the environment and generates a new state as output. Depending on the output, the agent usually receives a positive or negative reward (reinforcement), which can be saved in the agent's memory for future *exploitation*. On the long run, the agent's behaviour should converge to selecting actions that tend to yield the highest sum of rewards.

An important part of the reinforcement learning model is the exploitation versus exploration ratio. The exploration part of the agent's *brain* is based on random actions, while the exploitation part utilizes the memory of previous experiences to choose actions. In order for the agent to discover which behaviour give the highest reward, the environment must be explored, and thus, random actions are necessary. However, since the goal is to train the agent to predict the best actions to solve a task, the exploration ratio should decrease on the long run.

How to exactly manage the trade-off between exploration and exploitation is not trivial, and testing is usually required to find the best configuration. For example, if the exploration is removed too early during the training, the agent might learn a sub-optimal strategy since it never discovered a more favorably sum of rewards.

2.2 Proximal Policy Optimization

Proximal policy optimization (PPO), proposed by Schulman et al. [1], is a novel reinforcement learning algorithm based on *policy gradient methods* [3]. The biggest advantages of PPO over standard policy gradient methods are better performance, regarding computational complexity, and it is significantly simpler to implement.

The main idea behind PPO is to make the biggest possible improvement on a decision-making policy using the currently available data, without causing a performance collapse. In practise, the method can be crudely explained by the following steps:

1. An agent collects a small batch (multiple time steps) of experiences by interacting with the environment.
2. The collected batch is then used to update the decision-making policy.
3. Every time the policy is updated, the collected batch is discarded.
4. Collect a new batch of experiences using the updated policy.

Note that each batch of experiences is only used to update the current decision-making policy once.

2.2.1 Clipped Surrogate Objective

To avoid making too large updates, which can cause the new policy to diverge far away from the old, a clipping method is normally used (alternatively, a penalty variant similar to *trust region policy optimization* (TRPO) [4] is used). The clipping yields less variance to ensure a more stable training, at the cost of minor biases.

Using the clipping approach, the policies π are updated by collecting minibatches (multiple steps) of SGD (stochastic gradient descent) to maximize the objective θ according to Equation 1.

$$\theta_{k+1} = \underset{\theta}{maximize}\ \hat{E}_t[L(s,a,\theta_k,\theta)], \tag{1}$$

where the k subscript denotes the old minibatch, s is the current state, a is the action, $\hat{E}$ indicates the empirical average over the batch samples, and L is the surrogate objective given by Equation 2.

$$L(s,a,\theta_k,\theta) = min\Big(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), g(\varepsilon, A^{\pi_{\theta_k}}(s,a))\Big), \tag{2}$$

where A is the advantage function of the action-state pair, ε is a hyperparameter that influences how far away the new policy can diverge from the old policy (the clipping threshold), and the function g is defined by Equation 3.

$$g(\varepsilon, A) = \begin{cases} (1+\varepsilon)A, & \text{if } A \geq 0 \\ (1-\varepsilon)A, & \text{if } A < 0 \end{cases} \tag{3}$$

Equation 2 looks complex, but if the advantage A is positive, the expression reduces to Equation 4, and if A is negative, it reduces to Equation 5. The term $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ is a ratio that measures the difference between the old policy π_{θ_k} and the new policy π_θ.

$$L(s,a,\theta_k,\theta) = min\Big(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1+\varepsilon)\Big)A^{\pi_{\theta_k}}(s,a) \tag{4}$$

$$L(s,a,\theta_k,\theta) = max\Big(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)},(1-\varepsilon)\Big)A^{\pi_{\theta_k}}(s,a) \quad (5)$$

The intuition of Equation 4 is, that the objective will increase if $\pi_\theta(a|s)$ *increases* (an action becomes more likely), but the term is clipped to $(1+\varepsilon)A^{\pi_{\theta_k}}(s,a)$. In Equation 5, the objective will increase if $\pi_\theta(a|s)$ *decreases* (an action becomes less likely), but the term is clipped to $(1-\varepsilon)A^{\pi_{\theta_k}}(s,a)$. Thus, the new policies do not diverge too far away from the old policies, by clamping the objective value in the interval $[(1-\varepsilon)A,(1+\varepsilon)A]$.

2.2.2 The Actor-Critic Model

The PPO uses an Actor-Critic approach, which involves an *actor* model and a *critic* model, when training the agent. Briefly explained, the actor model learns to predict what action to choose for the currently observed state, while the critic model learns to evaluate the outcome of the actions taken (in terms of positive or negative rewards) and returns the feedback to the actor.

The PPO algorithm proposed by Schulman et al. [1], using N parallel actors for each iteration, collecting data of T timesteps, is shown below. The surrogate L with respect to the objective θ is optimized using minibatch SGD for K epochs.

Algorithm 1: PPO, Actor-Critic Style

for *iteration=1, 2, ...* **do**
  for *iteration=1, 2, ..., N* **do**
    Run policy π_{old} in environment for T steps;
    Compute advantage estimates $\hat{A}_1,...,\hat{A}_T$;
  end
  Optimize surrogate L wrt the objective θ, with K epochs and minibatch size $M \leq NT$;
  Update previous objective $\theta_{old} \leftarrow \theta$;
end

2.3 Curriculum Learning

A big problem in reinforcement learning is the amount of training time needed for solving tasks of high complexity. Since reinforcement learning is widely based on random exploration, some challenges can be daunting or practically impossible for a self learning agent to overcome.

Curriculum learning proposed by Bengio et al. [5], is a strategy for accelerating the training process by gradually increasing the complexity based on the learning progress of, in the case of reinforcement learning, the AI agent. The main goal is to find better local minimas as well as finding a faster convergence.

Just like how children first learns the basics before moving on to more advanced material during elementary school, curriculum learning applied in machine learning algorithms first introduces a simple problem for the model to solve and gradually increases the complexity. This way, previously encountered concepts can be utilized to ease the learning curve when facing new challenges.

The curriculum learning strategies defined by Bengio et al. [5], leave most of the work to the teacher. The key aspect in designing curriculum strategies is to recognise how the problem should be subdivided into lessons and how the knowledge to solve one problem can benefit the learning of solving another.

3. Method

In this section, the details on the implementation of the project is covered. First the environment is explained, followed by the drone controls, observation system, reward system, and the curriculum learning strategy.

The proximal policy optimization method was implemented using the Unity integrated library called *ML-Agents*, which includes a ready-to-use template that contains all necessary parts of the algorithm, as well as a configuration system for setting the training parameters. This library was mainly used to facilitate the construction of a customized 3D environment using the game engine Unity. ML-Agents also supports automated curriculum learning which means that the environment can automatically change during training based on specified threshold values.

3.1 The Environment

The goal of the project was to train an agent to navigate a quadcopter drone through an obstacle course while performing a simple task. When designing the environment, the learning goal of the agent was divided into several sub-goals:

- Learn to gain height
- Learn to leave the starting area and land in a goal area.
- Learn to avoid obstacles.
- Learn to adapt the flight path based on the environment.
- Learn to perform a task.

To encourage the agent to keep the drone in the air, the start and goal areas consist of cylindrical platforms reaching far from the ground level. This results in that the drone is airborne immediately after leaving the starting area. Since there is nowhere else to land except for the goal platform after leaving the start platform, both platforms are distinguished from the rest of the environment.

The entire obstacle course is contained inside four walls to force the agent to be aware of its surroundings. Additionally, a large wall with a small opening separates the start and the goal so that the drone has to be carefully navigated through it. The four walls and the center wall is the obstacles that the agent needs to learn to avoid while navigating the drone.

The task to solve in the environment was represented by a floating sphere checkpoint that needs to be collected before

landing in the goal area. This task forces the agent to search the environment while navigating through obstacles on the way to the goal area.

Finally a randomization system was implemented to randomize the positions of the start and end platforms, the position of the wall gap, and the position of the checkpoint. Thus, each training episode is unique, which forces the agent to adapt its flight path based on observations of the environment.

3.2 Actions

In order for the agent to explore and interact with the environment, a set of available input actions are necessary.

The quad-copter drone is represented by a rigid body and is moved by forces applied to it. The idea is that the agent controls the drone with a calibrated remote controller device like a human would. For example, this means that the height is computed by a simple PID controller system so that the agent only needs to be concerned about the control inputs (not the actual forces involved). The height $u(t)$ is computed based on the control function given by Equation 6 [6].

$$u(t) = K_P \cdot e(t) + K_I \cdot \int_0^t e(\tau) d\tau + K_D \cdot \frac{de(t)}{dt} \tag{6}$$

where K_p is the proportional gain, K_I is the integral gain, K_D is the derivative gain, e is the error signal (the difference between desired height and measured height), τ is variable of integration, and t is the time input.

To make the training easier for the agent, the drone controls are limited to five inputs: move vertically up or down, rotate left or right, and move straight forward.

3.3 Observations and Sensors

Some form of observation system is needed in order for the agent to perceive the environment and make decisions based on the current states. To make observations, the drone was equipped with a set of ray perception sensors which can identify obstacles as well as measure the distance to them based on ray casting techniques.

Alternatively, cameras can be used as sensors, however, this usually requires some kind of image processing which was out of scope for this project.

3.4 Reward System

The idea of reinforcement learning is to let an agent explore an environment by itself and, based on the actions taken, provide it with positive or negative rewards to reinforce certain behaviours. The total score at the end of an episode is the sum of all negative and positive rewards, and reflects how well the agent behaved. Chains of actions that consistently leads to a high average score will ultimately be prioritized, and actions that leads to a lower average score will be avoided.

3.4.1 Step Penalty

Typically a time step penalty is used to reinforce the agent to complete a task in minimum time. In this project a time step penalty p_t, proportional to the maximum number of steps N in an episode was used. If the time runs out during a training episode, the sum of the step penalties p_t will result in negative one, according to Equation 7,

$$p_t = -\frac{1}{N} \tag{7}$$

A critical skill for the drone agent was to learn how to leave the starting platform by moving straight up. To speed up this learning process, an additional step penalty was added for each time step on which the drone stays on the platform. This additional penalty reinforces the drone agent to leave the platform as soon as possible by earning a higher average score for doing so.

3.4.2 Checkpoints

One part of the learning goal was that the agent should be able to solve a task while navigating through an obstacle course. In the final implementation the task was to collect a randomly placed checkpoint before landing on the goal platform. To encourage this behaviour, a positive reward is earned if the agent collects the checkpoint.

3.4.3 Goal Location

To make sure that the agent learns where to move, a positive reward is earned when the agent finds the goal platform. Additionally the current training episode ends, and consequently, the step penalty is terminated.

To speed up the learning process, the agent is rewarded with a bonus score if the checkpoint was collected before reaching the goal platform. Note that this rule not only reduces the learning time, but also decreases the chance of learning a bad pattern. A bad pattern could be that the agent fails to learn that it should collect the checkpoint and instead takes the shortest path directly to the goal location for a fast finish time.

3.4.4 Obstacles

If the agent accidentally fly into a wall or similar, the current training episode will immediately end with a score of negative one. This is incorporated to make sure that the agent carefully avoids crashing into obstacles and loosing control.

If this rule was not implemented, there is a possibility that the agent learns to take unnecessary risks in order to get a fast finish time. This would not be tolerable if the goal is to implement the agent to fly a real-life quad-copter drone.

3.5 Training Acceleration

Aside from accelerating the training process by using curriculum learning, parallel training was utilized. Parallel training is supported by the library ML-Agents and works by simply adding multiple identical copies of the environment in the scene before initiating the training. All agents share the same policy network meaning that the same amount of training steps, compared to a single agent in a single environment, can be achieved in a proportionally faster time. An example of how this is implemented in Unity is shown in Figure 3.

3.6 Curriculum Strategy

The curriculum learning strategy was implemented based on internal difficulty levels. The idea was to increase the difficulty during the training based on the agent's average score progress. This translates to: as soon as the agent has learned to consistently master level one, the environment changes to level two, and so on.

The environment has a total of six difficulty levels (levels 1-5 is illustrated in Figure 1). Levels 1-3 involves different spawning areas for the start and end platforms, where the spawning area as well as the distance between them increase each level. Additionally, the height of the goal platform is incremented on level 2 and 3.

In levels 4-5, two vertical walls in the middle of the room has been introduced with different horizontal gap sizes. On level six, horizontal walls are added which results in a square gap. Note that the placement of the gap is randomly placed for all levels 4-6, where in levels 4-5 the gap is sampled in one dimension (horizontal), and for level 6 the gap is sampled in two dimensions (horizontal and vertical).

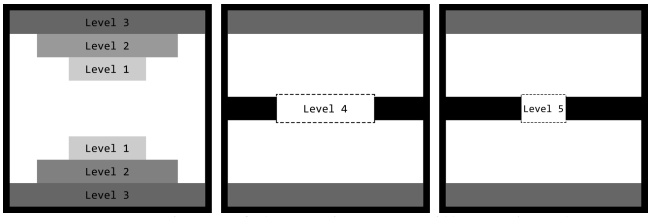


Figure 1. Top views of the environment illustrating the different difficulty levels used for curriculum learning. The left figure shows levels 1-3 where the gray squares represents spawning areas for the start and end platforms. The other two figures (middle and right) shows the horizontal size of the gap in the obstacle wall.

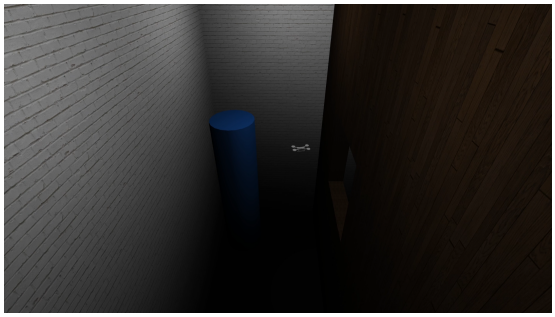
The idea of the difficulty level strategy was to gradually increase the complexity of the environment to not overwhelm the agent, yielding in a faster and higher convergence of the average score.

4. Result

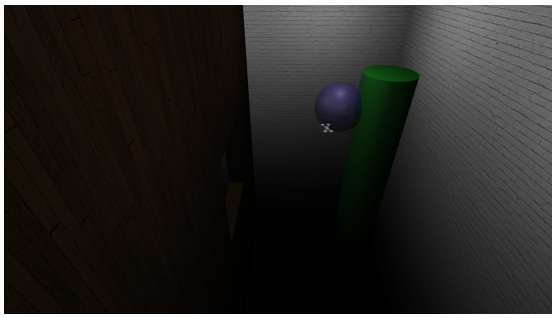
The final result was a policy agent trained based on the methods described in Section 3, that consistently managed to leave the start platform, navigate through an obstacle, collect the checkpoint, and finally land on the goal platform. The total training time was 15 million steps trained with a curriculum learning strategy. The hyperparameters and neural network settings that was used are available in Appendix A.

The environment configured to the final difficulty level is shown in Figure 2. The screen captions show an agent controlled quad-copter drone leaving the start platform, navigating through the obstacle and collecting the checkpoint.

The training process was accelerated using 20 parallel environments. An example of the parallel training, covered in Section 3.5, is seen in Figure 3.



(a)



(b)

Figure 2. An exhibition of the environment set to difficulty level 6. The room is divided by a wall with a square gap, 2(a) shows the drone leaving the starting platform headed towards the gap, and 2(b) shows the drone navigating towards the checkpoint after passing through the gap.

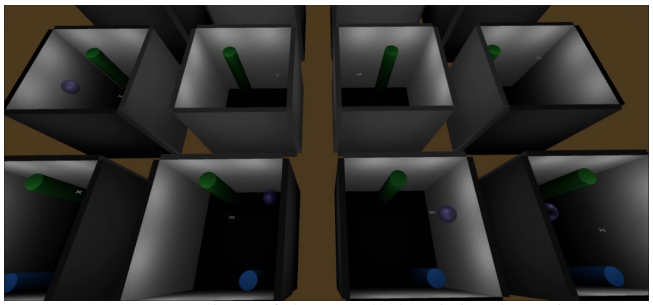


Figure 3. An overview of the parallel training by using multiple copies of the environment. The blue platform is the starting location, the green platform is the goal location and the purple sphere is the checkpoint to collect. The environments were currently set to difficulty level 3.

Theoretically, the maximum possible average score for an agent is 2. The score is based on how fast an episode is finished, and whether or not the checkpoint was collected before reaching the goal platform.

In Figure 4, a graph of the training progress is shown. Note that up to 2 million steps, difficulty levels 1 to 3 were traversed automatically based on a threshold value of 1 for the average episode score. The completion of levels 4, 5, and 6 were manually managed since no suitable threshold value had been determined. For example, when the training of level

4 showed a stable convergence, the training was manually paused, and then resumed on level 5. The difficulty levels are explained in Section 3.6.

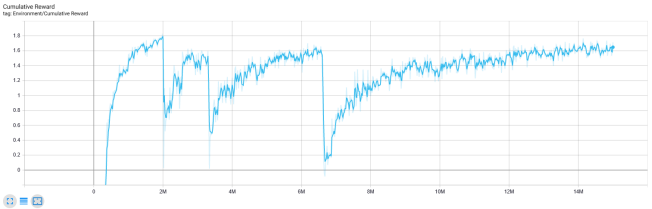


Figure 4. The training progress of an agent using the curriculum learning strategy. The y-axis represents the average episode score and the x-axis represents the number of training steps. At two million steps, difficulty level 3 was completed. The following curve dips represents the beginning of level 4, 5, and 6 respectively. The training was terminated after 15 million steps

To compare the efficiency of utilizing curriculum learning, training sessions using a static difficulty level was done. In Figure 5, the blue curve represents an agent trained using curriculum learning from level 1 to 5, and the red curve represents an agent trained on level 3 from start to finish. At 2 million training steps and with an average score of 1.8, level 4 was manually initiated for the blue agent, while the red agent at the same step had an average score of 0.6. At the end of the graph, both agents had an average score of 1.5 and the blue agent was completing level 5.

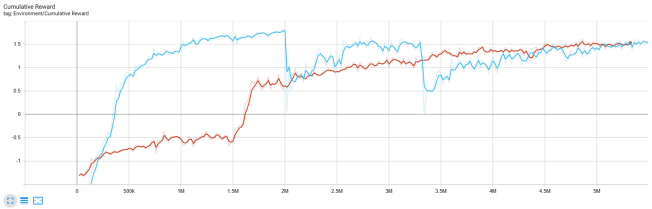


Figure 5. A comparison between two training sessions, where the blue curve used the curriculum learning strategy, and the red curve trained on level 3 consistently. Note that at the first dip on the blue curve, level 4 was manually initiated.

Figure 6 showcases the inconsistency of overcoming important learning milestones. The curves show the progress of training two agents with identical parameter settings without using curriculum learning. Based on the random exploration aspect of reinforcement learning, the red agent made a breakthrough leading to a spike in higher average scores faster than the blue agent. The milestone in this case was that the agents learned to land on the goal platform. Both environments used a static difficulty level of 3.

5. Discussion

The final environment used to train the agent contained a much less complex obstacle course than what was initially the ambition, however the result show that it can easily be modified by incorporating more difficulty levels to the curriculum

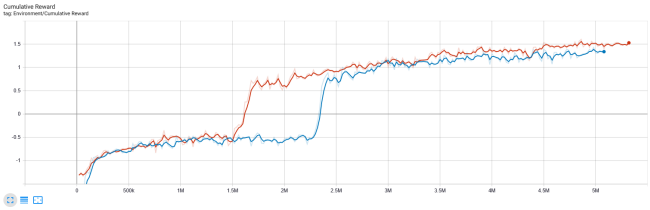


Figure 6. A comparison between two different training sessions without curriculum learning, using identical parameter settings. The y-axis represents the average episode score and the x-axis represents the number of training steps.

strategy. For example, the size of the room can be expanded, more walls (obstacles) could be added, and the task could be more advanced. A more advanced task would be to have the agent pick up the checkpoint and drop it off at a certain location, before moving to the goal platform.

The main reason for constraining the complexity of the obstacle course was to keep the time needed for training low. This was necessary for the project to be feasible since the development of the environment including the reward system, action system, and the curriculum strategy was very time consuming. The total time for the final training session was about 3.5 hours.

Although the curriculum learning was successful, it could probably better optimized. By looking at Figure 4, it is clear that the step from difficulty level 5 to 6 (the last curve dip) is quite dramatic in terms of the drop in score. This might be a consequence of the fact that the vertical walls are not introduced until the final level, yielding a large skill gap between level 5 and level 6.

Reinforcement learning assisted with curriculum learning is a powerful tool for training AI-agents to solve complex tasks. One of the biggest advantages of reinforcement learning is that an agent can be retrained to adapt to new problems. With a carefully designed environment, reward system, and curriculum strategy, an agent could potentially be trained to solve a general task, prepared to advance into branches of more specific problems. If this would be possible, the training time for adapting the agent to solve new problems could be considerably reduced.

6. Conclusion

An AI-agent was successfully trained to consistently and safely navigate a drone through an obstacle course while performing a simple task. The agent learned to solve the task by interacting with the environment, using reinforcement learning methods. Specifically, the proximal policy optimization algorithm was used, and implemented with the game engine Unity and the library called ML-Agents.

Result show that curriculum learning is a powerful strategy for optimizing and accelerating the learning process of the agent. By using a difficulty system containing six levels, the task started out simple and gradually became more complex, easing out the steep learning curve.

References

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[2] Leslie Pack Kaelbling and Michael L. Littman. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[3] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *ICML*, 2016.

[4] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *ICML*, 2015.

[5] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. *ICML 09: Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.

[6] Kiam Heong Ang and Gregory Chong. Pid control system analysis, design, and technology. *IEEE Transactions On Control Systems Technology*, 13(4), 2005.

Appendices

A Parameters

Hyperparameter	Value
Batch size	256
Buffer size	10240
Learning rate	0.0003
β	0.005
ε	0.2
λ	0.95
Epochs	3
Learning rate decay	Linear

Table 1. The PPO hyperparameters used in the final training of the drone agent. The parameter β is the strength of the entropy regularization (for proper exploration of the environment), ε influences how far the new policy can diverge from the old policy, and λ is a regularization parameter used for calculating the General Advantage Estimate.

Neural network settings	Value
Hidden units	256
Number of hidden layers	2
Vector normalization	False
Encoder type	Two convolutional layers

Table 2. The PPO neural network settings used in the final training of the drone agent.