Reinforcement Learning for the Control of Quadcopters

Alexandre Sajus
CentraleSupélec
asajus47@gmail.com

Abstract

Quadcopters are used for many applications (racing, filming, combat...) and can be flown manually and autonomously. Currently, quadcopter controls are generally based on PID controllers. Sensors on the drone register the drone's current position, rotation, speed of its rotors... and the flight controller uses these inputs to generate voltage instructions for its motors. Reinforcement Learning has proved its ability to provide control policies in unstable environments such as the CartPole environment. In this work, we compare the capabilities of Reinforcement Learning for the task: control of a quadcopter to navigate to a waypoint. To this end, we create a 2D rigid-body physics simulated environment and we compare the performance of Reinforcement Learning agents and PID-controlled or Human agents.

The project's GitHub repository can be found at https://github.com/AlexandreSajus/Quadcopter-AI.

1. Introduction

The goal of this project is to compare three approaches to control a Quadcopter in the task of navigating to multiple waypoints without crashing:

- Human: a human directly controls the thrust of the rotors using the keyboard
- PID: the quadcopter navigates autonomously towards its waypoints by using his position and the waypoint's position as input and using PID controllers to output the thrust of the rotors
- Reinforcement Learning: the quadcopter navigates autonomously towards its waypoints. The thrust of the rotors is generated from the quadcopter's position and the waypoint's position through the action policy of an agent that trained itself on a similar environment.

1.1. State of the Art

Quadcopters nowadays are mostly controlled by humans and control theory. PID controllers are often used to control the motor power of each propeller on the quadcopter by looking at sensor information. This method of control allows with a bit a tuning to stabilize the drone in flight but control theory can also be used for more complex tasks such as autonomous navigation like in Adaptive Digital PID Control of a Quadcopter with Unknown Dynamics [1] which develops a drone autopilot using control theory.

Using Reinforcement Learning to control quadcopters has been successfully attempted before like in A Zero-Shot Adaptive Quadcopter Controller [4] in which a robust flight controller is developed using Reinforcement Learning. The biggest drawback of this method, in contrast with PID control which only requires a few hours of manual tuning, is that it requires a lot of training. Since that training can often not be done in the real world (to avoid crashing drones or time consumption), a virtual model of the drone and its environment needs to be created for the training.

Reinforcement Learning agents started out with simple algorithms which could beat Atari games by optimally choosing discrete actions like in Playing Atari with Deep Reinforcement Learning [3]. Nowadays, state of the art agents such as SAC presented in Asynchronous Methods for Deep Reinforcement Learning [2] are capable of learning complex environments faster and in a more stable manner. These agents can also work with continuous action spaces.

1.2. Environment

To evaluate the different control approaches, we use the Balloon environment: a 2D rigid-body physics simulated environment where a 2D Quadcopter has to navigate to randomly generated waypoints represented as balloons.

1.3. Physics

At each step of the simulation, the agents have to provide values between -0.003 and 0.083 (values chosen arbitrarily) for the thrust of their left and right rotors. These values are then used to move the quadcopter according to physics constants, rigid-body physics, Newton's second law of mo-

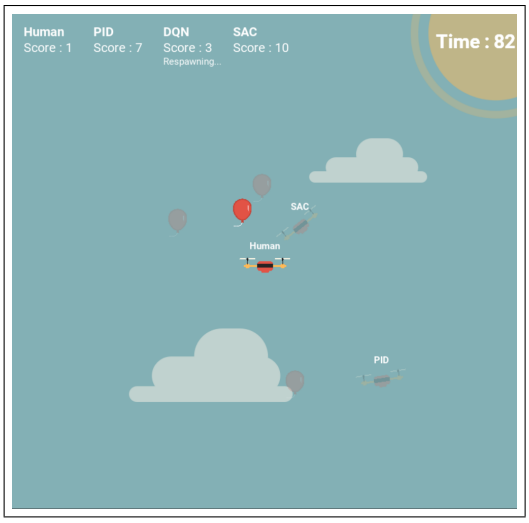

Figure 1. The Balloon Environment

tion and finite differences approximations. The steps at *t* to calculate the position of the drone at *t+1* are the following:

1. Calculate the drone's accelerations from the input thrust of its rotors according to rigid-body physics:

$$\ddot{x} = \frac{-(T_l + T_r)\sin(\theta)}{m}$$

$$\ddot{y} = \frac{-(T_l + T_r)\cos(\theta)}{m} + g$$

$$\ddot{\theta} = \frac{(T_r - T_l) \cdot l}{m}$$

Rigid-Body Physics Equations

with:

- $\ddot{x}; \ddot{y}; \ddot{\theta}$ the accelerations of the drone (on the x axis, the y axis and the angular acceleration)
- $T_l; T_r$ the input thrusts of the left and right rotors
- θ the angle of the drone following the z axis
- $m; g; l$ are constants chosen arbitrarily to represent mass, gravity and the quadcopter size.

2. Derive the speed and then the position of the drone using finite differences approximation with $dt = 1$

$$\dot{x}(t+1) = \ddot{x} \cdot dt + \dot{x}(t)$$

$$x(t+1) = \dot{x} \cdot dt + x(t)$$

Finite Differences Equations

1.4. Scoring

The simulation steps 60 times per second and has a time limit of 100 seconds. In that time limit, agents have to collect has many balloons has possible. Reaching a balloon rewards the agent with one point and respawns the balloon at a random position within the simulation window. If the drone gets too far from the balloon, we consider that it has crashed and it will respawn to its initial position after a 3-second respawn timer. The score is the amount of balloons collected within the time period.

2. Agents

Here we will explain how the different agents (Human, PID, Reinforcement Learning) were developed for the Balloon environment.

2.1. Human Agent

The human agent is very simple: the human player controls the drone using the arrow keys of his keyboard (keys can be pressed simultaneously):

- Thrust initializes at $T_l = 0.04; T_r = 0.04$
- "Up" results in $T_l \mathrel{+}= 0.04; T_r \mathrel{+}= 0.04$
- "Down" results in $T_l \mathrel{-}= 0.04; T_r \mathrel{-}= 0.04$
- "Left" results in $T_l \mathrel{-}= 0.003; T_r \mathrel{+}= 0.003$
- "Right" results in $T_l \mathrel{+}= 0.003; T_r \mathrel{-}= 0.003$

This control scheme is hard to use at first but, after training, it allows the player to navigate waypoints and crash rarely. This agent will be used as a baseline to evaluate the performance of other agents.

2.2. PID Agent

This agent takes inspiration from flight controllers who also use PID controllers. Control here is determined by the positional error between the drone and its target.

The vertical error is used to control the thrust of both rotors (if both rotors are thrusting at 0.08 power, the drone will go up)

- The vertical distance between the drone and the target is sent to a PID controller to output an optimal vertical speed
- The difference between the current vertical speed and the optimal vertical speed is sent to a PID controller to output a thrust amplitude sent equally to both rotors (betweeen 0.08 and 0)

The horizontal error is used to control the difference between the thrust of each rotor (if the left rotor if at -0.003 power and the right at +0.003, the drone will angle left and then hover to the left)

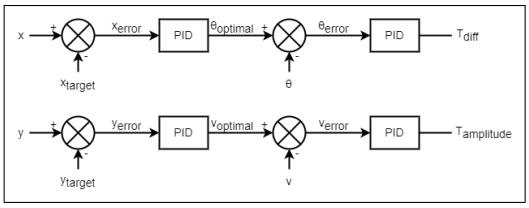

Figure 2. Four PIDs in cascade controlling the quadcopter

- The horizontal distance between the drone and the target is sent to a PID controller to output an optimal angle for the drone
- The difference between the current angle and the optimal angle is sent to a PID controller to ouptut the difference of thrust between both rotors (between -0.003 and 0.003)

The proportional, derivative and integral coefficients of the 4 PID controllers have been manually tuned to maximize performance in the Balloon environment. This agent will also be used as a baseline to evaluate the performance of other agents.

2.3. Reinforcement Learning Agent

This agent trains itself on multiple episodes of the game by testing different actions and learning from the rewards it gets. From this training, it derives an action policy: it takes as inputs observations from the environment and returns the thrust of each rotor.

2.4. RL Agent: Training Environment

The training environment called droneEnv is very similar to the balloonEnv but what the agent can observe, how he can act and how he is rewarded is carefully chosen to ensure better training. We must make sure that observations are easy to understand and relevant to the task at hand, actions have to be simple enough to use but not restrictive and rewards have the goal of motivating the agent to learn balloonEnv but also to guide him in his training. Therefore, we have made the following decisions:

For observations, the agent needs information about his current position and speed to make his decisions:

- distance to target : the distance between the drone and the target
- angle to target: the angle between the drone's vertical axis and the vector from the drone pointing to the target
- angle to up: the angle between the drone's vertical axis and the world's vertical axis (to make it understand gravity)
- velocity: velocity of the drone
- angle velocity: the angle between the velocity vector and the world's vertical axis
- angle target and velocity: the angle between the velocity vector and the vector from the drone to the target

For rewards, the agent needs to learn to collect balloons as fast as possible but encouraging it early to not crash can make its training faster:

- At each step of the simulation, the agent is awarded $+\frac{1}{60}$ as a reward for surviving (60 is the steps per seconds)
- At each step of the simulation, the agent is awarded $-\frac{dist}{100 \cdot 60}$ as a punishment for being far from the target (100 represents a normalizing factor for the distance)
- Each time the drone collects a balloon, the agent is awarded $+100$ as a reward
- Each time the drone crashes (is too far from target), the agent is awarded -1000 as a punishment for crashing which ends the episode

2.5. RL Run 1: DQN

The first attempt of training a reinforcement learning agent was used to check that the training pipeline was working and to establish a baseline. For this attempt, we use the default DQN from stable-baselines3 as our agent. DQN is a simple algorithm that is limited to discrete actions that we have to define:

- "Pass" results in $T_l = 0.04; T_r = 0.04$
- "Up" results in $T_l = 0.08; T_r = 0.08$
- "Down" results in $T_l = 0; T_r = 0$
- "Left" results in $T_l = 0.0394; T_r = 0.0406$
- "Right" results in $T_l = 0.0394; T_r = -0.0406$

The agent only chooses an action every 5 steps of simulation, this chosen action is used for the 5 steps.

Training was done on 1M steps (decision steps where the agent made a decision) and was tracked using Weights and Biases.

The Experiment Tracking plot shows the agent slowly learns in the beginning. The exploration rate is too high at the start which causes the drone to often take random actions which often lead to a crash with a punishment of 1000 points which explains why the rewards are close to -1000 at the beginning.

When the Exploration Rate hits 0.4, the agent seems to have enough control and understanding of the environment

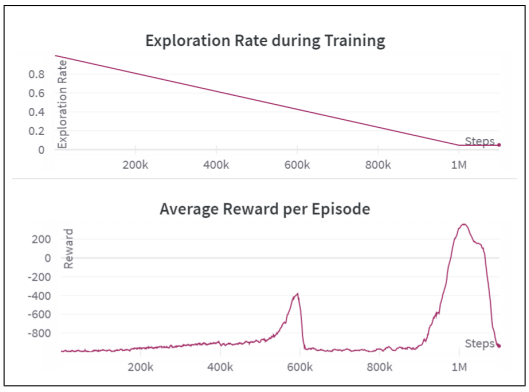

Figure 3. Run 1: DQN Experiment Tracking Results

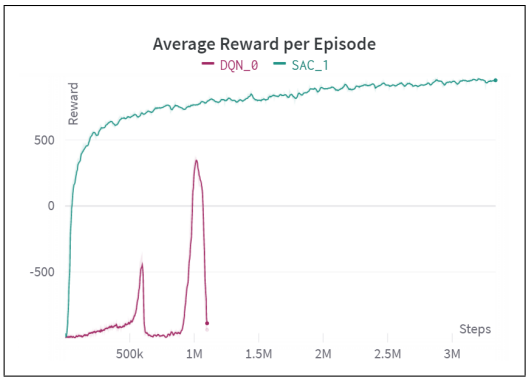

Figure 4. Run 2: SAC Experiment Tracking Results

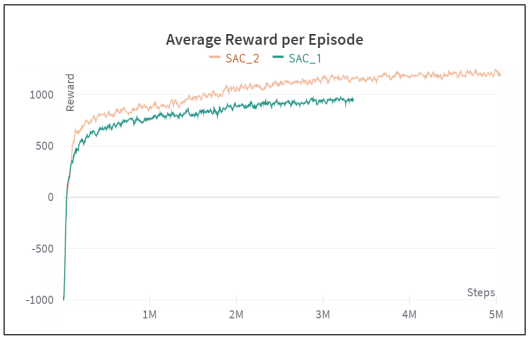

Figure 5. Run 3: SAC Experiment Tracking Results

to not crash as often with an average reward of -400. Unfortunately, this behavior does not last and the agent keeps crashing repeatedly between 600k and 800k steps.

At around 1M steps, the agent hits his highest average reward of 345 and unfortunately starts crashing again right after.

This result is promising as it shows that reinforcement learning agents can learn successful behavior in this unstable environment. But unfortunately, the training of DQN here is very unstable which could be explained by the fast variation in exploration rate (the agent might have adapted to a specific exploration rate around 0.4) and the discrete choice of actions.

2.6. RL Run 2: SAC

SAC is a much more advanced algorithm and is currently state of the art when it comes to continuous actions. The chosen action space is the following: the output of the agent is two floats action0 and action1. action0 represents the thrust amplitude applied to both rotors. action1 represents the relative difference of thrust applied to both rotors. The thrusts can be derived from the actions using the following formulas:

- $T_l = action_0 \cdot 0.04 + action_1 \cdot 0.0006 + 0.04$
- $T_r = action_0 \cdot 0.04 - action_1 \cdot 0.0006 + 0.04$

Conversion of Actions to Thrusts Equation

Training was done on 3.3M steps and was tracked using Weights and Biases.

On the experiment tracking plot, we can see that SAC performs much better than DQN on this task. At only 50k training steps, the average rewards exceeds 0 meaning that SAC learned to not crash. The average rewards keep increasing over time which means that the training is stable over time. After that point rewards slowly increase as SAC learns to collect balloons faster and faster. Eventually the rewards converge at around 950 per episode which translates to around 9 balloons collected every 20 seconds.

This result shows that SAC is very capable of learning on the droneEnv environment. But when diagnosing SAC's behavior, we could find that SAC was often operating at its physical limits with action1 often hitting its maximal values of -1 or 1. This could mean that SAC's performance is bottlenecked by the action shaping imposed on it. We wanted to verify that hypothesis by doing a third run.

2.7. RL Run 3: Higher Differential Thrust

Here we keep SAC but change the Actions to Thrust Equations so that the agent is allowed to rotate the drone more aggressively. The thrusts can be derived from the actions using the following formulas:

- $T_l = action_0 \cdot 0.04 + action_1 \cdot 0.003 + 0.04$
- $T_r = action_0 \cdot 0.04 - action_1 \cdot 0.003 + 0.04$

Conversion of Actions to Thrusts Equation 2

Training was done on 5M steps and was tracked using Weights and Biases.

The intuition developed in Run 2 was correct as shown by the Experiment Tracking plots. Allowing the drone to rotate more aggressively increases the performance of SAC.

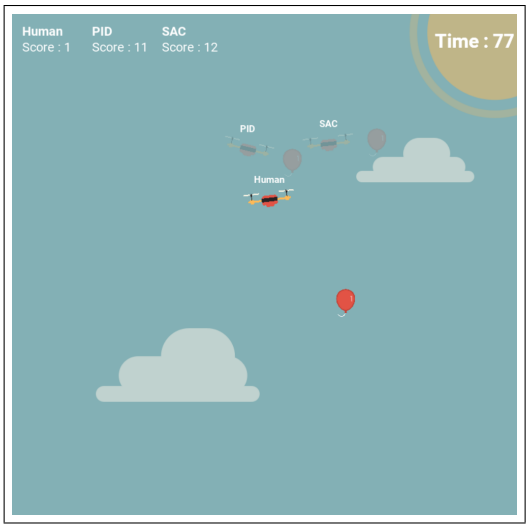

Figure 6. Human, PID and SAC agents competing in BalloonEnv

Agent	Score
Human	51
PID	55
SAC	66

Table 1. Scores in BalloonEnv

Its performance now converges at 1200 of reward which translates to around 12 balloons collected every 20 seconds.

2.8. Final Results

We now come back to the test environment of BalloonEnv where agents are only evaluated on their ability to collect as many balloons as possible within a 100 seconds time limit.

As the table shows, SAC beats PID which beats Human.

- Human is prone to mistakes and overcorrecting, the human agent can be agressive but is more prone to overshoot a target and not be ready for the next one.
- PID is very stable, it chases targets very defensively. This guarantees no crashing and no overshooting but comes at the cost of speed.
- SAC makes no mistake, it adapts to every situation and reacts to new targets with a lot of speed without overshooting therefore making it the better agent.

2.9. Conclusion

Reinforcement Learning is capable of learning the complex and unstable environment of drone controlling to navigate waypoints. This method of control outperformed PID control and Human control which are currently the main ways in which real drones are controlled. But we must remind ourselves that this performance comes with a cost that is easy to pay in a simulated environment but much harder to pay in the real world: training. To achieve this performance, SAC had to train on 5 million decisions. Getting this amount of training using real drones would be very hard even when not considering imperfect information from sensors. On the other hand, PID controls are easy to implement, they just require a few hours of manual testing to tune the PID controllers and the decisions of PID controllers is fully explainable which is important in critical usecases.

It would have been interesting to analyse the robustness (by adding imperfect information or a wind force) of the different agents as this is an important advantage that control theory offers. Combining the methods by tuning PID parameters in real-time using reinforcement learning could also have been an interesting experiment.

Reinforcement Learning can outperform current methods of drone control but comes at the great cost of training and explainability.

References

[1] Ankit Goel, Abdulazeez Mohammed Salim, Ahmad Ansari, Sai Ravela, and Dennis Bernstein. Adaptive digital pid control of a quadcopter with unknown dynamics, 2020. 1

[2] 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. 2016. 1

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. 1

[4] Dingqi Zhang, Antonio Loquercio, Xiangyu Wu, Ashish Kumar, Jitendra Malik, and Mark W. Mueller. A zero-shot adaptive quadcopter controller, 2022. 1