Exploring the use of Reinforcement Learning for Flight

Theo McArn

CSCI 1420: Machine Learning

1 Abstract

The purpose of this project was to gain an introduction to Reinforcement Learning(RL) by training a virtual quadcopter to learn optimal flight controls. After experimenting with many different models, with various training environments and reward mechanisms, my best result is an RL model which fine-tunes PID parameters in order to control the flight of the drone. This controller slightly outperforms a manually tuned PID controller. The results was not as successful as I had hoped but this experience has given me a lot of ideas for ways in which this model could be improved. Additionally this project has given me ideas for future RL projects to take on.

2 Motivation

Last semester I took CSCI1951B: Introduction to Robotics which was an influential class for me. In this class we assembled drones and programmed them to be autonomous utilizing state estimation, and PID controls. I wanted to take what I have learned from Machine Learning and apply these skills to Robotics in order to make a drone with more sophisticated controls and adaptability. What makes learned controls more desirable as compared to PID controls is their ability to adapt to unknown environments and events. In other words the control policy is dynamic. However, with PID controllers, the control policy is set manually on initialization and is used in the same way independent of the environment. However, PID controllers do have benefits such as their explainability and physics-based policy. As a result, I wanted to try and use Reinforcement Learning to optimize the PID controller parameters in attempts to get the best of both approaches.

3 Related Work

There is currently a lot of work being done on using Reinforcement Learning to control drone flight. It has been proven to be a powerful and adaptive form of control. One popular use of RL is to counteract extreme unexpected winds the drone might experience in the air. It is also commonly used in path planning to find the most optimal path and follow it. Researchers use complex simulations which model drones with environmental factors such as drag, wind, and obstacles among others. They then train these drones in simulation using RL in order to optimize control and adaptivity. Then, since the simulations are modeled so closely to real life, these learned controls can then be transferred to actual drones with very little finetuning necessary. This method is used to develop controllers that allow drones to be more agile, and overall more stable.

While this research is very interesting, the complexities that go into simulating these environments and training these controls is very advanced and far beyond my abilities as someone set out to understand Reinforcement learning.

However, I came across a paper by Alexandre Sajus (Sajus 2022) which was the main inspiration for this project. Sajus reduced the dimensionality of the simulation by modeling a two dimensional

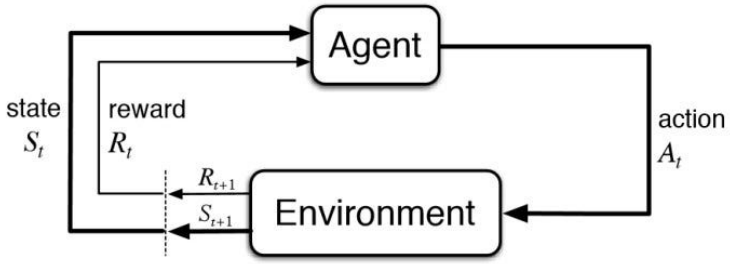

Figure 1: Diagram of RL training (Sharma 2020)

drone rather than a 3D one. By creating a 2D drone, this reduces the degrees of freedom for the drone which as a result, makes modeling and training the controls much easier. Now the drone has only two actuators to adjust, the left and right motors, and only 3 degrees of freedom, x translation, y translation, and roll angle. In comparison, a traditional drone has 4 actuators and 6 degrees of freedom. By reducing the dimensionality in this way, it simplifies the problem extremely and makes reinforcement learning a much more reasonable and achievable goal for someone with no prior experience in this field.

4 Background and Setup

In RL, the basic concept is that there is an agent, our drone, and it interacts with an environment. At each time step, the agent makes observations about the environment and acts in accordance of these observations to change its own state as well as the state of the environment. In our case, the drone observes the location of the target, and acts on this observation by changing its thrust to move towards the target. After moving, a reward is calculated, i.e. if the drone moves in the correct direction, the reward is high and if it moves in the wrong direction, the reward is low. The RL model can the understand that action x produced observation y and reward z. Through many iterations, or episodes, of training, the subject steps through the environment and the RL model attempts to optimize the subjects actions in order to maximize the rewards. The idea of the general training loop can be seen in Figure 1.

Since the RL models are only trying to maximize the reward, engineering when rewards should be given and how much should rewarded dictates training and the final behavior as well. Additionally, the way that the agent interacts with the environment will play an equally large role in how the agent performs after training. Therefore, these parameters such as the reward mechanisms and environment dynamics serve as additional hyperparameters for the RL models and dictate the problem being solved.

Because the environment plays such a large role in defining the objective and the training process, I wanted to define my agent and environment based on Sajus's configuration as it is well defined for this task. By using this predefined environment as the starting point, it allowed me to spend more of my time experimenting with models and tweaking the reward mechanism instead of designing a new environment from scratch. I did however refactor the environment into a more structured and object oriented code-base so that it would allow for more experimentation and modularity.

Additionally, since this project was my introduction to Reinforcement Learning, I utilized a li-

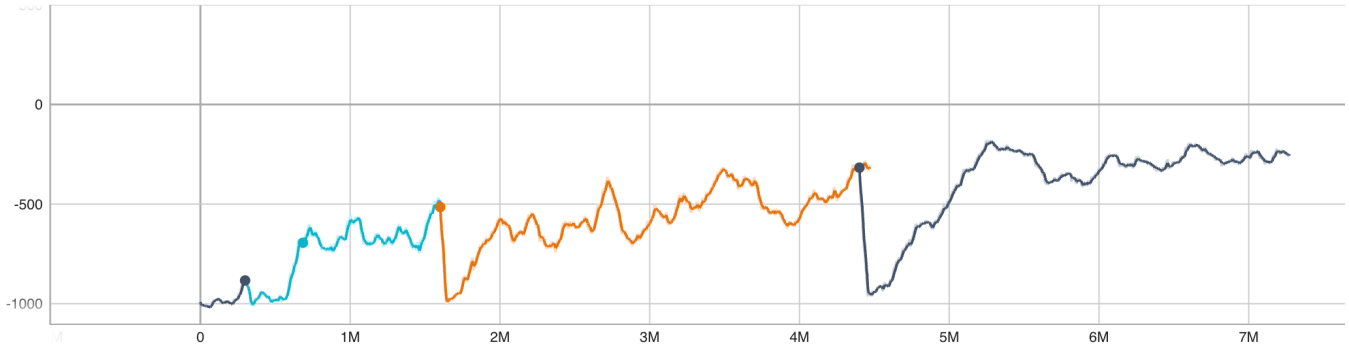

Figure 2: Plot of Initial Attempt at SAC training

brary called **stable_baselines3** which initializes, trains, saves and logs models with very simple API calls. Using this library was a great choice as it allowed me to focus on training and experimentation rather than implementing the specific algorithms.

5 Approach and Iterations

5.1 First Steps

My first steps for this project was to create the basic environment and allow the drone to respond to this environment in a natural way. Using rigid body physics equations, I created a function that translates thrust into acceleration, and then acceleration can be integrated to determine the velocity and integrated again to determine the change in position of the drone. This defines a clear way in which the actions, thrust, influence the environment, location of the agent.

5.2 PID Controller

After modeling the physics of the drone, I then created a basic PID controller to act as a baseline for the RL controllers. The goal of the project was to outperform this controller. I also ran this PID controller in the RL environment to get a sense of the score that it would receive. While the score is non-deterministic due to randomness of the targets and drone positions, the PID controller averaged a score of about 250 points. This would be the score to beat for the other models.

5.3 SAC 1.0

For my first attempts at Reinforcement Learning, I used **stable_baselines3** to train a Soft Actor-Critic (SAC) model. This is one of the current state of the art models know for smooth convergence and importantly, it produces continuous action predictions which allows for continuous thrust control. Despite success on the PID controller, training this SAC model proved far less successful. Training was extremely slow and results were subpar. Initially, I was training this model locally but in order to speed up the process, I continued training remotely on Brown University's High-Performance Computing Machine called Oscar. In all, I trained this initial model for over 7M iterations for a total of 28.4 hours. The training can be seen in Figure 2.

However, even after all of these episodes, the performance was very subpar. At best, the model produced rewards around -200 points which, when visualizes, shows that the drone managed to hover

for sometime before quickly losing control. Its possible that more training would have improved this issue but it seemed that the rewards had been flattening out. One observation I found odd was that when restarting training from a saved location, the reward always seems to steeply plummet before going up again. I could not figure out why this was happening but it drastically slowed down the process of training.

5.4 SAC 2.0

After these initial unpromising results, I made many attempts to improve training by modifying reward mechanisms, altering the number of steps taken with the same action, and changing the agent observation.

Initially, for each time step, the agent was rewarded for staying alive, and reaching a target, and penalized for being far from a target, or crashing. However in hopes of making the actions of the drone smoother, I implemented further penalty if there was a large difference between the previous and the current thrust of the model. I had hoped that these added constraints would make the model more realistic but also more fluid. In addition to changing the rewards, I also changed the observations. Initially, the SAC model took in 6 observations to make an action prediction:

1. Angle of roll
2. Magnitude of velocity
3. Angle of velocity
4. Euclidean distance to target
5. Angle to target
6. Difference between the angle of velocity and the angle to the target

While these are very useful observations, they lacks any direct information on acceleration and angular velocity. I felt that these were extremely important observations and would help the SAC model to make more informed predictions for optimal thrust. Therefore I added 3 more observations:

1. Magnitude of acceleration
2. Angle of acceleration
3. Angular velocity of roll

Additionally because I now penalize the agent for inconsistent thrust, I also added the previous left and right thrusts to the set of observations. This way, by knowing the previous thrust, the model can know how to make more incremental changes and therefore receive better rewards. In total, this brings the set of observations from 6 to 11. However, despite these changes, the training seemed to be even worse than before. I trained this model for 1.5M iterations over the course of 7 hours but rewards quickly leveled out at around -1020. The training can be seen in Figure 3.

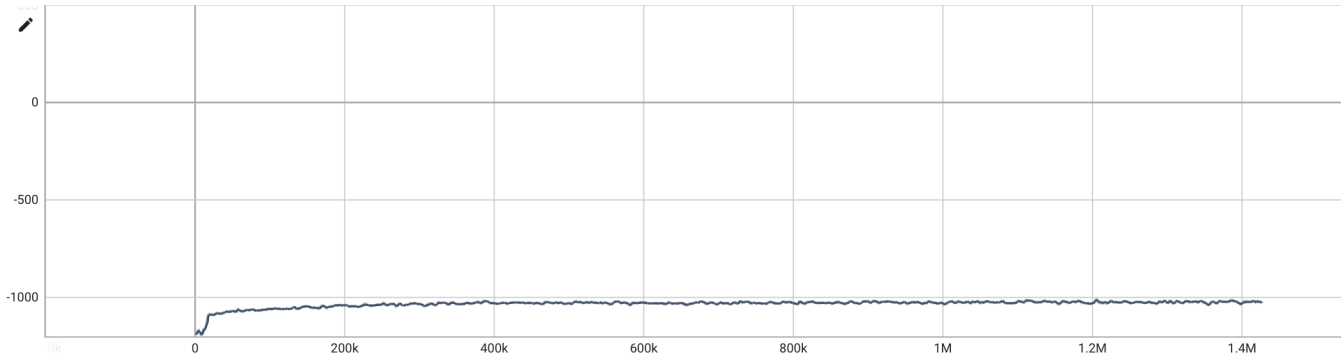

Figure 3: Plot of Second Attempt at SAC training

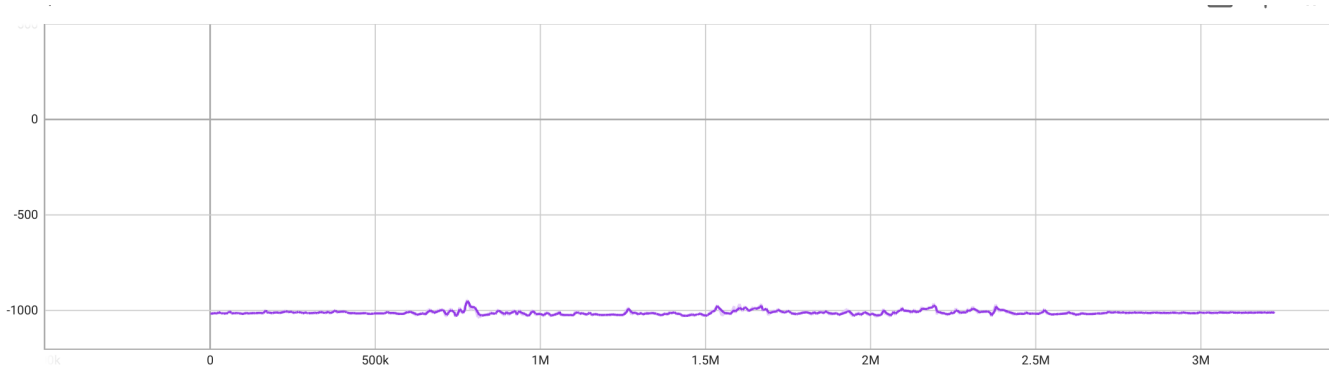

Figure 4: Plot of Attempt at DQN training

5.5 DQN

With all of these disappointments, a attempted to switch directions and adapted this environment to run on a Deep Q Learning agent (DQN). DQN is know as one of the first RL models, and works based on a discrete action space rather than a continuous one. It is a much simpler algorithm so I hoped that this would make training perform better. The results were far from an improvement. I trained for over 3M steps and the reward stayed pretty much constant throughout. The training can be seen in Figure 4.

5.6 PID-SAC 1.0

After these sub-optimal results of using RL directly for the thrust output, I decided to switch gears and focus on a different type of model. I figured that since the initial PID model worked so well on the defined task, I would try and use the same SAC algorithm but instead of predicting thrust actions, I would have it predict PID parameters. I reverted back to the initial reward mechanism and observations for this version. The way I set up this environment, I would initialize the drone with the default PID controller and then tune it once at the beginning based on the initial observation. Then I would continuously update the drone with this same PID controller and it would hit multiple targets, until it crashed or passed the time limit. Then the next step would occur with a new tuning. Training this model actually worked surprisingly well given that the tunings are only updated once per run. After training for over 16 hours, the model seemed to level off at a reward around 300. This just barely out performed the reward score of the baseline PID controller. The training can be seen in Figure 5.

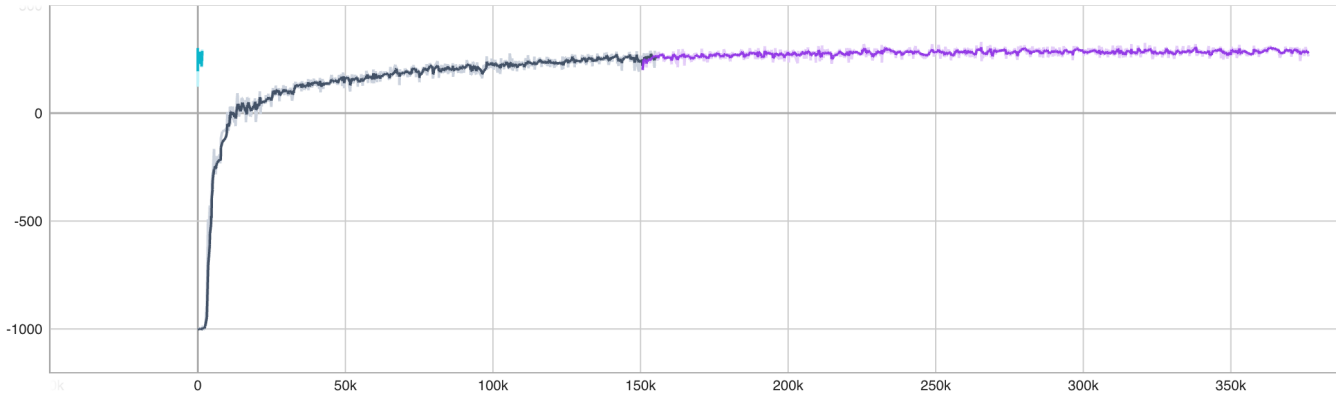

Figure 5: Plot of Initial PID-SAC training attempt

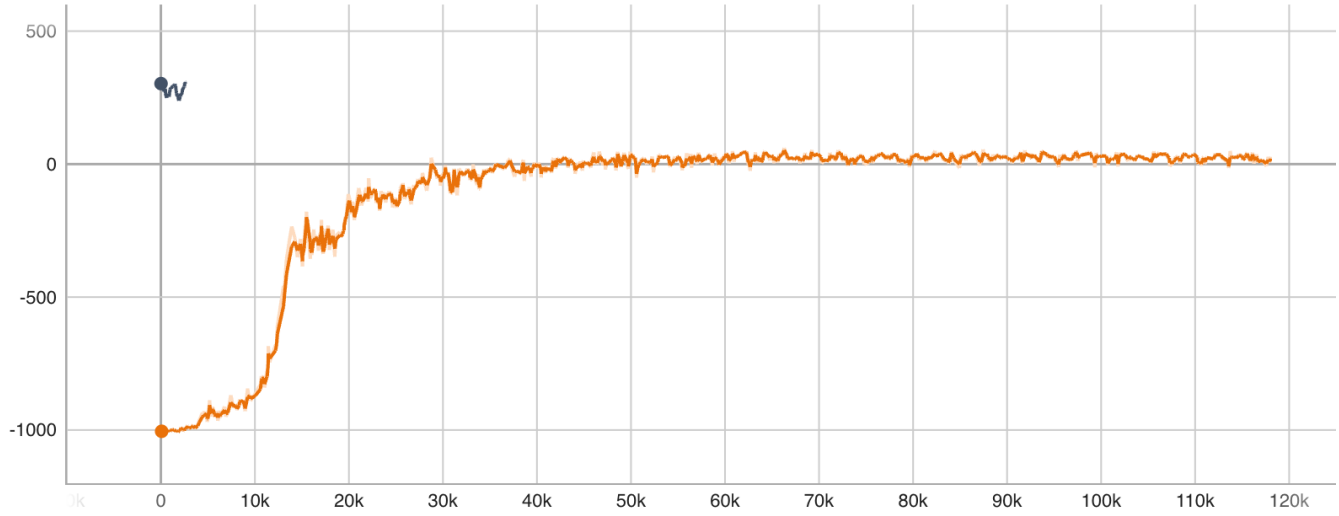

Figure 6: Plot of Second PID-SAC training attempt

5.7 PID-SAC 2.0

The results of PID 1.0 seemed promising and left a lot to be improved on. Since the PID controller is only being modified at the start of every run, the model only takes into account the observations with respect to the first target. However, the way the environment is set up, the agent is using the same controller for each target regardless of where the next target is relative to the drone. Therefore, I tweaked the environment to run a new prediction step every time a new target is assigned. This way, each PID tuning can be perfectly fit to optimize the path to a single target, rather than a tuning that accommodates an average of all of the targets. While I was expecting the results from this training to be a large improvement, the actual results were far from ideal. I trained this model for 4.5 hours but it quickly leveled off with a reward around only 30 points, far lower than the previous model. The training can be seen in Figure 6.

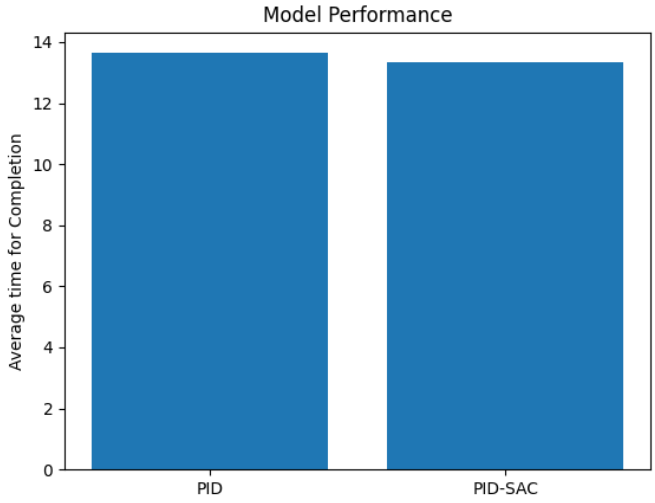

Figure 7: Average Time in Seconds to Reach 5 Targets (40 trials)

6 Final Results

After all of these iterations, I ended up with 2 viable controllers. The benchmark manual PID controller, and the PIDSAC 1.0 . Running both of these together, they compete fairly evenly but the PIDSAC 1.0 drone seemed to perform slightly better often getting to the targets just before the normal PID drone. In order to test this out empirically, I timed how long it took each drone to reach 5 targets and repeated this experiment 40 times. After averaging the results, it seems that the PIDSAC 1.0 version is slightly faster at reaching the targets. The graph can be seen in Figure 7. However, while PIDSAC 1.0 is slightly faster, it is far less consistent. There are many times where the PID controller fails unexpectedly.

If you would like to run the visualization yourself and see the different versions of the drones compete, the repository as well as instructions to run the demo can be found at "https://github.com/tmcarn/cs1420 .

7 Improvements and Future Work

Its clear from the results that there is not much improvement from the manual PID to the learned PID, especially given all of the time needed to train, but I know that there exists a more optimal solution to this problem that would have much larger performance results. If I were to continue this project, I would want to attempt to train another RL model similar to SAC 2.0 where the action space is thrust, rather than PID parameters. I think that a model like this has a lot of potential adaptive capabilities that you cannot achieve with a normal or RL-tuned PID controller. I know that getting success with a model like this is achievable, I suspect there is some issue with my environment that I have not been able to catch. Additionally, I think adjusting the hyperparameters of the SAC model might also improve training.

In addition to getting my normal SAC drone to fly better, I think it would be cool in incorporate objects such as obstacles or disturbances such as wind into the environment. This way, the drone could be trained to be even more adaptive and stable in its environment.

For future work apart from this project, I would love to be able to implement and train a RL

model to control a 3 dimensional drone. While this is a significantly more challenging task, this current project has been a good introduction to put me on the path to this larger goal. I still have the drone that I build as part of CSCI 1951R and I would love to be able to create a simulation based on the physics of this drone, train a controller, and then transfer this controller to the physical drone.

This project has taught me a lot about Reinforcement Learning and I am really excited to take this knowledge further and expand on it to make some really cool robots.

References

Sajus, Alexandre (2022). "Reinforcement Learning for the Control of Quadcopters". In: *GitHub*. URL: https://github.com/AlexandreSajus/Quadcopter-AI.

Sharma, Siddharth (2020). "The Ultimate Beginner's Guide to Reinforcement Learning". In: *Towards Datascience*. URL: https://towardsdatascience.com/the-ultimate-beginners-guide-to-reinforcement-learning-588c071af1ec.