A Methodology for Learning Airplane Descent Patterns

Michael Tsai

12/14/22

Ridgefield High School

Abstract

In this study, I trained a plane to land without human assistance. I did this by using reinforcement learning, a type of machine learning where the agent (the AI "pilot" in this case) runs iterations of the task and adjusts its methodology depending on this interaction with its environment. This is called the Markov Decision Process. This is an important problem to solve, as reinforcement learning applications such as this one can be used to improve the safety of travel and everyday life, as well as quality of life for everyone involved. Additionally, the fact that reinforcement learning can be used to solve problems like these implies further applications in complex fields like flight and other modes of transportation. To figure out how to train a plane to land, I used OpenAI Gym to build a simulated environment to test different landing maneuvers and trained the agent to optimize the angle of attack for landing. Initially, I experimented with optimizing the angle of attack for variables like time taken to land and descent rate of the plane, but ultimately found it most effective to optimize for having the lowest velocity when the plane reaches the ground. When optimizing for time taken to land, the plane doesn't follow a smooth descent, causing the plane to simply freefall to the ground, but when optimizing for ending velocity, the plane follows a more normal path. Further research could potentially experiment with how different variables such as speed, acceleration, and angles of the left and right ailerons affect the plane's landing.

1. Introduction

In recent years, Machine Learning technologies have improved drastically, giving the potential for new applications in automation that can improve safety and quality of life for those involved. Since Reinforcement Learning is used to train an AI to accomplish a task using the Markov Decision Process, it is the primary type of Machine Learning used in automation. Current applications of Reinforcement Learning in automation mainly fall in the domain of self-driving cars, with other applications in Atari games. However, Reinforcement Learning has not yet been applied to something like a self-flying plane, which adds the additional element of complexity in the flight dynamics of the plane. If Reinforcement Learning could be trained to cope with these dynamics and learn to properly fly a plane, this could drastically improve the safety of both the pilot and the passengers on the plane. In this study, I simulated a plane's landing, keeping the thrust constant. Then, I trained the model to optimize the plane's angle of attack, keeping track of the horizontal and vertical coordinates of the plane. Finally, I plotted these coordinates along with the long-term rewards for the agent, and compared the flights to one another.

2. Background

Currently, the vast majority of planes use autopilot to fly autonomously. While this method does not use Machine Learning and is therefore an easier algorithm to train, it is less reliable. Even with autopilot, the pilot has to remain attentive and aware of what is going on, in case something fails and he or she needs to take control of the plane. Additionally, autopilot still cannot takeoff or land a plane, as these maneuvers are too complicated for autopilot. This is why

Reinforcement Learning is a better solution. Reinforcement Learning is more adaptable, meaning that it can be trained to accomplish goals in numerous situations such as landing, taking off, or flying over mountains [5]. Previously, Yann Berthelot has created a Reinforcement Learning model that can train a plane to take off without human assistance [1]. This is an interesting solution to this problem, as it leverages Proximal Policy Optimization to train the model [2]. However, from a physics perspective, the more difficult aspect of a plane's flight to accomplish is the landing, since the model cannot simply optimize for landing in the shortest amount of time possible. Instead, it has to account for factors such as horizontal velocity, vertical velocity, and descent rate. I found that the typical cruising speed for an Airbus A30 is about 228 m/s at an altitude of 12,500 meters [7], which were numbers that I took into account when I was training my model.

3. Dataset

In order to simulate a plane's landing, I used a physics based model of an airplane built in OpenAI Gym [1]. Here, the model takes into account factors such as angle of attack, thrust, and velocity to determine the vertical and horizontal positions of the plane. These positions are then stored in a list, where the data is then used for visualization outside of the simulation. The position and velocity are used to compute the plane's value for a given simulation run.

4. Methodology / Models

In order to train the plane to land autonomously, I used reinforcement learning: a type of machine learning typically used to train an agent to accomplish a goal. Reinforcement Learning is

primarily used in atari games, but also has applications in self-driving cars and other disciplines. Reinforcement learning, similarly to many other types of machine learning algorithms, works by a process of trial and error. During training, the agent initially takes arbitrary actions to try to accomplish a goal, and then depending on the reward the agent receives at the end of the episode, the agent adjusts its algorithm to more efficiently solve the problem. Additionally, there are different policies that the algorithm can use which determine the way that the agent interprets the reward. In this case, I used Proximal Policy Optimization, a policy that uses Trust Region Policy Optimization to ensure that the updated policy never deviates very far away from its original policy. This makes it so that the new data being collected by the agent is consistent, allowing the agent's policy to improve. Proximal policy optimization does this by "clipping" its advantage function as shown in the diagram below:

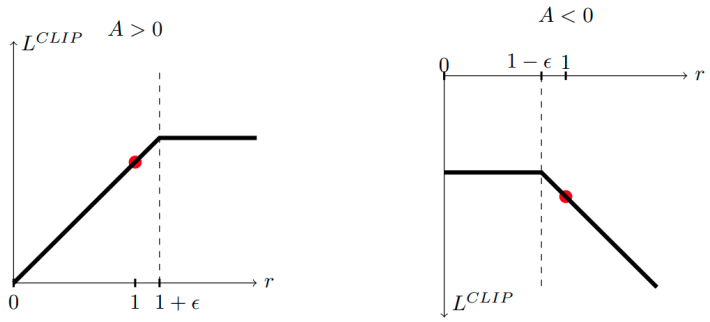


Figure 1: Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., $r = 1$. Note that L^{CLIP} sums many of these terms. [5]

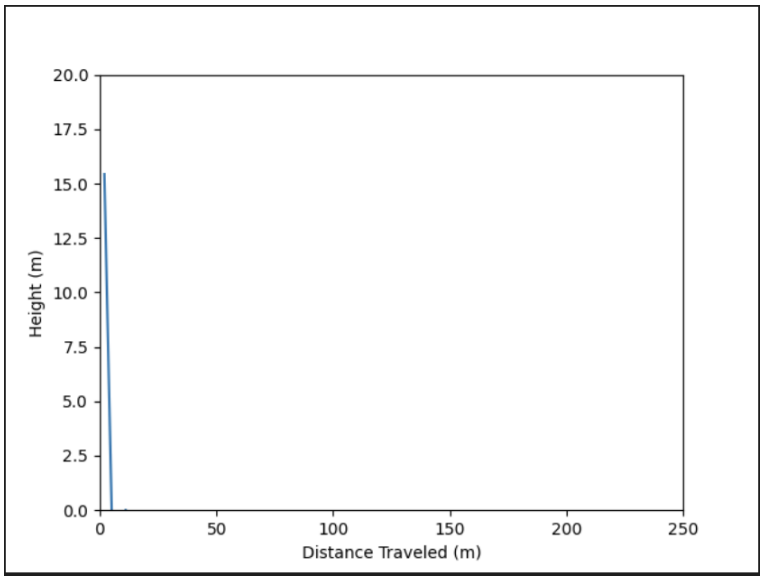
Clipping the advantage function ensures that the agent is conservative if the advantage function moves too far away from the current policy.

I used four main files to train the plane: main.py, utils.py, FlightModel.py, and FlightEnv.py. Main.py was primarily used to initialize the agent and an instance of FlightModel, as well as call gridsearch_tensorforce, which trains the agent over a certain number of episodes.

The reward function and the terminal states are both defined in FlightEnv.py. For each updated reward function, I trained the model over 1500 episodes, keeping track of the position and angle data for each 100 episodes. I then graphed the plane's flight after this training to compare the effects of each reward function on the plane's flight. For my final reward function, I also trained the agent over a longer period of time and plotted the reward it achieved over training.

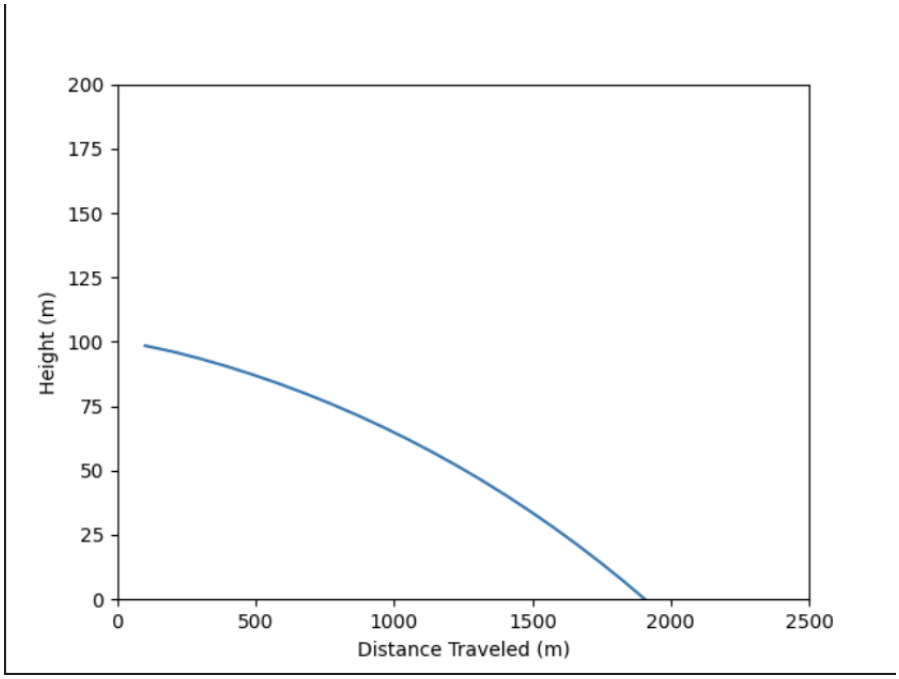
5. Results and Discussion

Figure 1:



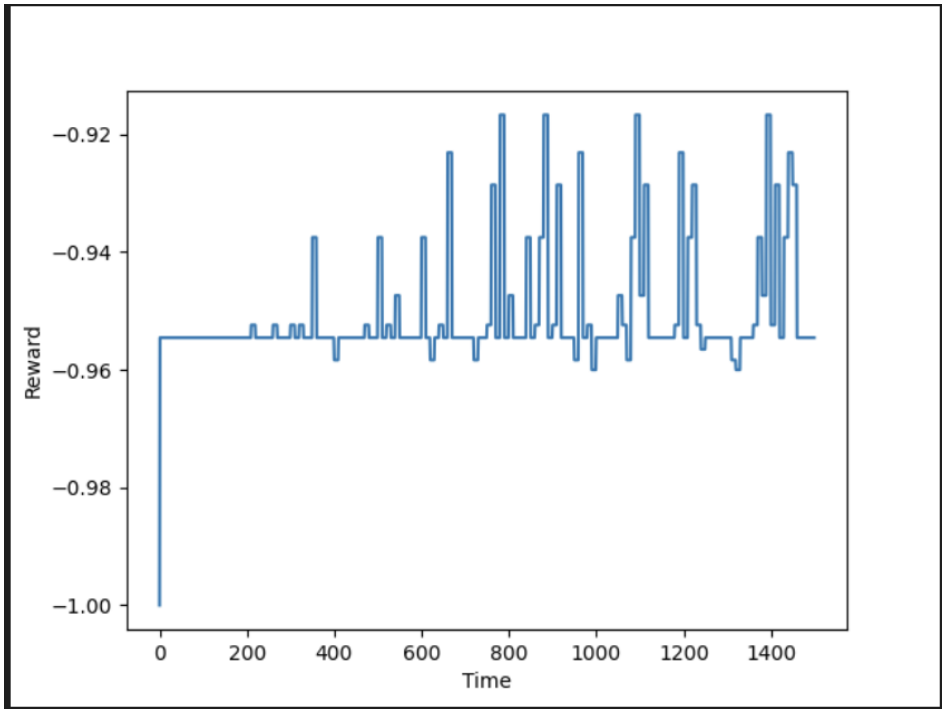
This is an image of the plane's landing under my initial reward function, which minimized the time taken for the plane to land. Note: the scales on the y-axis and x-axis are different than those in other reward functions. However, since the ratio between the x-scale and the y-scale is still the same, this still accurately depicts the slope of the plane.

Figure 2:



This is a plot of the plane's flight, trained over 25000 episodes.

Figure 3:



This is the reward plot of the plane's final reward function.

In this study, I found that in order to properly land a plane, the reward function should reward for a lower velocity of the plane. I experimented with various reward functions. First, I tried rewarding the agent for taking the shortest time to land the plane. However, I found that this caused the plane to simply fall out of the sky, which wouldn't be realistic because of the extreme forces that the passengers would feel in this scenario. In Figure 1, the plane is decreasing its height with a slope of -1.64. This means that for every meter it travels horizontally, its height decreases by 1.64 meters. This would mean that the plane would be decreasing at over a 45 degree angle downward, which is not realistically safe. Because of this abnormal path, I changed

the reward function to increase for taking longer to land until the landing time got to a certain value. If the plane took longer to land than this, I deducted the reward from the plane. This would make it so that ideally, the plane would increase its landing time while not taking an infinite time to land. With this reward function, for whatever reason, the model did not train in a feasible amount of time. This is why I changed the reward function to my current reward function, which optimizes for having a lower velocity at landing. This function seemed to work, as the plane was able to land with the path shown in Figure 2. Additionally, the plane achieved the reward plot shown in Figure 3, meaning that the agent was able to maximize its rewards with this reward function.

I also experimented with different initial values for the speed, acceleration, and position of the plane. The average cruising speed for an Airbus A320 is about 228 m/s at an altitude of 12,500 meters [7]. Since I am landing from an altitude of only 100 meters, I decided to set the initial velocity at 100 m/s. I also decided to set the thrust to be 0. I initially set the plane to be 25 meters high, but then changed it to be 100 meters high, since 25 meters did not give the plane enough time to land given the starting speed.

6. Conclusions

In this study, I used a simulation of a landing plane to train an agent to change its angles of attack to land autonomously starting from 100 m/s horizontally and 100 meters high. I mainly experimented with how to reward the plane to accomplish this goal as effectively as possible. For each modified reward function, I trained the model over 1500 episodes. I then stored the angle and position data in a text file for each episode, and graphed the flight of the plane to visually compare which reward functions landed the plane in the safest manner possible. I experimented

with a few different reward functions: one which rewarded for taking the least time to land, one that rewarded for taking exactly a certain amount of time to land, and one that rewarded for the velocity being the lowest at landing. I also considered rewarding for decreasing height and speed at a fixed rate, but decided against it since I wanted to see if the plane could find a better method to land than humans currently do. In my final experiment, I found that considering horizontal velocity yielded a behavior that is more realistic than other trials.

References

1. Berthelot, Yann AI learns to fly (Part 1) | Airplane simulation and Reinforcement Learning | by Yann Berthelot | Towards Data Science, AI learns to fly (Part 2) | Create your custom RL environment and train an agent | Towards Data Science
2. Schulman, et al. https://arxiv.org/pdf/1707.06347.pdf
3. Recht, Benjamin https://arxiv.org/pdf/1806.09460.pdf
4. Henderson, et al. https://www.nowpublishers.com/article/Details/MAL-071
5. Wang, et al. http://proceedings.mlr.press/v139/wang21ae/wang21ae.pdf
6. Airbus https://aircraft.airbus.com/en/aircraft/a320/a320ceo
7. DiLisi, Gregory A. https://aapt.scitation.org/doi/abs/10.1119/1.2731279