Reinforcement Learning Based Autopilot

Muhammad Rizwan Malik
rizwanm@usc.edu

Muhammad Oneeb Ul Haq Khan
mkhan250@usc.edu

Martin Huang
hhuang04@usc.edu

Krishnateja Gunda
kgunda@usc.edu

Rengapriya Aravindan
raravind@usc.edu

Abstract

State-of-the-art autopilot systems rely on control system theory, an approach which becomes extremely complex as demands from it increase. Our goal for this project is to develop a Reinforcement Learning (RL) agent, capable of learning basic flight maneuvers within a simulated environment (X-Plane 11). Having learnt basic flight maneuvers, we propose that our agent would be able to follow a provided flight plan. For our objective, we use several RL algorithms such as REINFORCE, Proximal Proximity Optimization and Deep Deterministic Policy Gradient.

I Introduction

With the recent advances in the field of Reinforcement Learning and Deep Learning there has been an increased interest in solving problems which were considered intractable in the past. One of the common themes in the current research has been to develop models which can learn and perform tasks with close-to-human performance. The problem that is discussed in this paper relates to development of a intelligent autopilot system based on reinforcement learning.

Human pilots are trained to handle flight uncertainties or emergency situations such as severe weather conditions or system failure. In contrast, Automatic Flight Control Systems are highly limited in a way that they are only capable of performing minimal piloting tasks in non-emergency conditions. Strong turbulence, for example, can cause the autopilot to disengage or even attempt an undesired action which could jeopardise flight safety. Moreover, the current regulatory requirements by International Civil Aviation Organization (ICAO) require constant monitoring of the system and the flight status by the flight crew to react quickly to any undesired situation or emergencies. On the other hand trying making an attempt to cater for every possible uncertainty in the flight conditions by hard-coding it into the autopilot is not only impossible but futile as well.

This work aims to address these problems by proposing a reinforcement learning based Intelligent Autopilot System(IAS) capable of learning to perform different flight maneuvers just by experience. The proposed model will be trained and tested in X-Plane 11 environment. Instead of having the agent learn by imitation as done in the past [1] this agent learns by performing actions by following a policy and then improving the policy based on the rewards it receives from the environment.

This paper is structured as follows: section II covers the relevant prior research, section III deals with the specification and overview of the environment used for training the agent, section IV concerns discussion of the algorithms which were implemented and V presents results and analysis.

II Related Work

Past research in the field of autopilot system has been focused primarily on control system theory. Classic and modern autopilots are based mostly on Proportional Integrated Derivative (PID) controllers or Finite-State automation. However, very limited research has been done in exploring self-learning or experiential learning autopilots. In the past, one of the limiting factors has been computational intractability of the close to infinite state-space of the real life flight dynamics model and another has been limited research into efficient algorithms to handle large and continuous state spaces. Some efforts were made towards development of IAS based on imitation learning [1], where a training dataset was first collected of a human pilot flying an aircraft on a simula-

tor. This dataset was then used to train an Artificial Neural Network (ANN) based model. This report [**Stanford˙University˙Final˙Report**] acts as a proof of concept that experiential learning can be used to train neural networks to control an aircraft.

III Data and Environments

A) Overview of X-Plane 11

For our RL Agents' environment, we will be using X-Plane flight simulator software.There are many other flight simulator softwares but we will be using X-Plane because X-Plane has the capability to read and write data from the simulator.Also unlike other flight simulator softwares where calculating aerodynamic forces such as lift and drag happens by using empirical data in pre-defined lookup tables, XP11 solves aerodynamic equations in real time.This allows XP11 to keep the simulation as close to reality as possible.

Figure 1: X-Plane 11 Flight Simulator used for this project

B) X-Plane Connect (XPC)

In order for our Python script to communicate with XP11, we are using an existing plugin, the NASA X-Plane Connect plugin which provides different function calls to send/read current control information to and from the simulator without the use of UDP sockets. The XPC Toolbox is an open source research tool used to interact with XP11. XPC allows users to control aircraft and receive state information from aircraft simulated in X-Plane using functions written in C, C++, Java, MATLAB, or Python in real time over the network.

IV Methods

The solution to any game based RL training problem can be divided into following parts:

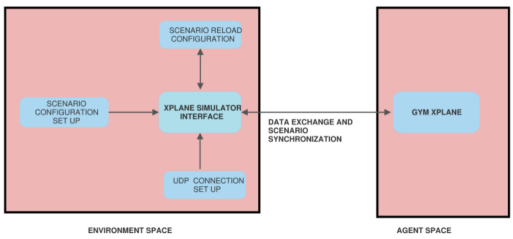

Figure 2: Agent-Environment InteractioFlow.

1. The first step is to configure the environment using which the agent will be trained. In our case this includes setting up the communication link between the Python script and the simulator. All the parameters will be passed through this link, i.e. XPC.

2. One of the most important tasks in solving a problem using RL is defining the reward function. It becomes even more important when dealing with a close-to-reality and complicated environment like flight simulation. Reward function is the only way an agent knows what is it supposed to learn and also plays a key role in determining how quickly it learns it.

3. Artificial neural networks are good function approximators whenever trying to learn complicated non-linear models.

4. Training the network based on policy gradient approach. Our network will basically output different actions that can be taken for a given observation and the reward function will evaluate whether that particular action in the given state took us closer to the target or not and that will be utilized to learn a policy for reward maximization.

5. Once the network is trained, it will be tested and results will be compared against the current data.

A) Simulation Environment

Reinforcement Learning problems require the agent to sense the environment, choose an action after evaluating the current policy for the sensed state, perform that action and sense the new state of the environment. Open AI provides an abstraction layer to perform these tasks in the form of Open AI Gym framework for several games and

other simplified environments. However, Gym does not have any environments for X-Plane 11 so we interfaced our own environment following the Gym API guidelines.

i. Rewards and Penalties

The following reward function was implemented for the RL agent to be able to learn to maintain the target altitude:

- reward =

  $$-\sqrt{|current\_altitude - target\_altitude|}$$

  for every step when the aircraft is not in the target zone, which is defined as: $(target\_altitude \pm 100)$

- reward = $+6000$ for being inside the target zone

- reward = -100000 in case the aircraft crashes

- reward = -20000 if the episode and ends and the aircraft was not within the target zone

B) Reinforcement Learning Agents

The project team has currently implemented 3 RL algorithms, in order to ascertain the better performing algorithm for the given use-case.

i. REINFORCE

REINFORCE is a Monte-Carlo variant of policy gradients (Monte-Carlo: taking random samples). The agent collects a trajectory of one episode using its current policy, and uses it to update the policy parameter. Since one full trajectory must be completed to construct a sample space, REINFORCE is updated in an off-policy way.

The loss function for REINFORCE, a vanilla Policy Gradient implementation, is given as:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[log\pi_\theta(a_t|s_t)\hat{A}_t]$$

In the above equation $L^{PG}(\theta)$ is the policy loss and it is equal to the expected value of taking action a_t in state s_t. The expected value is weighted by the estimated advantage function $\hat{A}_t$, which in the case of REINFORCE is simply the cumulative discounted rewards.

ii. Deep Deterministic Policy Gradient (DDPG)

DDPG is a reinforcement learning technique that combines both Q-learning and Policy gradients. DDPG is an Actor-Critic method. Actor-Critic methods are better suited for problems dealing with continuous action spaces. In Actor-Critic methods there are two policy networks being trained simultaneously. One network learns the actual policy to behave optimally given a state, this network is the Actor. The second network learns the value function of the underlying MDP, this network is called Critic because it criticizes how the Actor network evaluates the rewards of a state while updating its parameters.

DDPG is an "off"-policy method. DDPG is used in the continuous action setting and the "deterministic" in DDPG refers to the fact that the actor computes the action directly instead of a probability distribution over actions.

iii. Proximal Policy Optimization

Proximal Policy Optimization like DDPG also belongs to the class of Actor-Critic methods. To understand the main difference between vanilla REINFORCE algorithm and PPO, we need to look into the optimization objectives of both of these methods.

In the paper Trust Region Policy Optimization (Schulman et al, 2015) the authors introduced a concept of *Trust Regions* to limit the policy gradient step so it does not move too far from the original policy, preventing overly large updates that often ruin the policy altogether.
For this, they define $r(\theta)$ as the probability ratio between the action under the current policy and the action under the previous policy.

$$r_t\theta = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

Given a sequence of sampled actions and states, $r(\theta)$ will be greater than 1 if the particular action is more probable for the current policy than it is for the old policy. It will be between 0 and 1 when the action is less probable for our current policy. Since our action space is continuous and we sample the actions from 4 uncorrelated normal distributions. Therefore, instead of directly dividing the probabilities of actions, we take exponentials of probability density functions.

The loss function is defined using the probability ratio as follows:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t]$$

Here, the expectation is being computed over a minimum of two terms: normal policy gradient objective and clipped policy gradient objective. The second term plays a key role where the objective value is clamped between $1-\epsilon$ and $1+\epsilon$, ϵ being the hyperparameter, which in this paper is set to 0.2.
Furthermore, because of the min operation, this objective behaves differently when the advantage estimate is positive or negative.
One aspect of PPO which makes it more suitable for our problem is its sample efficiency. It makes use of a memory replay buffer to store the sample actions and then trains the model for several epochs over that data before discarding it, unlike REINFORCE algorithm where an experience trajectory is used only once to train.

V Results and Analysis

A) REINFORCE

Discussed here is the most promising run of our REINFORCE algorithm. Initially, the agent crashes the plane consistently, as is observed in Figure 3, however it eventually learns to not crash and then eventually also starts trying to come back to 2500m after it has descended past it, as observed in Figure 4. However, instead of improving upon this, we see in Figure 5 that the agent continues to descend past the target altitude. This trend continues for several hundred more episodes, after which the training for this run was terminated.

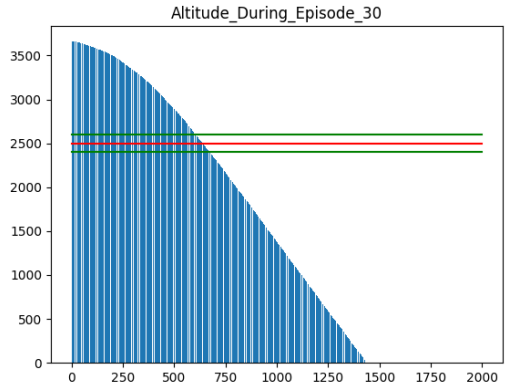

Figure 3: Episode 30 of REINFORCE

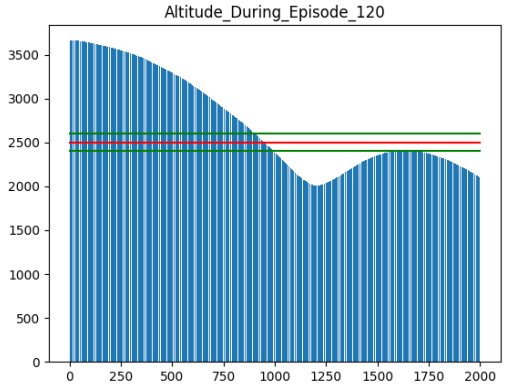

Figure 4: Episode 120 of REINFORCE

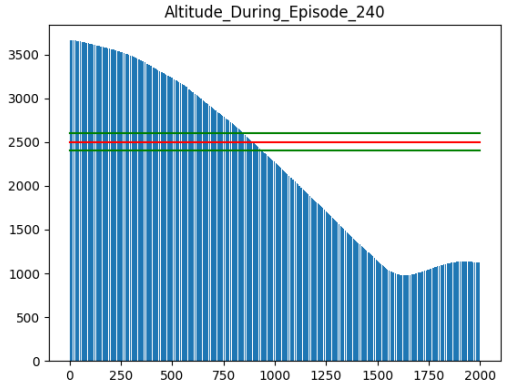

Figure 5: Episode 240 of REINFORCE

B) DDPG

In Figures 6, 7 and 8, it is observed that the agent learns to not crash the plan and that there is less penalty the closer the agent flies towards its target of 2500m. However, even so after 998 episodes the training the agent is unable to learn to fly at the target altitude or within in the tolerance range of 100m. More debugging and analysis is required here to figure out the necessary modifications required to the existing code for DDPG to ensure successful training.

C) PPO

Results for PPO algorithm have been inconclusive in our case. Average scores for the last 100 episodes did not show a lot of variation and the descent trajectory that the aircraft followed did not change either over the course of 500 episodes.

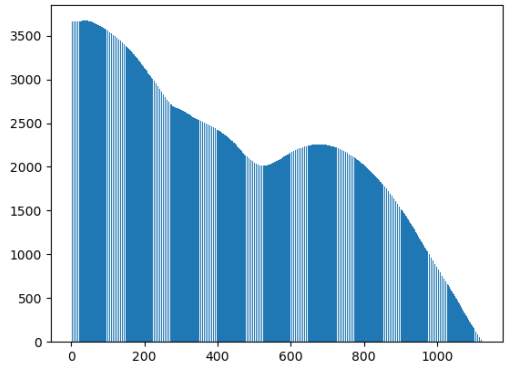

Figure 6: Episode 121 of DDPG

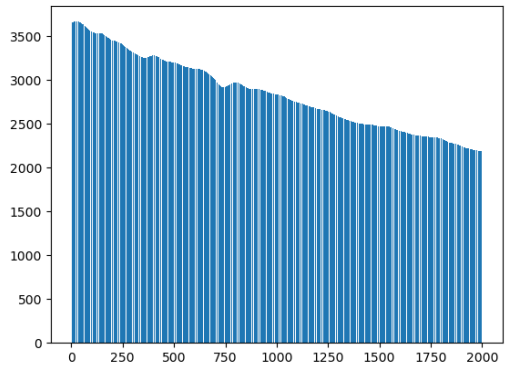

Figure 7: Episode 232 of DDPG

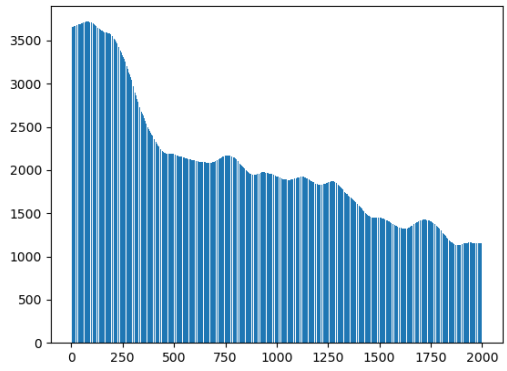

Figure 8: Episode 998 of DDPG

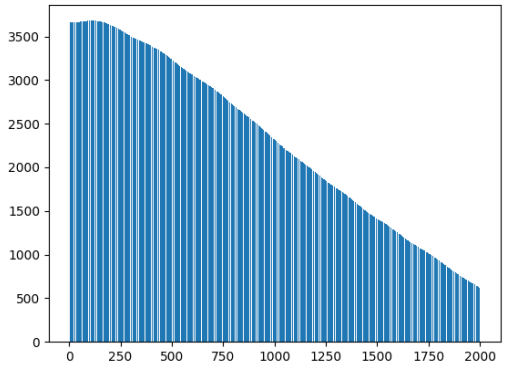

Figure 9: Episode 10 of PPO

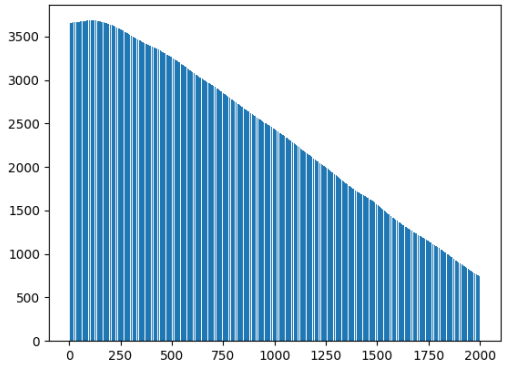

Figure 10: Episode 490 of PPO

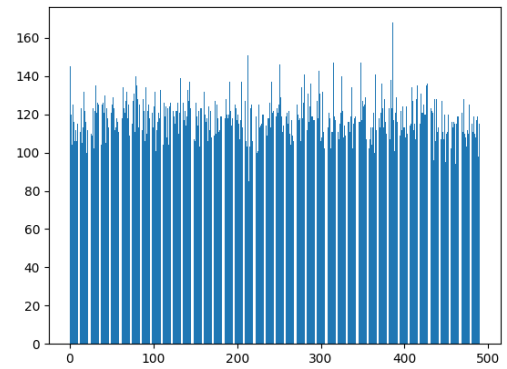

Figure 11: Number of Successful steps in each episode

VI Limitations, Conclusions and Future Work

The results from all 3 RL agents have been unsatisfactory so far. At the time of submission of this report, the project team understands why the results have not proven to be successful as yet. REINFORCE has proven to be an extremely inefficient approach, which requires training for several thousand episodes, since each episode takes over a minute to run, this quickly becomes a non-viable approach. Therefore, moving forward, we

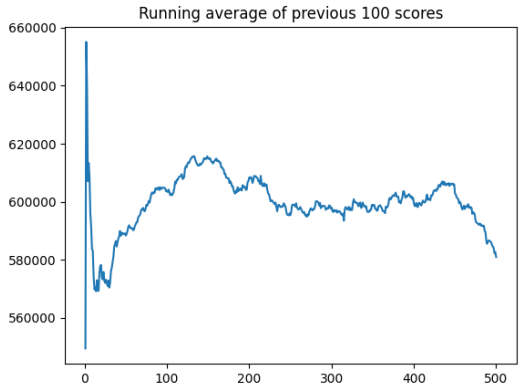

Figure 12: Avg score of last 100 episodes

will be focusing heavily on PPO and DDPG implementations, making the necessary adjustments to ensure that target objectives are being met. Currently, the goal has been to maintain altitude, once this is achieved, we will expand the goal objective to include maintaining heading as required as well. Once this objective is achieved, we will shift our focus so that our agent learns to achieve any target altitude and heading. This will allow us to piece together entire flight plans, which the agent should be able to follow.

References

[1] Haitham Baomar and Peter J Bentley. "An Intelligent Autopilot System that learns piloting skills from human pilots by imitation". In: *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2016, pp. 1023–1031.

[2] Jean de Becdelievre et al. "Autonomous Aerobatic Airplane Control with Reinforcement Learning". In: (2016).

[3] Yenn Berthelot. *AI learns to fly — Airplane simulation and Reinforcement Learning*. [Online; accessed 26-April-2020]. 2020.

[4] Yenn Berthelot. *AI learns to fly — Create your custom Reinforcement Learning environment and train your agent*. [Online; accessed 25-August-2020]. 2020.

[5] *Deep Deterministic Policy Gradient*. `https://spinningup.openai.com/en/latest/algorithms/ddpg.html`.

[6] William Good. *FlyWithLua for X-Plane 11*. 2020.

[7] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3 (1992), pp. 229–256.

[8] *X-Plane Connect*. `https://github.com/nasa/XPlaneConnect`.

[9] *X-Plane Datarefs*. `https://developer.x-plane.com/datarefs/`.

[10] Takeshi Tsuchiya Yuji Shimizu. "Construction of Deep Reinforcement Learning Environment for Aircraft using X-Plane". In: *Machine learning* 8.3 (2020), pp. 112–119.