Autonomous Plane Flight Using Deep Reinforcement Learning

Shaurya V. Raswan
shauryaraswan@gmail.com
Northwood High School
Irvine, CA 92620

1. Abstract

In this paper, we assess the capability of using Deep Reinforcement Learning (Deep RL) for autonomously flying a simulated plane. Using an Advantage Actor-Critic model with a custom policy, the reinforcement learning agent attempts to maintain a constant heading and altitude for a plane in mid-flight. The machine learning agent is composed of actor and critic neural networks that would need to achieve safe autonomous flight in X-Plane [8], a realistic flight simulator. The agent interacts with X-Plane by exploring and training from the simulation, using X-Plane plugins and a comprehensive gym environment built on OpenAI's gym standards. The X-Plane gym environment is used to observe the simulator, control the aircraft, and calculate rewards in real time. The agent communicates with the gym environment to determine the best action in the 4 continuous action space parameters, using the 11 continuous state space observable parameters to train. The actor predicts the best action to take based on the critic's return value function. The critic predicts how well the action did by using a neural network, while the actor updates the policy gradient based on the critic. The probability gradients for continuous actions are multivariate normal distributions, meaning the actor has 2 neural network models for each action. One model is for the standard deviation while the other is for the mean, both used for choosing the best action. The actor and critic update their neural networks at each timestep, trying to minimize their loss by inputting the states and rewards the gym environment sends. By analyzing the results, the pilotless plane can fly when using an optimized A2C policy after sufficient timesteps, allowing the plane to keep heading and altitude with few instabilities. Convolutional neural network support would further improve the agent, supporting plane functions like taxiing, takeoff, landing, and air traffic for full autonomy. Using computer vision, we can improve the model by including additional inputs and an ensemble of machine learning models.

Index Terms: Advantage Actor-Critic Policy, Continuous Action Domain, Plane Autonomy, Deep Neural Network, Policy Gradient Methods

Model: https://github.com/SRaswan/XplaneRL
Gym Environment: https://github.com/adderbyte/GYM_XPLANE_ML

2. Motivation

My main area of interest for research is how machine learning can accelerate aeronautics and aerospace. Simulation offers an efficient method to mathematically model complex systems. Machine learning sparked the creation of reliable and safe autonomous cars as seen by the software developed by Tesla and Waymo. There is a considerable amount of work done to evaluate the practicality of autonomous and pilotless planes, but we are far from that future because of the massive amount of data to support a multitude of real-world scenarios, like weather dynamics. I want to apply reinforcement learning concepts to build a deep neural network agent that can improve our current models of autonomous flight.

3. Introduction

Figure 1: The inner cockpit of a Cessna 172SP has a yoke/stick and rudder pedals as the main controls. The yoke controls the ailerons on the wings of the plane, allowing pitch and roll control. The rudder pedals allow for yaw control, which changes the heading when the plane is horizontal to the ground [8].

The goal of the agent is to successfully maintain a constant altitude and heading by minimizing the change in state space parameters over time, using X-Plane [8]. X-Plane is a realistic and accurate flight simulator, providing engineering tools to predict the flying capabilities of fixed and rotary-wing aircraft. It has a visual output that can use the user keyboard and mouse or controller inputs to manipulate the plane, and for developing the agent, I used a Cessna 172SP plane. It can simulate different geographical locations based on real-world terrain, weather, and airports.

Using the gym environment found at [9], the agent interacts with X-Plane by exploring and training from the gym environment that programmatically communicates with the flight simulator in real-time. The gym environment actually calculates the reward based on the altitude, heading, and crash event. It also retrieves the states from the environment and sends this raw reward to the agent. To properly communicate with the running simulation, the FlyWithLua and X-Plane Connect plugin is used [10], [24]. FlyWithLua is used for resetting the environment each episode when the plane crashes or its engine is turned off, while the X-Plane Connect plugin is used for programmatically retrieving the state space parameters from the plane as it moves in the simulator.

The 11 state-space parameters are used to observe the state of the plane for training data, including the velocity, pitch, altitude, and heading. The result should be a constant heading, which is the direction in which the aircraft points in the longitudinal axis, pitch, which is the tilt of the nose of the plane up or down, and the roll, which is manipulated by disproportionate lift on each wing to spin the plane sideways [8]. The agent would need to learn to keep the pitch, heading, and roll constant.

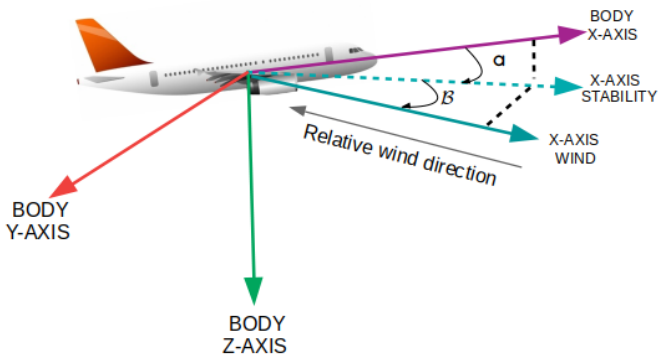

Figure 2: The agent attempts to achieve a perfect pitch and heading to maintain altitude, which requires an upwards angle from the horizontal [9]. The image is found on the gym environment GitHub.

I am using an Advantage Actor-Critic (A2C) model to write a custom policy for the agent. It is a synchronous algorithm that uses policy-based (actor) and value-based (critic) neural networks [17]. The agent and gym environment base communicate using the state spaces, where the gym environment calculates the reward and the agent uses it to evaluate each action in the 4 action space parameters. Based on the rewards, it then determines what action to take by manipulating the next applied action, which includes the latitudinal stick, longitudinal stick, rudder pedals, and throttle. All four actions are always applied, but how much and what direction of each action will change. The action space parameters are continuous and the agent uses activation functions from TensorFlow [5], a Python reinforcement learning toolset for algorithms like Advantage Actor-Critic models. The critic in the Actor-Critic agent estimates the Q-value function for the actions and the actor updates its policy gradient based on the critic [14]. However, the action space is continuous for the 4 actions, so there can be an infinite range between a maximum and minimum for each action. The agent, therefore, needs to find a specific sample of that range that is beneficial to the current plane state. The gym environment only determines what the rules and goals are, which is a constant altitude and heading. Respectively, the agent determines what the best way to reach that goal is.

Both the actor and the critic neural networks update their weights, dictating the predicted return and the best action to take in the next timestep. A2C is a policy gradient method that is on-policy, meaning that it estimates the possible reward for taking an action at a specific state, but it assumes the current policy is being followed [20]. The agent collects state data as it explores, then it updates the policy after updating its weights based on the reward, and finally starts over by following the updated policy. Our goal is to test how efficient and effective deep A2C agents are in autonomously flying a plane.

4. Development

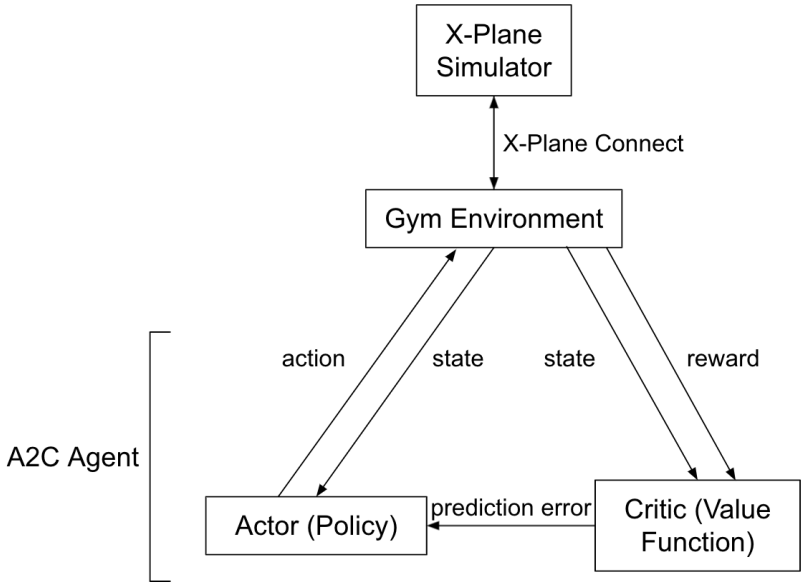

Figure 3: X-Plane sends and retrieves data from the gym environment by the X-Plane Connect plugin. The gym environment sends states to both the critic and actor. The reward is sent to the critic from the gym environment, while the critic calculates the prediction error through the reward and states. The prediction error determines the actor's desired action, which is sent to the gym environment.

A gym environment communicates with the X-Plane simulator, retrieving and sending data based on the agent. Some modifications to the gym environment were made to send only the necessary states and have an updated reward calculation function. Setting up the gym environment to connect to X-Plane consists of setting up a situation file and a reloading Lua script that sets up the initial position of the plane. After installing and configuring FlyWithLua and X-Plane Connect, I transferred all other Lua scripts to inactive folders so the plane properly resets when it crashes to its initial position. In the gym environment, I created a custom agent in the examples folder, which uses Python 3.6 with TensorFlow 1.15.5 installed. By modifying the X-Plane Connect on the simulator and the gym environment to both be the latest version, the correct number of bytes is able to communicate between the environments. I updated the ports that the gym environment uses to access data from X-Plane.

I increased the acceptable altitude range around the target altitude to have a slightly larger leeway for the plane to gain reward. The target heading is 164 degrees with a leeway of 15 degrees and the target altitude is 6000 feet with a leeway of 10 feet. The environment calculates the reward with a pairwise distribution function to portray the difference between the current state and target state. The metric used for this is a cosine function that computes the distance between the states [22]. I also made sure that the simulation pauses every time the reward is calculated so no extraneous forces are acting upon the plane when flying, preserving the Markov Decision Process [11].

After starting up X-Plane, choosing clear weather conditions and the situation file installed earlier, the Cessna 172SP is already in flight because of the Lua script. The random agent example in the

gym environment can be run, which will manipulate the plane if correctly configured and send random actions to the action space. On the simulator, the plugins tab will show both FlyWithLua and X-Plane Connect running. For manual control of the plane, the center yoke stick allows for pitch and roll manipulation, the rudder pedals allow for heading/yaw maneuvering, and the throttle controls the power output and RPM of the propellers [8]. For optimal training, I did not touch the controls because it introduces lurking variables. When the simulation is configured in headless mode, the gym environment supports quicker training and less resource utilization.

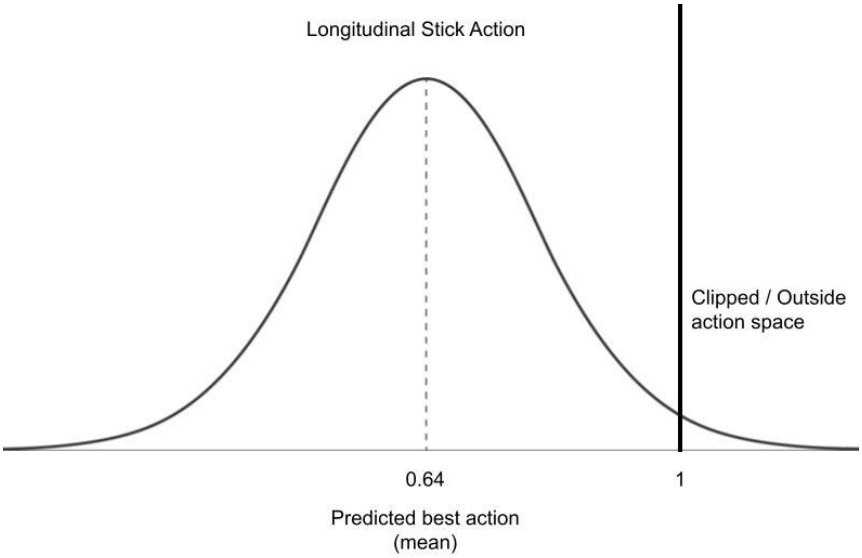

Figure 4: The longitudinal stick action is a continuous value from -1 to 1. The actor determines the mean and standard deviation for the normal probability distribution of all four actions.

The actor is the policy that determines the next possible action, while the critic is a Q-value function that determines if the action taken was good or bad, further updating the agent. First, the actor has 2 models and 400 neurons that take the 11 continuous states and give 4 continuous actions. Each model has one fully connected layer. The goal of the actor is to get possible continuous values between the possible minimum and maximum of the actions (usually -1 to 1). The weights are initialized using the Xavier normal initializer [18]. The 2 models result in a multivariate Gaussian distribution, where each continuous action has a mean and standard deviation coming from the hidden layers. A sigmoid activation function is used for the mean and standard deviation, creating a probability gradient for both. The farther from the actor's recommended action, the less likely that action is to execute. This stochastic policy allows the agent to explore while also trending toward the recommended action by the actor [23]. The output is the probability distribution that is clipped to be between our maximum and minimum values for each of the four actions.

Next, we use the Adam Optimizer algorithm to optimize the loss functions for both the actor and critic, while the critic also uses a mean squared error loss function for Q-values. The optimizer is where the model actually decides what action to take next because it determines which direction a step should go to minimize the loss function [6]. This creates the main intent for the next action. The length of the step in the x-direction for the predicted loss function is determined by the learning rate. A very small learning

rate means that the models do not deviate much from their current action, while a very big learning rate means the model may overshoot the minimum of the loss function. In addition, values with a lot of variances, like standard deviations and means with a lot of deviation, do not impact the loss functions as much. The actor loss function is a log variant that quantifies and normalizes that loss in a logarithmic scale [25].

The critic has its own model which also uses 400 neurons and Xavier as its initializer. The model has one fully connected layer. It only has an output of a one-dimensional return based on the reward from the environment. Each episode changes when the plane crashes, making a large loss in reward. More time at the same altitude means a decrease in loss.

$$R_t = \sum_{k=0}^{\infty} \gamma^k * r_{t+k}$$

Figure 5: Bellman expectation equation can be used to derive the advantage function. R_t is the advantage, also known as the return. γ is the discounting factor. r_{t+k} is the future reward [21].

The agent learns from the critic's evaluation of the current state and action. The prediction error, or the temporal difference error, is how the critic calculates the Q-values, which is the predicted reward for the future [14]. Then the actor acts on this information by adjusting what it believes should be the next action to get more optimal states.

The temporal difference target is the reward that the critic calculates for future states at that timestep. The same state can lead to multiple different episodes, and therefore different returns (R_t). The temporal difference error is the temporal difference target subtracted by the current state's reward [23]. Therefore, if the current temporal target is the same as the current state's reward, then the error is zero. The future reward values are usually multiplied by some discounting factor to only get a portion of the future rewards, which makes it an advantage function. If the advantage is positive, then it will trend towards that direction.

The actor has an output of the multivariate probability distribution [4]. The sampled values for each action from the probability distribution contribute to the advantage function. Therefore, if the advantage is positive when deemed good, the mean trends toward that direction on the axes of the normal distribution. The standard deviation also updates based on the rewards as the timesteps increase. This means it is more likely for the agent to choose that action as the mean of the normal distribution trends toward that desired action.

$$V(s_t) = r + \gamma * V(s_{t+1})$$
$$\delta = V(s_t) - r - \gamma * V(s_{t+1})$$

Figure 6: The Bellman expectation equation has $V(s_t)$ as the temporal difference target, which is the current return. r is the current reward when going from the current timestep to the next. $V(s_{t+1})$ is the return from future possible episodes. δ is the temporal different error that the neural networks try to minimize to zero [19].

The critic outputs the temporal difference target, $V(s_t)$, at that current state. The reason we use the mean squared error loss function for the critic is to update its weights because the closer the temporal difference error, δ, is to zero, the better the model [19]. This will show convergence because the model has no need to change. The recursive equation for the target means that the error is close to zero when the target is close to the current return, r, and the future returns. This means the neural network will constantly try to maximize the current reward to keep up with additional future possible returns, converging the error to zero.

The pseudocode for the agent incorporates all aspects of an Advantage Actor-Critic Method [16].

Algorithm Advantage Actor-Critic

```
Assign random weights for the actor and critic
Initialize the actor with two models (Multivariate normal distribution)
Initialize the critic with a model (value function)
For each episode:
    Reset the environment and global variables
    For episode length and if not crashed:
        Retrieve the states from environment
        Actor decides desired action (mean and standard deviation)
        Run action from probability distribution and retrieve reward
        Critic predict TD target and TD error (Bellman expectation equation)
        Update Critic weights by loss
        Update Actor weights by loss
        Update the state
    End for
End for
```

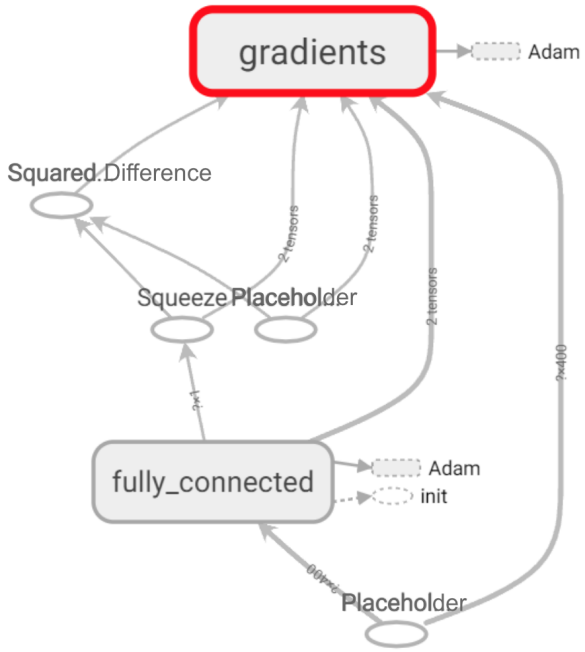

Figure 7: A TensorBoard graph model shows a TensorFlow model that shows the model's structure with computational nodes, including inputs, outputs, and shapes.

After the model finishes running and is saved, the results and reward values are viewed through TensorBoard, which also shows a schematic of the series used to develop the actor and critic. The gradient is all of the probability gradients that determine the 2 models of standard deviation and mean for the actor and the critic's model. These are optimized by Adam to minimize loss. The functions and shapes, such as mean squared difference error and squeezing the tensor, are shown as nodes. Squeezing a tensor means a dimension of the model is removed before it is shown as a gradient [5]. The placeholder has the states and actions for the actor and critic to observe.

5. Conclusion

Figure 8: The training started at a reward of 0 and had a duration of 2 hours, 18 minutes, and 28 seconds with a 16 GB of RAM, i7-9700k, and Ubuntu system. The entire training period had four crashes, which are represented by the spikes in reward decrease (116 steps at 11m 34s, 146 steps at 14m 44s, 167 steps at 16m 53s, 258 steps at 4m 9s).

This model was training for about 2 hours and 20 minutes for 1.95 thousand timesteps. The return converges to zero instead of a positive value because it is extremely difficult for the agent to keep the plane at the exact target altitude and heading at all times. Although it is much slower than discrete and single variate spaces, it is able to converge at about 570 timesteps, or 43 minutes, into training. The agent is able to stabilize by keeping its heading, pitch, and roll constant at convergence. The training instability onsets were due to exploration phases and were recovered from after hours of training. After an hour, it seems to be completely stable, not veering to the left as it would normally do and going relatively straight. It would initially veer to the left because the center of gravity for a Cessna 172SP is slightly on the left. There were a total of 4 crashes throughout the entire training period around 15 minutes, most likely because of the small learning rate, making it difficult for the plane to reach a minimum loss but eventually more accurately reaching it after about 30 minutes. The initial low loss was due to how the initial state of the plane was already positioned with a relatively good pitch, heading, and roll, however, the model quickly faltered after 11 minutes possibly due to the random weights for the actor and critic. An Advantage Actor-Critic agent will succeed in maintaining a constant altitude and heading for a plane after a sufficient amount of training and exploration.

Some challenges I faced when developing the agent had errors when defining the actor's outputs because different activation functions needed to be used. I initially attempted to discretize the action space into bins and use the softmax activation function for the 2d array, however, this did not work because the action space had incompatible sizes to softmax when using Stable Baselines [17]. Stable Baselines is another reinforcement learning library that supports A2C algorithms. In addition, there were many compatibility errors between X-Plane Connect, Python, and TensorFlow.

6. Future Work

Possibly different models should be optimized and used for maximizing reward. For Advantage Actor-Critic Methods, the agent executes an action, then calculates a reward, and then achieves another state. Therefore, it constantly learns in each timestep while discarding older data. A larger environment like this may make the agent falter more because it would take longer to converge and be more stable. Different algorithms, like ones with replay buffers that are off-policy, can save past data to make more accurate models. An example of this is Deep Q-Network algorithms, however, this does not support continuous data on Stable Baselines and TensorFlow [3].

In addition, more real-world situations have different variables that the agent would need to account for, like weather and turbulence. Future projects would also include other plane functions like taxiing, takeoff, and landing which would allow the plane to be fully autonomous. Convolutional neural networks would also be necessary for full autonomy because computer vision would leverage the statistical regularities in rich visual inputs and add more accurate state observable parameters to navigate the environment [1].

Adaptive learning rates could be implemented to have a very accurate model for minimizing the loss function [13]. This is so that the learning rate is dynamic, starting larger and then decreasing over time to hit the approximate minimum. The convergence of the model would be much quicker and more accurate, something that this Advantage Actor-Critic Model needs to improve upon. The reward system

by the gym environment could also be more robust, allowing a stronger reward gradient for altitude and heading so the model trends toward the correct direction better.

7. Acknowledgements

I would like to thank Ali Ekin (Remedy Robotics Machine Learning Engineer) for support and discussion about policy gradients and activation functions. I would also like to thank Olagoke Lukman for providing an open-source gym environment for X-Plane. Without the gym environment, this research would not be possible.

8. References

[1] Becker-Ehmck, P., Karl, M., Peters, J., & van der Smagt, P. (2020). *Learning to Fly via Deep Model-Based Reinforcement Learning*.

[2] Belkhale, S., Li, R., Kahn, G., McAllister, R., Calandra, R., & Levine, S. (2020). *Model-Based Meta-Reinforcement Learning for Flight with Suspended Payloads*. doi:10.1109/LRA.2021.3057046

[3] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *CoRR, abs/1606.01540*. Opgehaal van http://arxiv.org/abs/1606.01540

[4] Do, C. B. (2008, October 10). The Multivariate Gaussian Distribution. *Stanford University*. cs229.stanford.edu/section/gaussians.pdf

[5] Introduction to RL and Deep Q Networks: TensorFlow Agents. (n.d.). *TensorFlow*. tensorflow.org/agents/tutorials/0_intro_rl

[6] Janner, M. (2019, December 12). Model-based Reinforcement Learning: Theory and Practice. *The Berkeley Artificial Intelligence Research*. https://bair.berkeley.edu/blog/2019/12/12/mbpo/.

[7] Kim, H., Jordan, M., Sastry, S., & Ng, A. (2004). Autonomous Helicopter Flight via Reinforcement Learning. In S. Thrun, L. Saul, & B. Schölkopf (Reds), *Advances in Neural Information Processing Systems* (Vol 16). Opgehaal van https://proceedings.neurips.cc/paper/2003/file/b427426b8acd2c2e53827970f2c2f526-Paper.pdf

[8] Lockwood, J. (n.d.). X-Plane 11 Cessna 172 Pilot's Operating Manual. *Laminar Research 2017*. https://www.x-plane.com/manuals/C172_Pilot_Operating_Manual.pdf

[9] Lukman, O. (2019). GYM X-PLANE (Version 2.0.4) [Computer software]. *GitHub*. https://github.com/adderbyte/GYM_XPLANE_ML.

[10] Lynker, C. (2021). *FlyWithLua*. https://github.com/X-Friese/FlyWithLua/.

[11] Markov Decision Process (n.d.). sciencedirect.com/topics/computer-science/markov-decision-process

[12] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. A. (2013). Playing Atari with Deep Reinforcement Learning. *CoRR, abs/1312.5602*. Opgehaal van http://arxiv.org/abs/1312.5602

[13] Morales, E. F., & Sammut, C. (2004). Learning to fly by combining reinforcement learning with behavioural cloning. In C. E. Brodley (Red), *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*. doi:10.1145/1015330.1015384

[14] Patel, Y. (2018, June 18). Reinforcement Learning with Keras and OpenAI: Actor-Critic Models. *Medium*. towardsdatascience.com/reinforcement-learning-w-keras-openai-actor-critic-models-f084612cfd69

[15] Polvara, R., Patacchiola, M., Sharma, S. K., Wan, J., Manning, A., Sutton, R., & Cangelosi, A. (2017). Autonomous Quadrotor Landing using Deep Reinforcement Learning. *CoRR*, *abs/1709.03339*. Opgehaal van http://arxiv.org/abs/1709.03339

[16] Raswan, Shaurya. (2021, December 20). X-Plane Actor-Critic Agents. *GitHub*. https://github.com/SRaswan/XplaneRL.

[17] Sharma, S. (2021, July 4). Activation Functions in Neural Networks. *Medium*. towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[18] Stable Baselines. (n.d.). *Stable Baselines 2.10.2 Documentation*. https://stable-baselines.readthedocs.io/en/master/modules/a2c.html.

[19] Steinbach, A. (2019, June 8). RL introduction: simple actor-critic for continuous actions. *Medium*. medium.com/@asteinbach/rl-introduction-simple-actor-critic-for-continuous-actions-4e22afb712

[20] Tovar, A. D. (2020, January 3). Advantage actor critic (A2C) implementation. *Medium*. https://medium.com/deeplearningmadeeasy/advantage-actor-critic-a2c-implementation-944e98616b#:~:text=Brief%20summary%20of%20A2C,function%20by%20following%20another%20policy.

[21] Tovar, A. D. (2020, March 4). Advantage Actor Critic: continuous case implementation. *Medium*. https://medium.com/deeplearningmadeeasy/advantage-actor-critic-continuous-case-implementation-f55ce5da6b4c.

[22] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods, 17*, 261–272. doi:10.1038/s41592-019-0686-2

[23] Wang, M. (2021, January 26). Advantage actor critic tutorial: minA2C. *Towards Data Science*. https://towardsdatascience.com/advantage-actor-critic-tutorial-mina2c-7a3249962fc8#:~:text=In%20the%20field%20of%20Reinforcement,Based%20and%20Value%20Based.

[24] X-Plane Connect. (2020). *National Aeronautics and Space Administration*. https://github.com/nasa/XPlaneConnect.

[25] Yoon, C. (2019, July 17). Understanding actor critic methods. *Towards Data Science*. https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f.

[26] Zare, N., Brandoli, B., Sarvmaili, M., Soares, A., & Matwin, S. (2021). *Continuous Control with Deep Reinforcement Learning for Autonomous Vessels*. ArXiv, abs/2106.14130.