Reinforcement Learning for Simulating Personal Navigation UAV

Raphael Loren L. Dalangin
College of Computer and Information Science
Malayan Colleges Mindanao
Davao City, Philippines
rlDalangin@mcm.edu.ph

Abigail Adrianne M. Dubouzet
College of Computer and Information Science
Malayan Colleges Mindanao
Davao City, Philippines
aaDubouzet@mcm.edu.ph

Alexander Manuel Dean T. Ferrazzini
College of Computer and Information Science
Malayan Colleges Mindanao
Davao City, Philippines
amdFerrazzini@mcm.edu.ph

***Abstract*— Unmanned Aerial Vehicles (UAVs) or drones as we call them today has been in military research for the previous 150 years and just as early as 2006 was the first recorded use of drones for non-military ventures. Today with computer hardware technology catching up followed by research developments in the fields of algorithms and artificial intelligence, multiple possibilities and ideas have opened. With this, the proponents have decided to dive deeper into the subject and attempted to broaden the uses and enhance the potential of drones through the utilization of trained machine learning algorithms to develop a navigation UAV agent that is capable of following its owner. This study particularly uses the reinforcement learning algorithm Proximal Policy Optimization (PPO) to allow the agent the space to learn on its own. The agent is trained in a simulated virtual environment procedurally generated using the video game engine Unity. The game engine's toolkit ML-agent was utilized. The proponents were successful in training the navigation agent however the current model is not yet ready for production use. The navigation agent requires more training to perform better in a real-world setting.**

Keywords*— *Machine Learning, Reinforcement Learning, Personal Navigation UAV, Unmanned Aerial Vehicle, Proximal Policy Optimization

I. Background of the Study

Unmanned Aerial Vehicles (UAVs) or drones as we call them today have been in military research for the previous 150 years and just as early as 2006 was the first recorded use of drones for non-military ventures [1]. Today with Computer Hardware Technology catching up followed by Computer Science research in Algorithms and Artificial intelligence, multiple possibilities and ideas have opened in the field.

It was in 2015 when Recreational Drones became popular in the United States with approximately 1 million expected to be sold by the end of the year [2]. The main idea of recreational use of drones is the operation of an Unmanned Aerial Vehicle (UAV) for personal interest or enjoyment which includes the use of a drone to take pictures or film videos for the user.

In 2020, The global drone market in terms of revenue was worth USD 18.28 billion [3]. This includes military, commercial, and consumer use which shows how Unmanned Aerial Vehicles (UAVs) are relevant in the market and the world today.

With acknowledgement of the relevance of drones today, the proponents have decided to dive deeper into the subject and broaden the uses and actions of drones through training an algorithm for the Personal Navigation UAV which is to follow its owner. With the help of machine learning, particularly reinforcement learning and to create a simulated environment for it to train in.

The simulation was done through the video game engine Unity. The game engine's toolkit ML-agent was utilized in the said simulation. As for the reinforcement learning algorithm to be used, the Proximal Policy Optimization (PPO) was selected to train the personal navigation drone. PPO makes use of a neural network to find and provide the best action based on an agent's current state and observable parameters.

The research and the simulation will be done at the homes of the proponents in Davao City, Philippines. As mentioned earlier, the simulation will occur using Unity as the simulation environment with the use of the ML-agent toolkit.

As for the limitations of the research, the personal navigation drone was created and trained in a virtual environment. The proponents focused on developing a navigational model, they assumed that the flight capabilities of the drone are already implemented. Flight capabilities include the following but are not limited to: hovering, movement, weight management, and gyroscopic stabilization. No hardware training and implementation were conducted. Additionally, the agent was given 10 million steps to train in learning to follow the owner. The reason that the proponents were only able to do 10 million steps was due to the unavailability of funds and access to utilizing better computing devices. Moreover, the specifications of the computer used to run the simulations comprise of a processor of Intel i7-8700 CPU @ 3.20GHz (12 CPUs) with a memory of 16GB, and a graphics card of 1050ti. The entire training time took 36.40 hours to do.

II. Related Works

Liu et al. introduced the self-play actor-critic (SPAC) method through training agents to play computer games by themselves [4]. The proponents compared the mentioned method with the deep deterministic policy gradient (DDPG) and proximal policy optimization (PPO) algorithm. For the simulation, the agents are tested in four environments. Two of the environments utilized Unity's ML-agents toolkit,

"Tennis" and "Soccer". By the end of the research, the SPAC method performed better than the DDPG and PPO algorithms.

A paper by Dudarenko et al. discusses formulating a trained algorithm for a mobile robotic platform to adapt to static and dynamically generated environments [5]. The two algorithms were trained algorithms in static and dynamically generated environments. Moreover, the learning occurs in Unity, similar to the first related work mentioned and it also utilized the IDE's ML-agents toolkit for visualization and testing. The results of the paper show that the mobile robotic platform trained in a static generated environment adapts to static and dynamically generated environments better than the mobile robotic platforms trained in dynamically generated environments.

For papers in relation to the topic, Ravichandran et al. simulate pedestrians where the goal is to make them learn separate objectives and to evaluate the learning algorithm [6]. One of the objectives of the simulated pedestrians is collision avoidance which is accomplished through a collide module. How it works is that the agent is surrounded by a collision-cell with a given polar coordinate.

Another paper in relation to the topic is by Gonzalez-Garcia et al. which is about creating a controller for an unmanned surface vehicle (USV) with the use of Adaptive Dynamic Programming (ADP) and deep reinforcement learning (DRL) as basis [7]. The environment for deep neural network training was made in Python3 and TensorFlow. As for simulation results that were validated through the VTec S-III USV mathematical model, the controller with ADP and DRL as its basis surpasses the nonlinear PID.

For the last paper, Zhao, Ma, and Hu studied about solving the path-following control problem [8]. It would be solved through deep reinforcement learning with random braking (DRLRB). It has been shown that the DRLRB works as USVs are trained to slow down and stop due to random braking.

Two papers utilized Unity and their ML-agents toolkit. The first paper uses Unity ML-agents toolkit "Tennis" and "Soccer" for its environment for the simulation. Another paper made use of Unity's ML-agents toolkit for testing and visualization. As for the remaining papers, they are related to the paper in a way that focuses on collision avoidance, unmanned technology, and path-following. The following topics are helpful in formulating a way to simulate a UAV as a personal carrier.

III. Materials and Methods

A. Algorithms Used

For the algorithms that will be used, the proponents have chosen a proposed new family of policy gradient methods known as PPO or Proximal Policy Optimization algorithm for the agent to learn on. Since the agent receives visual input in addition to numerical coordinates, a CNN or Convolutional Neural Network was utilized. The algorithm will be applied in the same environment and parameters in Unity using Unity's Machine Learning Agent or ML-Agents, an open-source unity toolkit.

B. Proximal Policy Optimization Algorithm (PPO)

A Policy Gradient Method is a type of reinforcement technique that relies on optimizing parametrized policies with respect to the expected return, a long-term cumulative reward, by gradient descent. Gradient descent is an optimization algorithm that is used when training a machine learning model which is based on a convex function that tweaks its parameters iteratively to minimize a given function to its local minimum [9]. With this, the proponents have chosen a new family of policy gradient methods for reinforcement learning that alternates between sampling data through interaction with the environment and optimizing a sub-objective using stochastic gradient descent. Knowing that current standard policy gradient methods carry out one gradient update per data sample. The new family of policy gradient methods known as the Proximal Policy Optimization (PPO) encompasses a novel objective function that permits multiple epochs, an occurrence or a time marked by an event that begins a new period of mini-batch updates. The algorithm has benefits over the Trust Region Policy Optimization (TRPO), another policy gradient method in reinforcement learning that avoids parameter updates that change the policy excessively, making it more general, simpler to implement, and has a better sample complexity empirically compared to the latter. Tests on PPO on different benchmark tasks involving simulated robotic locomotion and playing video games shows that the algorithm outperforms other online policy gradient methods and overall is balanced between sample complexity, simplicity, and wall-time.

C. Convolutional Neural Network (CNN)

CNN, also known as Convolutional Neural Network, is an algorithm that can solve complex problems. One of the tasks that CNN can perform is image classification. The components of a CNN are the following: convolution layer, pooling layer, activation function, and fully connected layer [10]. Based on the stated source, image classification occurs in the Convolution Layer and the Pooling Layer decreases trainable parameters. Once the first 2 components are finished, the Fully Connected Layer gets information from the first 2 components to create an output. Lastly, Activation Function comes in where the concept of Rectified Linear Unit (ReLU) is introduced.

D. Tools Used to Conduct Machine Learning

The environment chosen by the proponents that will be used for the machine learning will be Unity, a cross-platform engine that is supported on Windows, Mac, and Linux. The engine itself supports building games on multiple platforms. Developed by Unity Technologies written in C++ runtime while C# on the Unity Scripting API. The ML-Agents or Unity Machine Learning Agents SDK, an open-source project that enables games and simulations to be used as environments for training, is a key part of machine learning. Moreover, the Realistic Drone package created by AnanasProject was used as the model to depict UAV agent [11]. The package was made by utilizing a PID controller with the following formula which was taken from Wikipedia. The values of

K_p, K_i, *and* K_d are non-negatives.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de(t)}{dt}$$

E. Unity Machine Learning Agents SDK (ML-Agents)

The ML-Agents or Unity Machine Learning Agents SDK is an open-source project that enables games and simulation to be used as environments for training and machine learning by making it repeat an iteration on a fabricated situation on the environment until it completes an x number of iterations inputted by the user until it finishes the number of iterations inputted and gathers the data from the outcome of the x number of iterations on a situation from the environment. Knowing this, the proponents have chosen and believe that the game engine and open-source program are suitable in fulfilling the role of providing the environment in which the reinforcement learning algorithm will reside and satisfy the current needs of the project.

F. Flowchart

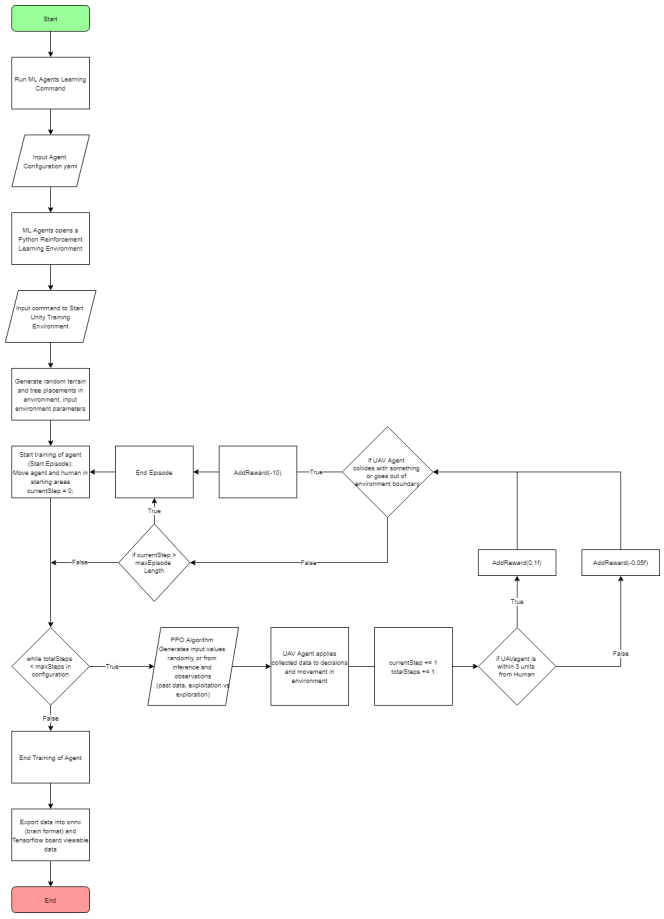

Fig. 1. Flowchart of the Personal Navigation UAV.

For the first step, the ML-agents learning command should run. Afterwards, the .yaml file of agent configuration will be inputted. The next step is that the Reinforcement Learning Environment which utilizes the Python programming language will be opened through the ML-agent and that is when the user will click to start the training environment in Unity. After starting, a generated environment will be created.

While the steps are not equal to the maximum set value, the PPO algorithm is running through inputting variables either at random or based on observation and inference. After inputting the data, the data will be applied during training through increasing the currentStep and totalStep value by 1.

In terms of rewards, if ever the drone is within 3 units from the human the reward will increase by 0.1. On the other hand, if the drone is not within 3 units from the human, the reward will be deducted by -0.05. Another deduction from the reward would occur if the drone collides with anything. The reward will be deducted by 10.

Once the PPO algorithm and the checking of rewards are done, then the whole process will loop until the totalSteps is no longer less than the maxSteps which is set to 10,000,000.

G. Config Parameters

Before conducting the training of the agent, a config file was created to provide the algorithm with a set of parameters.

The hyperparameters are listed as follows:

- *batch_size*: 256, pertains to the number of experiences for each iteration of a gradient descent update.
- *buffer_size*: 2560, describes how many experiences the agent should collect before learning and updating the model.
- *learning_rate*: 0.0003, corresponds to the strength of the iterated gradient descent update step.
- *beta*: 0.005, pertains to the randomness of the policy. The value of this variable dictates the explorative trait of the agent.
- *epsilon*: 0.2, describes the threshold differences of values between old and new policies.
- *lambd*: 0.95, corresponds to how the agent relies on the current collected value estimate when performing a calculation that updates the stored value estimate. The value of this variable dictates the bias of the algorithm.
- *num_epoch*: 3, pertains to the number of passes to the experience buffer during gradient descent.
- *learning_rate_schedule*: linear, describes that the algorithm should attempt to learn and provide values in a linear fashion.
- *normalize*: false, describes whether the agent normalizes the collected data. This is set to false since the agent is trained to process discrete values.
- *hidden_units*: 256, pertains to how many units are there in the neural network.
- *num_layers*: 1, corresponds to how many hidden layers are present. This does not include the hidden layers of the used CNN.
- *vis_encode_type*: simple, describes the encoder type in processing visual input or observations. The agent uses a simple CNN consisting of two convolutional layers.
- *goal_conditioning_type*: hyper, describes that the agent makes use of a HyperNetwork in generating the weights of the policy based on input.
- *gamma*: 0.99, corresponds to how far into the future should the agent consider the possibility of rewards. This value is set high since the agent is thought to survive in a prolonged environment.
- *strength*: 1.0, the multiplier of the reward given to the agent.

- *time_horizon*: 64, the number of experience steps the agent should obtain before adding to the experience buffer.

H. Behavior Parameters and Actions

This section describes the visual and numeric data that the agent will observe before outputting values to be used to perform actions.

During training and performance testing, the agent has a numeric vector observation of 7, and a visual observation of colored 64 x 64 pixels (4096 pixels to analyze using the simple CNN). A total of 4103 values to analyze and learn from.

The 7 vector observations are listed as follows:

- 3 vector observations pertain to the x, y, and z coordinates of the human being in the environment.
- 3 vector observations pertain to the x, y, and z coordinates of the agent in the environment.
- the last vector observation pertains to the distance between the human and agent in the environment.

In terms of actions, the agent outputs a total of 8 discrete values. These values are stored in 3 different discrete branches.

The 3 discrete branches are listed as follows:

- branch 0 with a size of 2 pertains to the decision of the agent to move within the x and z-axis. A value of 0 denotes the agent staying in place while a value of 1 denotes the agent moving forward.
- branch 1 with a size of 3 pertains to the decision of the agent to move within the y-axis. A value of 0 denotes staying in place, 1 denotes flying up, while 2 indicates flying down.
- branch 2 with a size of 3 pertains to the rotation of the agent along the y-axis. A value of 0 denotes no rotation, 1 denotes rotating to the left, while 2 indicates rotating to the right.

I. Environment

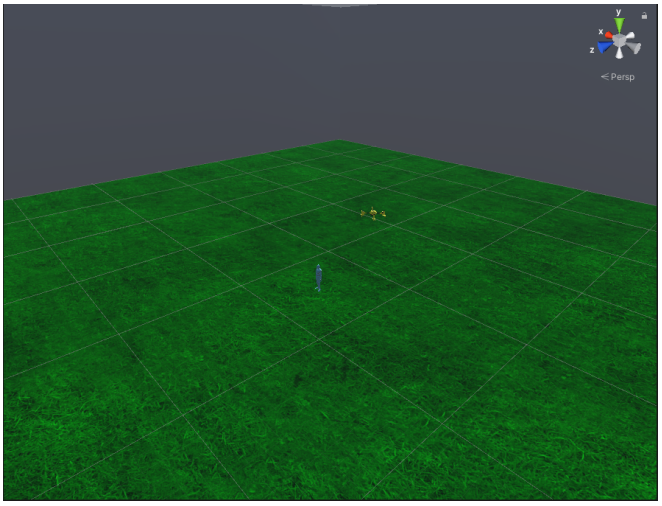

Figure 2. The environment before training.

Figure 2 illustrates the environment of the agent before training. As seen here, the environment is bounded by gray walls. The walls will serve as the barrier and boundary of the agent. The proponents chose the color gray to simplify the visual pixels that the agent will process. The blue humanoid entity in the middle represents the owner of the agent. The yellow drone represents the visual model of the agent. 15 parallel agents are trained simultaneously.

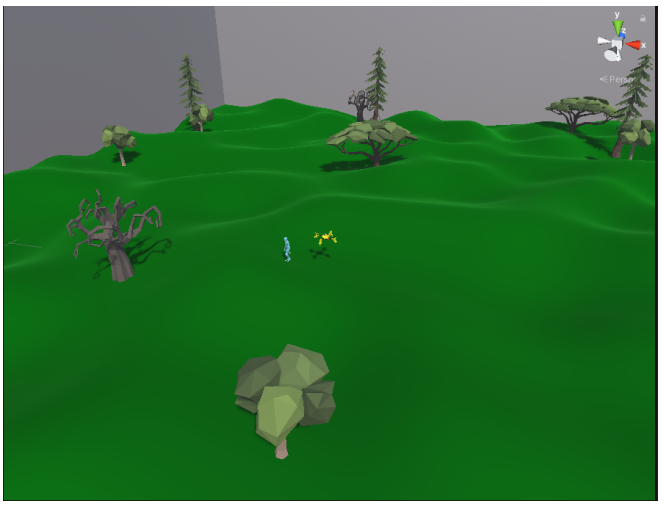

Figure 3. The environment during training.

Figure 3 illustrates the environment of the agent during training. An algorithm was coded by the proponents to procedurally generate a hilly environment and randomize the position of trees. Each of the 15 parallel environments have their own unique terrain and tree placements.

Initially, the proponents created a system that procedurally generates a randomized terrain for every episode start, however, this method proved to be straining to the system. To reduce and optimize the training time required, the proponents chose to perform the procedural terrain generation and tree placement only at the start of the training.

Figure 4. The visual input of the agent.

Initially, the proponents gave the drone cameras in all 6 directions (forward, backward, up, down, left, and right).

However, this method increases the complexity of the agent. To perform the needed task, the proponents reduced the camera count to one retaining the forward-facing camera. The pixel of the camera is set to only receive a 64 by 64 resolution. Figure 4 illustrates what the agent sees.

IV. Results and Discussion

The agent took the following time to finish each milestone:

- approximately 6.75 hours to reach 2 million steps.
- approximately 7.28 hours to reach from 2 million steps to 4 million steps.
- approximately 7.99 hours to reach from 4 million steps to 6 million steps.
- approximately 5.9 hours to reach from 6 million steps to 8 million steps.
- approximately 8.46 hours to reach from 8 million steps to 10 million steps.

A total of approximately 36.40 hours to reach 10 million steps.

Figure 5. Cumulative Reward graph.

The graph above shows the cumulative reward or the maximized expected return of the agent per step. The agent was trained to do an increment of 2 million steps for a total of 10 million steps for the simulation which explains the dips. The Agent did well on the first 2 million and 4 million steps showing equal rewards on 15 except for the slight curve on the 2 million to 4 million mark. The agent's rewards dipped and remained around 5 on the 4 million to 6 million then suddenly surpassed the former rewards hitting around 20 on 6 million to 8 million steps and without showing signs of stopping, continued to hit around the 25 and 30 mark before dipping back around 20 nearing its terminal state which was caused by a crash from lack of available RAM.

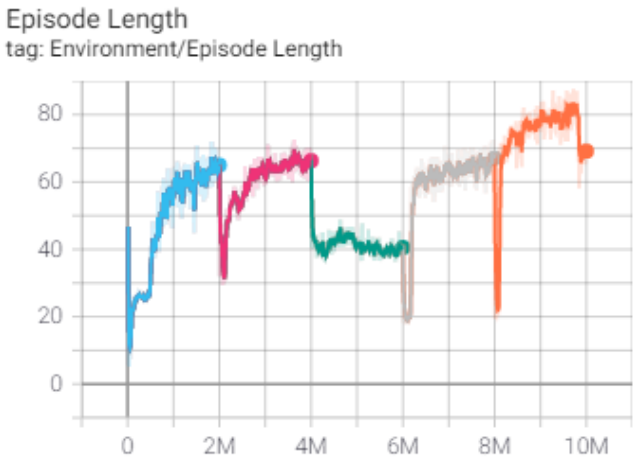

Figure 6. Episode Length graph

The graph above shows the episode length or the total flight time of the agent before colliding or finishing an episode with an increment of 2 million steps for a total of 10 million steps for the simulation. The graph's interpretation is similar to the Cumulative reward except for it starting on the 10 reward mark increasing to around 60 on its first 2 million increments showing a bit of increase barely touching the 70 mark on 2 million to 4 million. The rewards drop and remain around on the 40 - 45 reward mark on 4 million to 6 million. The agent's episode length then drops to around 20 for a bit then suddenly rises back to the 60 - 70 reward mark on its 6 million to 8 million steps. Showing no signs of stopping, it manages to surpass its former reaching around the 80 mark and drops to around 70 nearing its terminal state.

Figure 7. Policy Loss graph

The graph above shows the Policy loss from the Policy Gradient Reinforcement Learning approach which shows the agent's effort to minimize the mean squared error of the algorithm. The whole graph shows an increase-decrease pattern showing the agent's attempts in minimizing the error through trying different kinds of approaches in a randomized environment. The first 2 million steps show a similar increase and decrease of the trend, touching the 0.044 mark and maintaining in between the 0.049 and 0.051 mark. The 2 million to 4 million steps show a much wilder trend though showing signs of improvement in minimizing the error. Starting with an uptrend surpassing the 0.051 mark maintaining around the 0.049 to 0.051 mark while a sudden downtrend in the 3 million steps mark also surpassing the 0.041 mark. The 4 million to 6 million steps shows a much

better trend showing a more visible minimization of error and a controlled amount of increase-decrease of the trend steadily staying around the 0.043 mark to barely touch the 0.052 mark. The 6 million to 8 million steps showed a promising start with a controlled trend in between the 0.044 and 0.050 mark until it surpassed 7 million steps where it showed wild movements reaching almost 0.042 while surpassing the 0.052 mark. The 8 million to 10 million steps show another wild collection of uptrends and downtrends surpassing both the 0.042 and the 0.052 mark but showing good results on the 7 million steps mark while ending on 0.049 on its terminal state.

Figure 8. Value Loss graph

The graph above shows the Value loss of the agent while performing the simulation. The changes of reward for each 2 million increment is a good sign for the learning of the Agent as we need to understand that the value loss should not be stationary. The increase of loss in the first 2 million steps of the agent is an indication of good exploratory performances starting from the 2 reward mark to maintaining in between the 12 and 16 mark. The 2 million to 4 million steps show a downtrend but a steady movement in between the 8 to 12 mark. The 4 million to 6 million steps show an uptrend from the previous 2 million increments but have a more erratic movement moving from touching the 20 and going below the 12 reward mark to moving steadily in between the 12-16 mark. The 6 million to 8 million steps show a much smaller downtrend and have a steady movement in between the 11 - 16 mark. The 8 million to 10 million steps show a wilder collection of uptrends and downtrends touching the 11 reward mark to surpass the 21 mark.

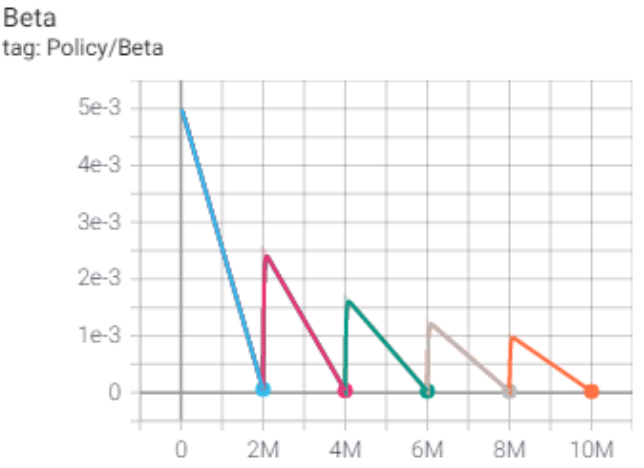

Figure 9. Beta Graph

The graph above shows the Beta Distribution which controls the modeling of uncertainty about the probability of success of the agent in its attempts in learning and improving in its simulation. The aim of the agent here is Exploration and Exploitation in which we can see from the graph that in the agent's first 2 million steps, he started from the highest mark which is 5e-3, and ended on the 0 mark. It then continued an exponential decay as seen in the graph until it reduced the uncertainty of the probability of success to 1e-3 which is around 5 times its initial state until it reaches its terminal state.

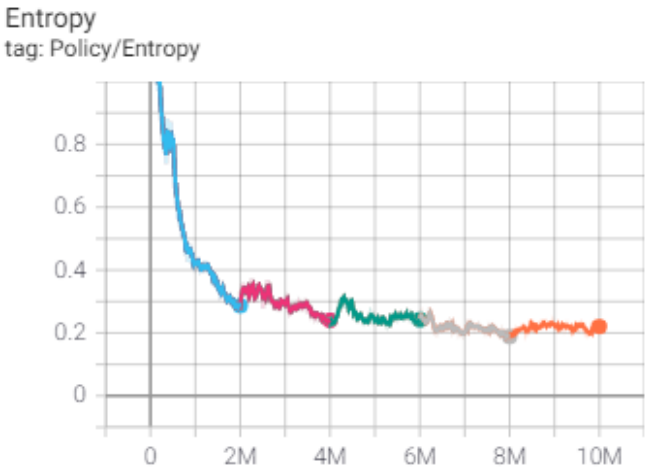

Figure 10. Entropy graph

The graph above shows the Entropy of the agent through 10 million steps of training. The Entropy reveals to us the randomness in decision making of the Agent where we can see initially it starts at 1. Its first 2 million steps are a bit far off from its next steps because of the agent optimizing its thinking, continually improving an approach he thinks works best, and continues with this decision. This is why the graph shows for the next 4 million steps in the 2 million to 6 million mark a more steady movement in between the 0.2 to 0.4 reward mark. In its last 4 million steps in the 6 million - 10 million mark, it shows a steady movement in the 0.2 to 0.3 reward mark showing that it surpassed the 0.2 mark a bit in the 8 million steps mark and ending in 0.2 in its terminal state.

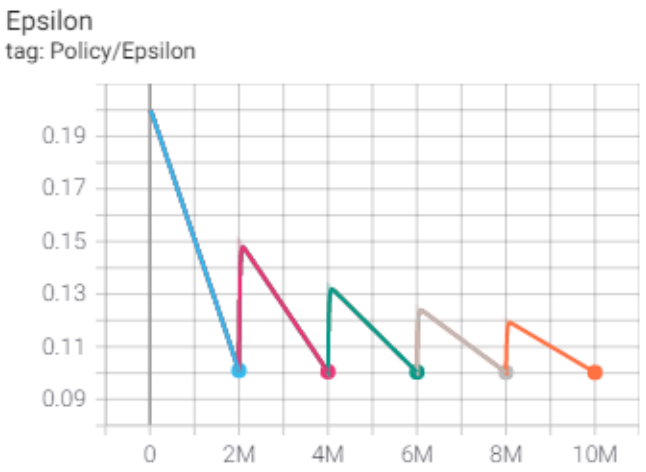

Figure 11. Epsilon graph

The graph above shows the value of Epsilon of the agent while performing the simulation. Epsilon dictates how much the policy can change and update in cases where uncommon scenarios are experienced. A higher value denotes stronger changes. This value is directly related to the epsilon found in the config hyperparameters. The agent starts at 0.2 epsilon. As training goes by and as the agent learns and fine-tunes itself, the value of epsilon decreases.

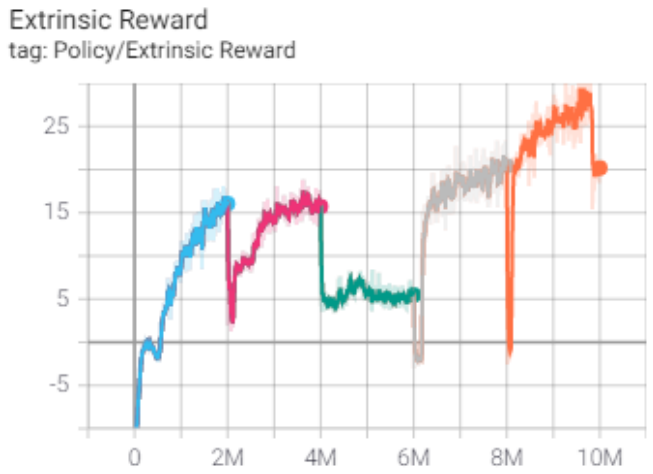

Figure 12. Extrinsic Reward graph

The graph above shows the extrinsic reward of the agent through the simulation. The extrinsic reward corresponds to the mean cumulative reward of the Agent, received from the environment per episode. If you compare both graphs of Cumulative Reward and Extrinsic Reward you will see how similar both are. The agent starts well by showing equal rewards in the first 2 million to 4 million steps. The Agent's reward suddenly dips and remains stable around the 5 to 10 reward mark. At the start of its 6 million steps mark the agent again dips reaching lower the 0 reward mark but suddenly shoots up and goes back to the 20 reward mark and shows consistent rewards that reached the 25 mark and barely reaches the 30 mark until it reached the 20 mark on its terminal state. The sudden dip at the start of the 8 million mark is caused by a crash due to a lack of available RAM.

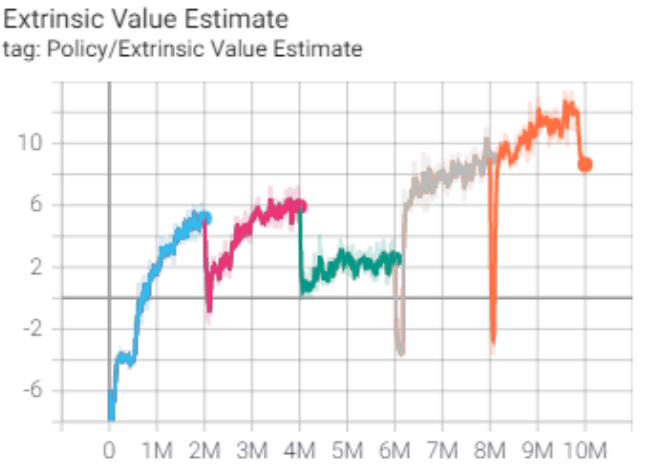

Figure 13. Extrinsic Value Estimate graph

The graph above shows the Extrinsic Value Estimate of the Agent through its run in the simulation. The Value Estimate shows the mean value estimate of all states visited by the Agent. Notice that the graph is also similar with its Cumulative reward and Extrinsic reward aside from the Y-axis inputs being different. The increasing value of the rewards shows a healthy training simulation. The Agent starts off well in its first 2 million steps barely reaching the 6 reward mark. Its 2 million to 4 million steps finally touch the 6 reward mark and show a stable flow in the area after the 3 million mark. The Agent then dips to 0 and remains stable on the 1 to 4 reward mark. After it dips again barely touching the -4 reward mark, it skyrockets and surpasses its former highest reward mark reaching the values of 6 to 10. It then continues rising above the 10 reward mark in its last 2 million steps and stops at the middle of the 8 and 9 reward mark at its terminal state.

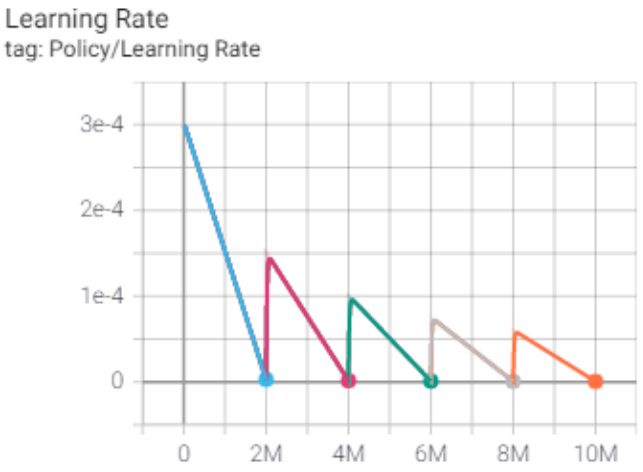

Figure 14. Learning Rate graph

The graph above shows the value of the Learning Rate of the agent while performing the simulation. The agent makes use of a linear setting. The learning rate dictates the weight of updates. The descent of the graph showcases the improvement of the agent's training and learning per 2 million increments.

V. Conclusion

Due to the popularity of drones in the modern era, the paper attempts to unravel the potential of utilizing drones. The following would be done through developing a navigation UAV agent that follows the path of its owner. The

PPO algorithm is what makes the drone learn on its own. Moreover, the environment is made in Unity and the entire process will be guided through ML-agent. Based on the data and details presented from the Results and Discussion, it is evident that the possibility of having reinforcement learning trained UAV drones flying around is high. However, more training is required to develop a model that is more capable than its current status. Newer iterations will need more visual and numeric inputs to perform better. Physical and hardware experimentation is also highlighted before deploying the agent with commercial, industrial, and/or personal causes.

REFERENCES

[1] Llly, *The evolution of drones: From military to hobby & commercial*, Percepto, January 15, 2019. Accessed on: October 17, 2021. [Online]. Available: https://percepto.co/the-evolution-of-drones-from-military-to-hobby-commercial/

[2] T. Luna, *New technology making drones easier, more affordable*, The Boston Globe, December 8, 2015. Accessed on: October 17, 2021. [Online]. Available: https://www.bostonglobe.com/business/2015/12/08/very-drone-christmas/bjoMHPmiidy0WHQXy6LjSN/story.html

[3] Brandessence Market Research and Consulting Private Limited*, At 12.27% CAGR, Drone Market Size to hit USD 40.9 Bn in 2027, says Brandessence Market Researc*h, Cision PR Newswire, September 21, 2021. Accessed on: October 17, 2021. [Online]. Available: https://www.prnewswire.com/news-releases/at-12-27-cagr-drone-market-size-to-hit-usd-40-9-bn-in-2027--says-brandessence-market-research-301381269.html

[4] S. Liu, J. Cao, Y. Wang, W. Chen, and Y. Liu, "Self-play reinforcement learning with comprehensive critic in computer games," Neurocomputing, vol. 449, pp. 207–213, 2021, doi: 10.1016/j.neucom.2021.04.006.

[5] D. Dudarenko, J. Rubtsova, A. Kovalev, and O. Sivchenko, "Reinforcement learning approach for navigation of ground robotic platform in statically and dynamically generated environments," IFAC-PapersOnLine, vol. 52, no. 25, pp. 445–450, 2019, doi: 10.1016/j.ifacol.2019.12.579.

[6] N. B. Ravichandran, F. Yang, C. Peters, A. Lansner, and P. Herman, "Pedestrian simulation as multi-objective reinforcement learning," Proc. 18th Int. Conf. Intell. Virtual Agents, IVA 2018, pp. 307–312, 2018, doi: 10.1145/3267851.3267914.

[7] A. Gonzalez-Garcia, D. Barragan-Alcantar, I. Collado-Gonzalez, and L. Garrido, "Control of an unmanned surface vehicle based on adaptive dynamic programming and deep reinforcement learning," ACM Int. Conf. Proceeding Ser., pp. 118–122, 2020, doi: 10.1145/3417188.3417194.

[8] Y. Zhao, Y. Ma, and S. Hu, "USV Formation and Path-Following Control via Deep Reinforcement Learning with Random Braking," IEEE Trans. Neural Networks Learn. Syst., pp. 1–11, 2021, doi: 10.1109/TNNLS.2021.3068762.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov. "Proximal Policy Optimization Algorithms" https://arxiv.org/abs/1707.06347?fbclid=IwAR1PCN4L0X05DAPohG-zTGWC4LIYcf65V43lXYasfqq-C6OO2mK3jkgzkwk (accessed, 12 Sept 2021)

[10] S. Indolia, A. K. Goswami, S. P. Mishra, and P. Asopa, "Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach," Procedia Comput. Sci., vol. 132, pp. 679–688, 2018, doi: 10.1016/j.procs.2018.05.069.

[11] AnanasProject, Realistic Drone, AnanasProject, n.d. Accessed on: September 21, 2021. [Online]. Available: https://drive.google.com/file/d/0BxkI7SoCaEk8NmVES2VoOGxHTjA/view?resourcekey=0-U5xTXONcx6v4I9ubUMP6-A