Vision-Based Unmanned Aerial Vehicle Navigation in Virtual Complex Environment using Deep Reinforcement Learning

By

JIAWEI (RYAN) LIANG
DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

MECHANICAL AND AEROSPACE ENGINEERING

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

ZHAODAN KONG, Chair

IMAN SOLTANI BOZCHALOOI

XIN LIU

Committee in Charge

2021

Contents

FIGURE 0.1. **Airsim UAV Model with Unreal Engine.**

Abstract

From driverless vehicles to mars rovers, autonomous navigation and task-taking are undoubtedly the future of robotics. Over the recent years, research in Deep Reinforcement Learning (DRL) has grown in many areas of navigation including unmanned aerial vehicles (UAVs). Most of them are far from realistic and assume perfect state observations. In many real-life scenarios like search and rescue in a complex and cluttered environment, GPS-denial tends to become a problem. Therefore, this research is interested in vision-based navigation and obstacle avoidance in realistic environments.

More specifically, this thesis aims to address the following research tasks: 1) To investigate the vision-based navigation of UAV in GPS-denied synthetic environments. This work will utilize a Variational Autoencoder (VAE) to improve sample efficiency, and develop a Proximal Policy Optimization (PPO) agent that can trace rivers in photo-realistic simulations. 2) To conduct reward shaping to deal with vision-based problems. Developing the correct reward function leads to the desired agent behavior, but it is a challenging task in vision-based learning. 3) To validate the PPO agent performance and compare it with another agent trained with imitation learning (IL). The evaluation metrics include the average distance traveled per episode, distance away from center of river, and standard deviation of actions taken.

Acknowledgments

Thank you so much Dr. Kong for all your time and guidance throughout this journey. I am really grateful to be given the opportunity to develop the foundations of DRL simulation research for the Cyber-Human-Physical Systems Lab, as well as being involved in the larger Neptune project with this spin-off research. Thank you so much Peng Wei for helping to develop the backbones of this project with many of the API's, VAE, and GH-DAgger scripts. You are the most resourceful and patient collaborator anyone could have asked for. Thank you Dr. Liu for allowing me to participate at your DRL review sessions. The advice I received from you and your students were important to the synthesis of key parts to this research. Thank you Dr. Soltani and Dr. Liu for your time in evaluating my work. Finally, I would like to thank my family and friends for all their love and support.

CHAPTER 1

Introduction

1.1. Motivation

The recent advancements in the field of machine learning has opened the door to countless capabilities and applications. Specifically in deep reinforcement learning (DRL), many researchers and companies have found success in developing prototypes for autonomous vehicles, which are revolutionizing the world of transportation for the twenty-first century. However, despite the overwhelming enthusiasm and achievements in DRL, there are still many unexplored opportunities that require initiative. For instance, autonomous unmanned aerial vehicles (UAV) applications in open and complex environments is an under-researched area with lots of potential [**44**]. Moreover, even less prior research focuses solely on vision-based navigation, which is a very complex problem on its own. On the other hand, applications such as search and rescue, reconnaissance, and network broadcast are all hindered by environmental factors such as GPS-denial and complicated visuals [**14**]. Therefore, the goal of this research is to develop an AI agent that can perform obstacle avoidance in such a complex open environment via DRL using only visual observations. The scenario chosen for this research will be to produce a policy that can trail along in simulated riverine environments using proximal policy optimization (PPO).

1.2. Literature Review

The twenty-first century has brought plenty of well-known machine learning (ML) achievements worldwide [**34**]. Many of these advancements had been made in the navigation industry, because that is one of the largest applications of autonomous robotics [**31**]. Some recent challenges of AI navigation are related to lane keeping, tracking, and cruise control [**32**]. One of the biggest things to prove was whether an algorithm can outperform conventional control theories in trajectory tracking. Indeed by 2019, a Deep Q-Learning algorithm was able to outperform Proportional-Integral Derivative controller in unmanned aerial vehicle (UAV) trajectory tracking convergence

efficiency and robustness; thus, validating the viability of ML in UAV controls [**26**]. Then, as of recent, there were numerous ML research using DQN to control UAV agents to cover network distributions and simulate disaster relief efforts [**24**] [**43**]. However, these simulations were all very simplistic, mostly operated in 1-D without UAV dynamics, and required perfect knowledge of all states. Nevertheless, these researches paved the way for ML-UAV research.

In recent years, many UAV autonomy research uses policy gradient algorithms to train their agents. Most of these UAV projects are interested in Deep Deterministic Policy Gradient (DDPG), because DDPG is able to function in continuous action space, capable of learning from other policies, and able to learn from a replay buffer [**18**] [**45**]. For example, by warm-starting a DDPG agent with imitation learning (IL), a UAV agent provides aerial surveillance by tracking ground targets while performing complicated maneuvers [**33**]. Similarly, IADRL uses IL to warm-start another DDPG agent to drive a ground vehicle carrying a UAV, then launch the UAV to reach some vertical target [**47**]. Fast-RDPG, a DDPG variant that breaks up correlation in the observation sequence, has been shown to be effective in navigating complex spaces [**42**]. All of these DDPG agents performed well for their tasks, but they required lots of local and global observations, plus perfect knowledge of states to properly function. Though DDPG is likely to still function under some uncertainty with state observations due to its use of the replay buffer, it is highly likely that DDPG will either converge sub-optimally or diverge completely due to its sample bias [**29**]. Nevertheless, this replay buffer is a huge draw factor to using off-policy algorithms, because one can manipulate the data in the buffer prior to the policy update to further encourage desirable actions. The real drawbacks are that DDPG is not easy to tune and may often experience instability or divergence due to the innate bias from its target network [**18**]; therefore, it is important to continue exploring other prior works.

On the note of ease of implementation, there are simpler methods to train navigating agents demonstrated by the numerous End-to-End (E2E) driving agents that were developed in recent years. In 2016, an E2E agent was trained on limited amounts of expert demonstrations to lane keep [**8**]. The model takes RGB images and expert commands to train a NN that takes the visual input and generates a command output. The only novelty was that they feed augmented data that shows the car in drift to encourage the agent to correct for drifting. Otherwise, the implementation

is as simple as image-in, command-out, hence the "end-to-end" pipeline process. Although the agent fails when the background becomes too complicated, it is no doubt inspiring to be able to simplify the training and operating requirements for such research.

Speaking on E2E, another research that was really interesting is the comparison between discrete versus continuous action space networks [**37**]. Under the same scenario of lane keeping, this time the agents have way more knowledge in observation than just visuals. The discrete DQN network was compared to a continuous DDPG algorithm, and both were successful in driving around a simulated track using correctly formulated termination functions. The key finding was that the discrete agent had rougher performance, whereas the continuous agent was moving smoother and steadier around the track. This is possibly due to the discrete agent not able to converge optimally versus the continuous agent. For this reason, it is recommended that this project also use a continuous action space model.

Besides considering the action space for the model, there was also another research that looked at whether online or offline learning had the advantage for lane keeping in 2020 [**17**]. It compared 2 DRL algorithms both paired with a VAE, one being DDPG, and the other being soft-actor-critic (SAC). Interestingly, the research found that the online algorithm, SAC, was able to outperform the offline method, DDPG, in episode length, cumulative rewards earned, and final convergence. The conclusion was that the VAE+DDPG was less effective in generalization due to overload in information, including cumulative rewards and sample noise, that degraded the quality of the representation. This could be the result of DDPG being less stable in a noisy environment [**29**], but it could also be a problem with the experience trajectories being too large to effectively communicate complex information. Thus, for the sake of maintaining good representations, it is recommended to tune the size of training data to the scenario.

On the other hand, VAE is really popular for sample efficiency and generalization, which is why it is a popular choice even for UAV research. In 2019, a research found that a Cross-Modal VAE (CMVAE) model, which compresses visual data and state knowledge to form a smaller representation, was easier for an IL agent to learn from and outperform a contemporary that did not employ any state reduction models [**9**]. The same team later took the simulation-trained agent to compare its performance in real-life and found relative success [**10**]. This research most closely

resembles the desired outcome of this current project, in which one can take a simplistic approach to train a robust UAV that can operate in open environments. The big difference is that the task in this work is easier to define compared to tracing a river; the reason for this is that in both simulation and real-life test, the CMVAE only needs to identify a clearly differentiable red target from a different colored background. In true natural open environments, there are no consistent distinguishing features that will always help identify the river. Nevertheless, it is still necessary to use a VAE for its benefits, but its downsides must be kept in mind.

Stepping away from all the inspiring work, let's revisit the main objective of this research, which was to produce a robust agent with minimal requirements that can operate in open and complex environments using only vision. Some conclusions made so far include utilizing a VAE to improve simplicity, sample efficiency, and generalization; adopting a continuous action space model produces smoother and more optimal performance; and using an online algorithm that is more resilient to sample noise. For those reasons, there was an intrigue to investigate PPO, a robust on-policy algorithm. The main reason why PPO is better than DDPG is that PPO is more robust to hyperparameters, meaning it usually converges regardless of the hyperparameters used [**13**]. The reason for this is because the bounded gradient step from PPO guarantees a near monotonic improvement, thus it is way more robust.

On the other hand, because DDPG has lower variance for having a target network, it tends to have larger bias, which will often cause divergence and challenge stabilization if not properly tuned [**30**]. This fact definitely makes PPO the easier choice to learn from for new users of policy gradient algorithms. On the other hand, since this project will be fully simulated and optimized, it will not be expensive to sample from; thus, using a more sample efficient algorithm like DDPG is not necessary [**11**]. In general, the reason why DDPG has more trouble converging is because it is off-policy, and the target network introduces bias towards a distribution that might reflect sub-optimal conditions. The fact that PPO only updates using the current policy samples, means that it will always update towards the most current objective. This makes it so PPO converges faster than DDPG [**41**]. For these reasons and more, PPO has been one of the most attractive algorithms since 2017, and was ultimately chosen to continue from this point of the literature review process.

Though there are many examples of PPO achievements over the years, there are not many published works on UAV applications using PPO. Similar to the research on CMVAE agent flying through targets, there are two projects that implemented PPO, instead of IL, to maneuver a UAV through hoops in simulation. The first research, published in 2021, aims to generate minimum time-of-flight trajectories through randomized targets in a 3D space [**40**]. The project found that sparse reward was bad for convergence of PPO long-term due to the difficulty in understanding whether individual actions taken were good within the trajectory; thus, they used straight line approximation as a reward function. Their agent was able to reach 5.2% of theoretical limit, but their algorithm only works with fully accurate GPS values. Even though they randomize their gates in certain locations, the relative distance and GPS locations are still given to the agent. Thus, it is possible to expand their work by incorporate vision into their agent and removing the reliance on GPS data. The second research, published 2020, simulates a racing drone flying through hoops in Airsim, maximizing reward based on proximity to the center of targets [**4**]. Though the agent was successful in demonstration, it also required full state knowledge and only operated in low flight-speed conditions. Plus, this agent was only able to perform in the training course, they also only trained the agent to fly in a circle. And even though this work incorporates visual observations, their agent still heavily relied on the GPS to function.

On the other hand, another PPO-UAV application would be tracking a moving target. A 2020 study trained a PPO agent that can track its target based on a known travel path, and then transferred its learning to a different scenario to track a randomly moving target [**25**]. The first part of training showed that PPO was able to perform its tracking task given full knowledge of the states and trajectory of the target. However, the second agent they trained had slightly worse performance in tracking a randomly moving target. This project shows the potential limitations of PPO in dealing with distribution shift and generalization, but these are problems that all algorithms have to struggle with. Also in 2020, there was an example of PPO-UAV navigating in a static obstacle course to reach different targets [**22**]. The agent was able to avoid static obstructions, and reach preset target locations with 81% success rate. The rest of the 19% failure is simply because the agent uses 2D depth perception, which is robust enough to maneuver around corners.

In retrospect, many prior works in PPO have very simple environments. Most of them mainly focuses on path planning. Whereas, simply conducting obstacle avoidance using vision based algorithms is a clear under-researched topic. Therefore, this project is inspired to pursue vision-based DRL in high-fidelity simulations as one of its novelties.

1.3. Goals and Contributions

As discussed, there is plenty of enthusiasm for PPO research, but there is a lack of high-fidelity UAV studies done. And from the discussion above, it is clear that simple scenarios can have questionable performance in real-life applications; and most prior works require loads of sensory data to achieve meaningful results, which is not always easy to obtain in the real-world. There is also little research done to validate PPO's performance navigating open environments with limited vision-based observations. Therefore, it is of interest to investigate this novel application and the issues of approaching realism.

Since there are no work like this project, there are no high-fidelity simulations that is directly available usage. Therefore, a significant portion of the project was dedicated to creating these highly photo-realistic riverine maps. These maps will not only be used for this project, but is also available for other students in the lab to use for their own projects.

On the other hand, the project will also dedicate time to reward shaping. Most past research simply used very complicated reward signals consisting of many components based on different states and interactions. However, under the scenario of vision-based DRL, it may not be possible to utilize a complex reward signal. Therefore, this project will also investigate how to properly define a reward function that can encourage desired behaviors.

Finally, the last contribution from this project will be to compare the performance of PPO to another competent algorithm. The competitor was based on the Neptune project agent, trained using HG-DAgger imitation learning. Comparing the two algorithms that will be using the same VAE and same training course should provide some good insight on benefits of DRL.

In summary, not only will this project provide a clear outline to how to perform vision-based UAV DRL, but it will also discuss the many hardware and software compatibility issues that are rarely discussed in other works. The list of actual contributions are:

- Applying DRL on a novel application
- Creating a large quantity of high-fidelity simulation environments
- Examining reward shaping in dealing with vision-based observations
- Outperforming an imitation learning trained algorithm in performance test

CHAPTER 2

Vision-Based Reinforcement Learning

2.1. Variational Autoencoder (VAE)

There are a few key algorithms that were used in this report, the VAE, HG-DAgger, and PPO. First algorithm to discuss is the VAE; how VAE works is by solving the loss function,

$$l_i(\theta, \phi) = -E_{z \sim q_\theta(z|x_i)}[\log p_\phi(x_i|z)] + KL(q_\theta(z|x_i)||p(z)), \tag{2.1}$$

where p_ϕ is the decoder, q_θ is the encoder, and p is the normal distribution with mean zero and variance one [**2**]. The first term is the reconstruction loss that basically measures the difference between the reconstruction to the input. The second term is a KL-divergence term that pushes the input towards the a Gaussian distribution; this is because the assumption of the VAE is that the latent space is Gaussian. Solving this objective allows us to obtain a network that can take data x and output latent z, as well as taking z and reconstructing $\tilde{x}$, as seen in figure 2.1.

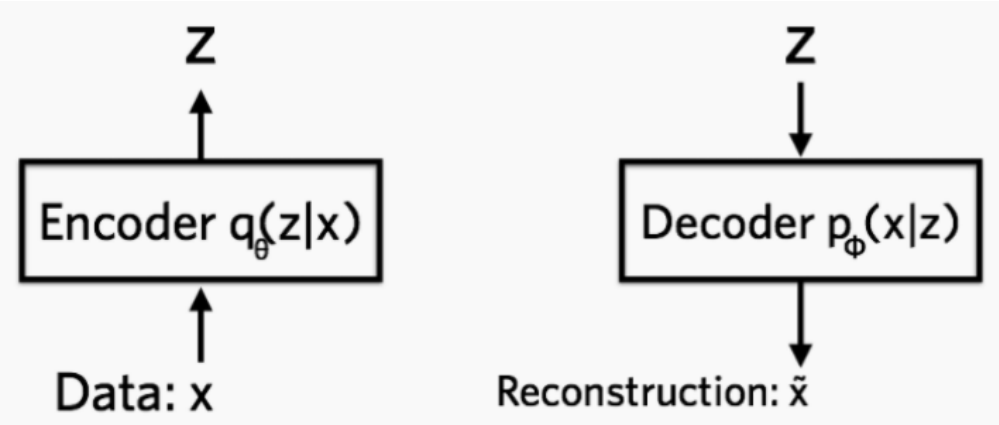

FIGURE 2.1. **VAE Encoder and Decoder [2].** The encoder takes an input and produces a latent output. The decoder takes the latent representation and tries to reconstruct the input.

The current architecture transforms a re-scaled, miniaturized view of what the onboard monocular camera will see, into an array of latent variables size 64. This latent array is the input to the PPO agent. Not only does it need a large amount of data, this project used over 600 GB of data to train the first VAE, but it is had a couple shortcomings. So a retraining was performed using an additional 100 GB worth of data. A more in-depth analysis will take place in the results section to discuss the insights to using a VAE system.

2.2. Human-Gated DAgger (HG-DAgger)

The second key algorithm for this project is a baseline for comparison of performance; and much like the VAE that was developed as part of the larger Neptune project, an imitation learning agent was also developed to travel in these riverine environments. For this project, the agent was trained using 3 iteration HG-DAgger in the same training maps to serve as a fair comparison to DRL agent.

HG-DAgger solves 1 of the biggest issues of DAgger imitation learning, which is that it requires a large amount of expert data [**1**]. Basically, HG-DAgger is a confidence-based, human-in-loop variant of DAgger that allows the expert to intervene at any time and demonstrate how to guide the agent back to safe states [**21**]. The key is that each iteration, new expert data is aggregated to the existing data pool,

$$D_i = \{(O(x_t), \pi_H(x_t)) | g(x_t) = 1, x_t \epsilon \xi_i\}, \tag{2.2}$$

where $O(x_t)$ is the observation associated with the $\pi_H(x_t)$ human expert input at state x_t. When the expert intervenes, an intervention doubt I_i is also recorded into a doubt intervention logfile I. The aggregated data set is used to update to the policy under the following equation,

$$l_t \leftarrow \|a_{\bar{nov},t} - a_{exp,t}\|^2, \tag{2.3}$$

where $a_{\bar{nov},t}$ are the novice action predictions, and $a_{exp,t}$ are the expert demonstrations. The difference between these two largely determines the policy move [**1**].

2.3. Proximal Policy Optimization (PPO)

Speaking of PPO, again it was discovered that PPO was one of the simplest policy gradient methods to implement, and due to its robustness, it was favored as the choice for this project. Traditionally, policy gradient methods alone have issues with slow convergence and learning a robust policy [**46**]. Thus, the actor-critic method was introduced to resolve those problems of a pure policy gradient actor by using a state estimator to evaluate the actions taken and reduce variance. However, this does not resolve the sensitivity issue which affects the robustness of the policy. And thus, methods like PPO impose bounds on the advantage estimation to prevent large policy moves [**19**].

PPO works similarly to all actor-critic algorithms, which relies heavily on the advantage calculation,

$$A(s,a) = Q(s,a) - V(s), \tag{2.4}$$

where $Q(s,a)$ is the state-action value estimate, and $V(s)$ is the average value of that state based on the trajectory's rewards. Simply, the advantage estimation compares the state estimates from the value function to the state returns, and guesses how good it was to be in those states. Usually this advantage is directly used to perform network updates, but for PPO this update is bounded as seen in the surrogate objective below

$$L(s,a,\theta_k,\theta) = min(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), clip(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon) A^{\pi_{\theta_k}}(s,a)), \tag{2.5}$$

where the left side is the surrogate advantage, and the right side is the clip function. The surrogate advantage measures how different the new policy is relative to the old one. The clip function can be reduced to,

$$g(\epsilon, A^{\pi_{\theta_k}}(s,a)) = clip(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon) A^{\pi_{\theta_k}}(s,a), \tag{2.6}$$

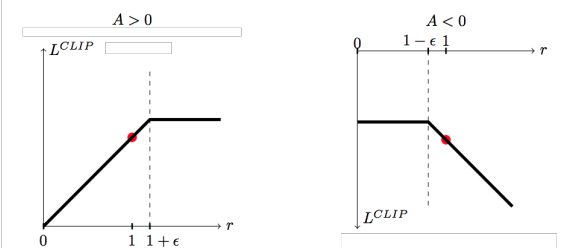

FIGURE 2.2. **PPO-Clip Surrogate Function [38].** This objective visualization shows that the clip function bounds the objective to prevent large policy moves.

which determines how large the boundary is between each iteration. More specifically, PPO-Clip uses a hyperparameter ϵ to bound the range of the policy move,

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A \leq 0 \end{cases}. \tag{2.7}$$

The figure 2.2 visualizes what the clip function does, which is to determine the direction of update. One can observe that if the advantage is positive, the objective will increase and update towards more likely actions in the future, and vice versa. Besides the direction, the surrogate advantage term is used to determine how different can the new policy relative to the old. Thus, together this function determines to direction and size of the policy shift.

In the end, the minimization between the surrogate advantage and the clipped advantage function allows for the largest and safer policy changes. Usually this bound is set to 0.2 for most cases, but it is usually within a range between 0.1-0.3 [**5**].

2.4. Neural Network Design

To maintain consistency amongst the HG-DAgger and PPO agent, it was necessary for their networks to share all the same shape. Both networks operate in a continuous action space with the same size and activation functions. Specifically, the policy is a fully connected deep network with 3 hidden layers of 512 nodes, an input layer, and an output layer with Tanh activation function. Note that an output to the network is the action that the agent will take for a given input.

Despite maintaining the same shape, there is one difference between the HG-DAgger and PPO agent, and it is that the HG-DAgger is a deterministic network while the PPO is stochastic. The Gaussian distribution attributed to the PPO network, however, actually made it rather easy to

equate the two networks. Basically, the network itself will always produce a single output ranging -1 to 1 due to the Tanh function as the mean of a standard deviation (STD); and a STD value is manually prescribed, with an appropriate STD decay, to obtain the true output of the PPO network. Maintaining the stochasticity of the PPO network was important because the surrogate advantage calculations depend on the probabilities of taking certain actions. It is also beneficial to utilize the stochastic nature of the algorithm to encourage explorations. Plus, stochastic networks are generally more robust in foreign environments than deterministic networks [**35**].

CHAPTER 3

System Implementation

3.1. Software Specification

As mentioned in the project scope, a proper simulator is a key part of this research. From the many choices for quadrotor UAV simulators that exist, Gazebo and Airsim were the top two options for most researchers [**7**]. Gazebo is popular in robotics because it provides the entire simulation environment that integrates support for ROS and hardware [**36**]. This makes it an attractive option for robotics researchers that would like to take simulation to real world testing. However, the level of detail for the environment is typically lacking, therefore, it was not the choice for this project. Luckily, the open source Microsoft AirSim provided exactly what was needed.

As a simulator designed specifically for conducting machine learning research and pairing with the powerful Unreal Engine, Airsim was the more appropriate choice [**28**]. Not only does AirSim, seen in figure 3.1, provide a highly realistic physics engine that can operate at a high frequency, it is also easy to integrate with its cross-platform support and myriad of user-friendly application programming interfaces (API).

On the other hand, the Unreal Engine 4 (UE4) is one of the easiest game engines to use that can create high-complexity environments. The level of beauty and realism that UE4 is able to achieve made it so that even popular TV shows, like "The Mandalorian" [**15**], will integrate its photo-realistic graphics. The art of map creation is actually very easy to learn, especially when the editor, as seen in figure 3.2, is extremely user-friendly with many drag-and-drop assets. For these reasons, plus the fact that Airsim has built-in UE4 support, these two became natural choices for high-quality simulation research.

Now, not only is UE4 simple to use, it also comes with a lot of free high-quality content. Most notably, UE4.26 released a water plugin, in figure 3.3, that was built by Epic Games' developers specifically for streamlining the world building process. Before UE4.26, water content had to be

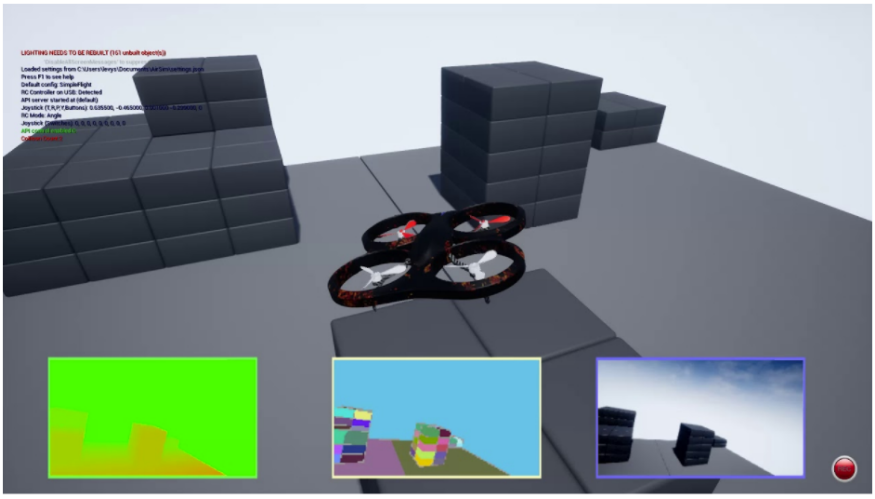

FIGURE 3.1. **Microsoft Airsim in UE4 Environment [23].** The different on-board sensors such as camera, Lidar, infrared, and etc. make Airsim even more attractive than other simulators.

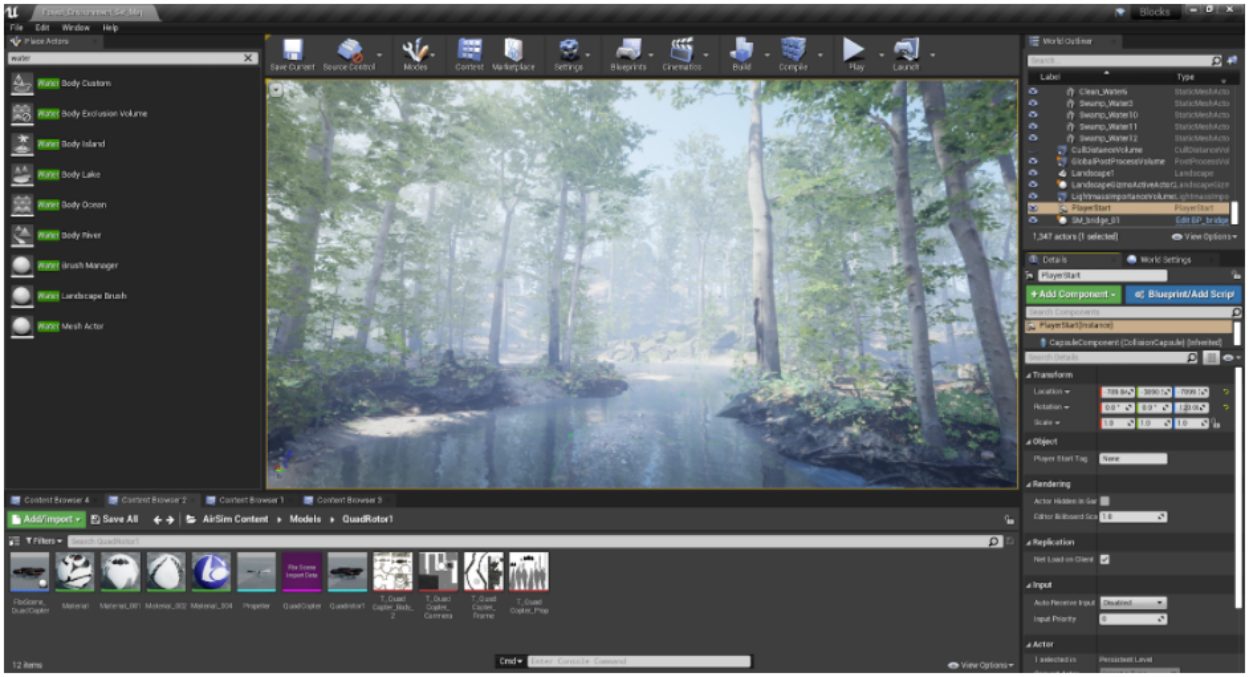

FIGURE 3.2. **UE4 Project Editor Screen.** This is the typical view of the Editor, which contains the actor contents (left), tools (top), world outline (right), rendered view (center), and project contents (bottom). Interacting with assets are as simple as dragging and dropping them into the rendered environment. Thus, it is extremely easy to learn!

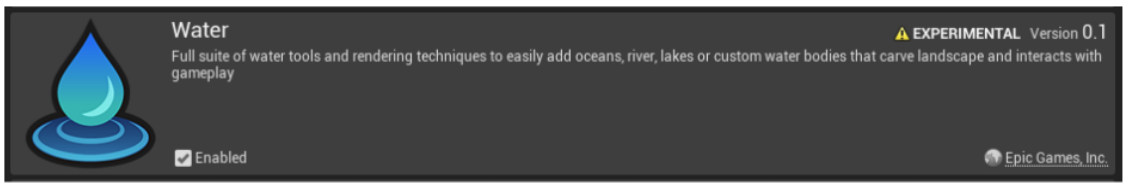

FIGURE 3.3. **Water Plugin Beta from UE4.26.** This plugin contains all the water body contents displayed to the left within figure 3.2. They are experimental content that was developed specifically to streamline the process of adding water-based actors to the environments. Due to this being a beta package, it contains many bugs.

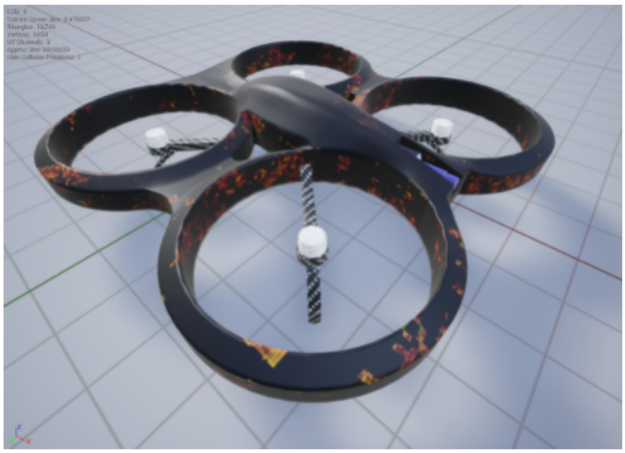

FIGURE 3.4. **1 Meter Wide Racing Drone Model from Airsim.** The Airsim engine includes all the physics component to the racing drone.

added to the content folder independently, and imported into the world as some sort of static actor; And if someone were interested in physics with the water content, they would need to program their own material or pay a hefty price online. Thus, it was lucky that this project started around December 2020 was when the water plugin's beta was released, which provided utilities to build hyper-realistic water bodies that automatically deformed the landscape and carried their own water physics.

Microsoft Airsim 1.3 came with a built-in physics model of a racing UAV, shown in figure 3.4, which made some parts of the project convenient, but not everything was trouble-free. The fact that UE4.26 was incompatible with the old Airsim 1.3, actually created a challenge for environment design; recall the problems of manually creating water bodies discussed above, it would have cost at least double the time to create each map. Thus, it was not until Airsim 1.4 released in January

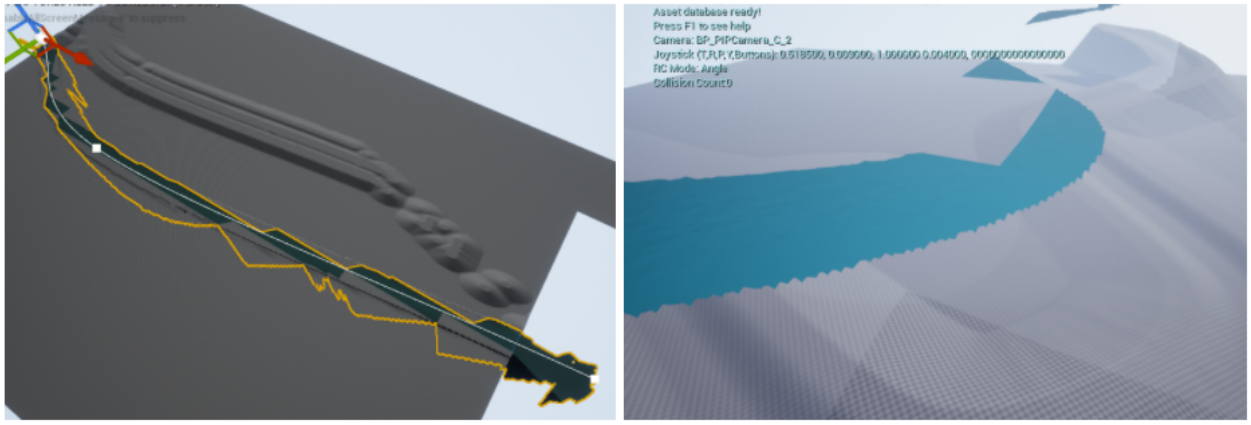

FIGURE 3.5. **Example Glitches with the Water Plugin.** The Water Body River actor will often dislocate from its original location when running in Airsim mode. On the left, one can observe that there is a limit to how large the landmass can be before the river breaks down. On the right, one can observe the river is rendered high up in the sky because the Player actor is not set to (0,0,0). There are plenty more glitches not shown here, and most of them have to do with Airsim mode.

that the project could resume in full force. Yet, even with the release of the new Airsim engine, not all bugs were resolved.

Specifically, the water plugin tends to break very easily when trying to run during Airsim mode. Most common example of bugs was glitching the rendering location of the river itself, as one can observe in figure 3.5. There are many ways to cause this issue, such as not setting the player-actor at 0,0,0 starting location, creating a super large landscape, or attempting to close and reopen the project after making duplicates of the same map. After learning to avoid these issues, many glitch-free environments were created and utilized alongside the Airsim 1.4 simulator.

3.2. Hardware Specification

Since UE4 editor has no Linux support, a Windows machine is required for the creation of the riverine maps. Besides the operation of UE4, there is no other need for the Windows machine. On the other hand, a Linux computer was built with RTX2080 GPU to run the simulations and API's simultaneously. This setup was the bare minimum to achieve 10 hz loop rate after parallel processing the simulation environment, UAV controls, and PPO training. Speaking of which, CUDA acceleration was also used to speed up the training process for the PPO agent.

CUDA acceleration is a commonly used technique that utilizes the GPU to conduct computations at a faster rate than the CPU. Usually, there would be multiple GPU's in a machine learning setup, where at least 1 GPU will solely function to perform computations alone and nothing else. The reason for this is that if one were to run a UE4 simulation, as well as some computations on the same GPU, one will experience a myriad of CUDA errors. In this project, there was a physical limitation of having only 1 available GPU. Therefore, many of the CUDA errors demanded extra attention in resolving.

3.3. Simulation Environments

As mentioned before, the project is an extension from the larger Neptune project, which means some of the work was doubled between the two projects. One large area of overlap is the creation of the simulated environments. One huge motivating factor in choosing riverine environments is because the university is located near Putah Creek, see figure 3.6. That location is where some of the real-life UAV testing took place.

Over the course of the two projects, at least 50+ different hyper-realistic riverine environments were built, although not all of which were used. The variety of these maps were made up by choices such as type of trees, shape of river, color of river, angle of lighting, river bed (scarp) details, river speeds, water formations, and floating objects. However, after much simplification in the original project, only 10 maps were used for the training of the first VAE, and 3 other maps were used for VAE evaluations. The criterion of simplification was consistency among the maps chosen; thus, for these 13 maps, similar models of American Beech trees, light green color river, black cherry shrubs, short grass, dirt scarps, floating leafs and branches, HDRI skydome, and same multi-texture landscape material was used. The only differentiating factor among each river was the shape of the rivers, lighting conditions, and water formations such as water falls or banks. An example of such environments can be figure 3.7 below. As mentioned in the VAE portion of the report, the original VAE was insufficient for this project. Thus, 2 more environments with greater detail in the forest surrounding the river were designed to collect new data of different foliage compositions. Together a total of 15 maps were designed specifically for training the new VAE.

FIGURE 3.6. **Aerial View of Putah Creek [12].** This was part of the motivation for using riverine environments as the scenario for the Neptune project.

For training the PPO agent, 4 different training maps were created following the same criterion above. Note that these rivers are completely foreign to the VAE. Anyway, 2 of the 4 maps are shorter and easier because some early testing like the reward functions needed to be conducted, and it will be more efficient to begin training in easier environments. The other 2 maps are twice as long and more complicated in shape, composition of foliage, and lighting. An image of one of the short training maps is shown in figure 3.8; the key features to notice is that there is only 1 left turn and 1 right turn, the river ends are clearly distinguishable, and each direction of the map is made of completely different states. For these reasons, each direction of the map is classified as an independent river, resulting in a total of 8 rivers worth of states that were used to train the PPO.

After creating these training maps, 2 performance evaluation maps were also created from the same criterion above. These 2 maps are completely different with independent shape, composition of foliage, and lighting; as seen in figure 3.9, the map is more complex than the training maps.

FIGURE 3.7. **Example of Riverine Environment.** Notice the complexity of foliage in differing compositions of grass, weeds, bushes, trees, trunks, scarps, and buoyancy objects. Each map will contain similar foliage species and river color. However, the placement of foliage, shape of river, and lighting conditions will be different. Each map will only contain 1 river body due to avoid glitching the rendering.

These maps are also completely new to the VAE, HG-DAgger, and PPO agent, and should provide the platform a fair comparison of performance.

Since there were so many maps to be made, it was discovered that by following some standard procedures, one can significantly reduce the creation time from 8-10 to 4-5 hours per map. The 9-step procedure listed in table 3.1 below was made by and for amateurs, thus there is still room for improvements. Nevertheless, it should be beneficial for whomever is interested in building high-quality simulation environments.

One other thing to mention about the simulation environment is the way the maps are loaded. Currently, only 1 map can be in communication with the API, and each map only contains 1 river. Although one can technically create a simulation that contains multiple maps each with a unique river, there is no actual way for the API to recognize which map was loaded up at any time during its operation. This limitation also stems from the fact that reward functions are built around some

TABLE 3.1. 9-Step Procedure for Generating Simulation Environment using UE4.

UE4 Editor Procedures	Important User's Notes
1) Open the Airsim "Blocks" UE4 project, and Enable Water Plugin	Be sure to first download and compile the Airsim packages in Microsoft Visual Studio, the "Blocks" simulation will be automatically built. Then, make sure to change the game settings to Airsim mode inside the UE4 editor. If one so chooses, they can also create a blank project from scratch instead.
2) Create a Landscape with Landscape tool	Be sure to enable Edit Layers to allow Water Plugin to automatically deform landscape. Avoid manually adding pieces of land to prevent random glitches from the Water Plugin.
3) Add a Water Body River actor from the Water Plugin, and Shape the river using various spline points	Water color must be tuned using absorptivity and reflectivity inside the material editor, and water roughness also affects the surface's shine.
4) Sculpt the Landscape around the river, and Paint the Landscape using a Multi-layer material	A Multi-texture material is required to paint landscapes of different textures in UE4.
5) Add in scarps around the river bed	This step is the most tedious and repetitive, but it will make the river beds look way more realistic.
6) Add in the following Actors: Player, Skylight, Light Source, HDRI Skydome, Exponential Height Fog, post-process volume, and buoyancy objects	Player actor must be at location (0,0,0) to not glitch the river. Skylight and HDRI doubles as light sources, but is necessary for realism towards the edges of the map. Make sure to tune the height and intensity of Skylight, Light Source, and HDRI to cast proper lighting and shadows. Exponential Height Fog is a component that is automatically rendered if not specified, and thus one must either make it invisible or include it in the game. The Post-Process Volume is used to tune the rendering quality. And finally buoyancy objects are created using Blueprints in UE4 to mimic floating leaves or branches.
7) Save the current project as a template	This step is crucial in saving time! It will usually take about 4+ hours alone just to perform steps 1-6. Step 8 and 9 can often take another 3+ hours.
8) Add Micro-foliage, Medium-foliage, and Macro-foliage using the Foliage tool	These include doing the grass, brushes, water weeds, rocks, overhanging branches, trees, and tree trunks around the river.
9) Reset the lighting, rebuild lighting, and export project	Resetting the lighting is as simple as turning the Skylight on and off. This step is important because sometimes when editing the project, the light rendering becomes oversaturated and shadows become glitched. Rebuilding the lighting is actually not necessary, besides clearing up a warning, once you reset the lighting. Exporting allows the project to be executed as a game environment, which may be required to run some simulations.

FIGURE 3.8. **Aerial View of Short Training Map.** This is the first map that training will be taking place. It has a long straight, 2 turns, and really short length, which makes it perfect for starting out training.

external knowledge about the rivers in each map provided by the designers. For example, the river spline coordinates unique to each map, as well as the different starting locations, target locations, and boundary conditions all have to be manually given to the API. Therefore, it was impossible to load more than 1 map per iteration of training.

On the other hand, the reason there are no more than 1 river per map is because the water plugin will start glitching like the figure 3.5 above. Plus, even if the rivers were glitch free, the boundary functions and reward functions will not work due to overlapping states in the x-axis. Nevertheless, at least each direction of the river is completely different to each other, meaning the agent is at least able to train in 2 very different sets of state space during each phase of training. The agent should still be robust enough, especially compared to prior works where environmental complexity is largely a non-factor.

FIGURE 3.9. **One of the 2 Test Maps used in Performance Evaluations.** The test maps are completely new to the VAE and agents.

3.4. UAV Controls

The UAV model is a 1 meter diameter racing drone provided in the Airsim flight simulator. Since the physics and dynamics of the UAV model is not of interest to this project, only the controls will be discussed. An API called Fastloop was a low-level PID controller developed to maintain the UAV's height and forward velocity. These two values were set constant, eliminating 5 degrees of freedom from consideration, and leaving yaw rate as the only axis of control for the agent. The input to the agent is a 2D RGB observation from the onboard camera.

The only other notable function for controlling the UAV is the hover function. Early in the controls development, it was discovered that the pre-built hover function from Airsim's library is highly flawed, in which the UAV will remain in motion due to its prior momentum. Due to this lingering momentum, if the UAV were to quickly hover and reset to a different location of the map, it would swing wildly out of control. Therefore, a separate hover function that measured the relative velocity of the UAV was created within the Fastloop to minimize this issue. However, this

hover function was still not perfect, in which the UAV will sometimes experience minor drift while hovering.

3.5. Simulation Controls

Besides the Fastloop, another API script called RunTraining was developed to provide capabilities such as image capture, reward management, and DRL training. More specifically, this script contains the embedded systems that connect the input images to the VAE, the VAE latent outputs to the agent, and PPO output to the Fastloop controller.

Note both Fastloop and RunTraining operate in parallel at 10 hz, which is a reasonable rate for reacting to the environment. And the reason why both API's are needed is because the Fastloop must be running at all times to make sure the UAV stays hovering even when the RunTraining is on performing updates.

There are also other functions within RunTraining that handle the interactions of the UAV inside the simulation. These regulatory functions all serve 1 main purpose, which is to reset the simulation so that the agent can move to the next episode.

The first reset function created was for reaching target coordinates. The target reset is as simple as a line from coordinates on the x-axis. However, for this application it may not be as intuitive as other projects. The targets from prior works are usually some easily distinguishable hoop that a UAV can see and fly through; but for a river, it is not as easy to distinguish a finale condition. Therefore, simple allowing the agent to approach the end of the river as the target is a reasonable approach.

Another important reset function is the collision function. If the UAV collides with any foliage, landscape, or assets within the environment, it should reset to a starting location. Sometimes, the UAV will not detect its own collision properly, during which the UAV will just be stuck in place, and will require a stasis reset function for that situation. The reset function operates similarly to the collision function in this case. Reinforcing the necessity of this stasis reset function, sometimes resetting the UAV position within the simulation can cause the Fastloop to lose all control of the UAV model. This is an unexplained glitch between Airsim and UE4 that occasionally occurs. Thus, the stasis reset function is needed to reset the UAV over and over until the API can regain control

again. During this circumstance, there is no way to detect the glitch, so unfortunately all the data gathered prior to the reset must be discarded because RunTraining continues to collect data as if the UAV was not stuck in place. Luckily, this glitch tends to be map specific, so a good way to deal with this stasis behavior is simply to create a new map. Note that some maps are just bound to have this glitch, there is no way of telling which map is somehow cursed.

Besides collision and stasis, if the UAV finds a way to fly out of the map somehow, there is also an out-of-bound reset function for that. However, this was just a fail-safe and was not always necessary due to the following 2 functions.

Sometimes during the training, an agent will learn to fly in circles. It is possible to use rewards to discourage this behavior or cut the episode after a certain maximum timestep. However, the fastest way to deal with this is just to create a function that prevents taking a U-turn. The U-turn reset simply measures the current heading direction of the UAV, and compares it to the global direction of the target. In each map, there are two directions the UAV is allowed to fly, and each direction has an independent target associated with it. The reset function will activate if the forward flying UAV starts heading backwards for more than 10 timesteps.

The last, but most important reset function is the river boundary function. This function actually serves dual purposes; one is to limit the exposure to less familiar states for the VAE, and two is to prevent the UAV from flying out of the map. As discussed in the VAE section, it is impossible to train a perfect VAE, thus it is reasonable to simply impose a boundary around the river to reduce the time spent capturing data of undesirable states. Recall the objective of the project is to fly along the river, thus further exploring the forest state space is unnecessary anyway.

Besides the list of reset functions, there is 1 more function that should be discussed, and that is the update function. There are different ways to perform updates to the PPO network, either episodically or non-episodically. At the beginning, it made sense that the agent should update at the end of each episode, however, this presented a new problem from the parallel access of the 1 GPU.

During the process of resetting to a new episode, the RunTraining API commands the Fastloop API to move the UAV to a starting location, which requires the GPU. At the same time, if an update is being performed, the GPU memory will also be accessed to perform the computations.

And it is precisely due to this overlap in memory access that triggers other CUDA errors to the ones mentioned before. Despite trying different means to separate the timing of the reset and update, similar CUDA errors will still occur despite the effort.

More on this tangent about overlapping usage of the GPU. Recall that the hover mode still causes a bit of drift in the UAV, typically the training is performed when the UAV is in hover mode. If the UAV were to collide with an object from drifting while the training is occurring, it will cause multiple collision signals to be sent from the Fastloop to the RunTraining API until the training is complete. In which, more CUDA errors will then occur and crash the API's.

For these reasons above, the alternative is simple to perform non-episodic updates. Under this architecture, the update will be performed only after reaching a certain number of timesteps. Immediately after this change, most of the CUDA errors were resolved and it became possible to finally perform training, but only using Monte Carlo Advantage estimates.

Finally, although many of the errors associated with the simulations and CUDA had been resolved, there was 1 last hurdle preventing long-duration training. The problem is that the GPU memory tended to overload after running the simulation for roughly 20 minutes. Avoiding this issue required figuring out how to clear the GPU memory, which allowed for prolonged training duration. This solution does not guarantee that the simulation will never crash, in fact random crashes were still observed with no discernible reason.

3.6. Reward Function Design

The reward functions are also part of the RunTraining API, and they are all related to the reset functions. After some testing, it was discovered that they were not all necessary or beneficial to the training itself, and will be discussed in the Experimental Results. The detailing of the reward functions will begin with the 2 most important reward signals, the spline reward and forward reward.

The first and most important reward function was a river tracing reward called the spline reward. Basically, it rewarded the agent for flying close to the center of the river. To build this reward function, the spline of a river was mapped in a graph reader, and broken up in equal spacing to provide coordinate points at each meter-interval in the x-axis. This method was found

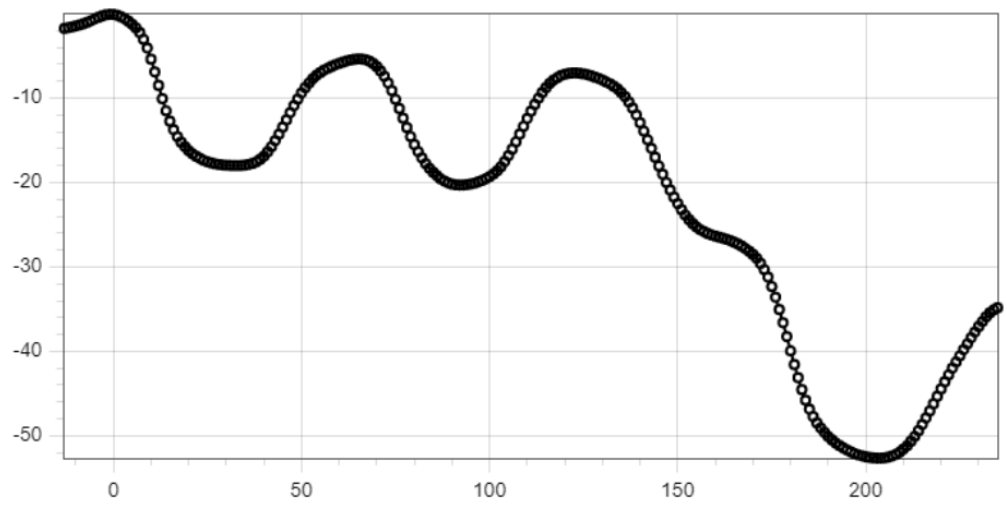

FIGURE 3.10. **Spline Points of Example River obtained using Online Graph Reader.** The points are spaced out in meter increments in the x-axis. These spline points are used to develop the spline reward function, which determines the closest point to the UAV at each timestep.

to be the easiest way to obtain the spline from a UE4 environment, as shown in figure 3.10; other possible methods are Bezier functions or coding a functional Blueprint in UE4. Regardless, it is recommended to use the simplest method found.

After obtaining the spline coordinates of the river, another function that searches the closest spline point to the global UAV position was developed. This is where a bit of nuance was introduced; since the UAV can either be heading forward or backward, it is necessary to actually set the UAV location to either the ceiling or the floor of the current location along the x-axis respectively. That way, the spline reward is also encouraging the UAV to head in the proper direction. Once a matching pair has been found, the absolute distance between the closest spline point and the global position is measured. The output of this measurement will undergo an S-curve treatment to transform it into a reward signal, as seen in figure 3.11. This reward signal intends to communicate that staying closer to the center of the river will provide a higher reward value.

The second reward function that was developed is a forward flight reward, which is a simpler version of the spline reward. During each timestep, if the UAV is traveling in the correct direction and visiting new states along the x-axis, then it will receive a reward of 1. This reward is simple

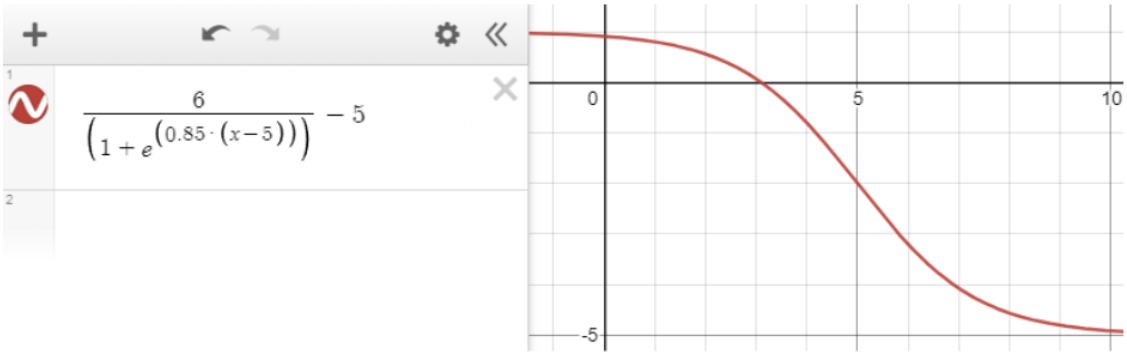

FIGURE 3.11. **S-Curve Spline Reward Function.** The function computes based on the total distance away from the closest spline point. Thus, if the agent is 3 meters distance away from the closet point, it will start to receive negative reward. This should ideally encourage the agent to stay within 3 meters on each side of the center.

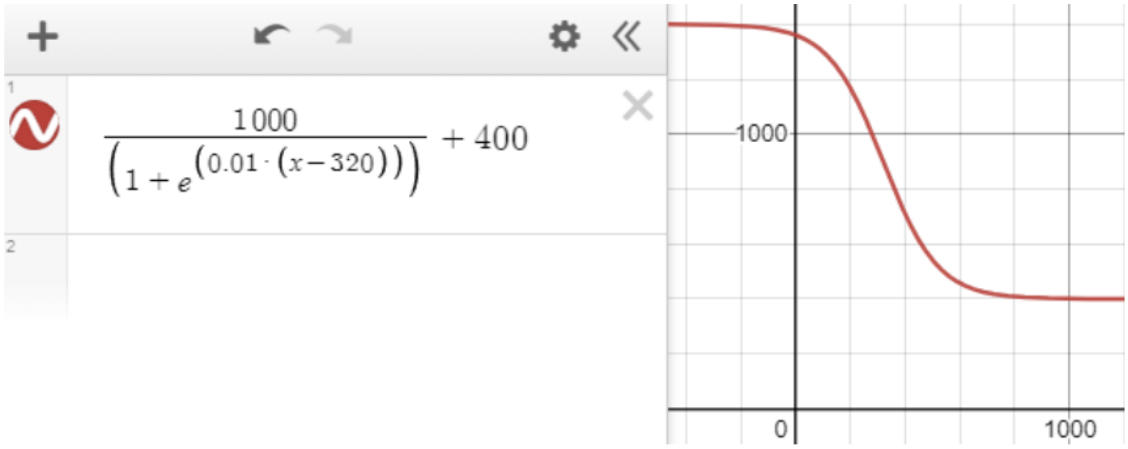

FIGURE 3.12. **S-Curve Target Reward Function.** This function rewards computes the time took to arrive at the target. It should encourage the agent to reach the target sooner. This example is river specific based on the length of the river itself. The standard is that for an average complete episode that last 500 timestep, the target reward should give around 500 points.

and only encourages the agent to fly as far and as long as possible. It does not directly encourage flying along the river and is part of the expectation.

The next set of reward functions are all related to the reset functions, which are meant to either encourage or discourage some sort of behavior. First is the target reward, which is also an S-curve function much like spline reward, in figure 3.12. However, this time the reward is diminished based on how many timesteps it took to get to that target. This time-based reward should encourage the agent to desire the target sooner.

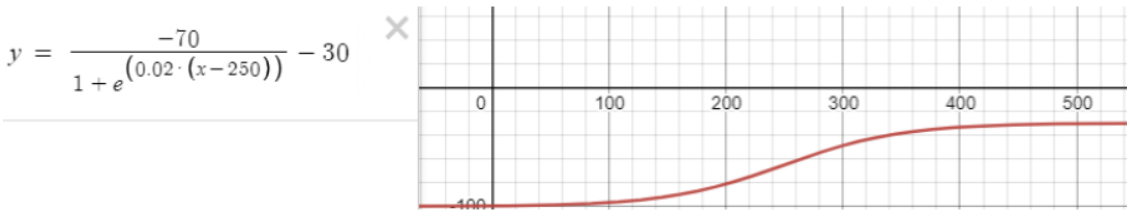

FIGURE 3.13. **S-Curve Collision Reward Function.** This function is universal for cases of collision, U-turns, and flying out of bound. The computation is also based on timing; it rewards more poorly early in the episode to discourage the agent from settling in local minimums. Again, this function is also river specific, in that different lengths matter to the scaling.

On the other hand, the other reset functions also have time-based S-curve reward to discourage colliding, taking U-turns, or flying out of boundary early in the episode, universally represented by figure 3.13. The purpose this time is to avoid creating any local minimums, and disincentivize making mistakes sooner rather than later.

3.7. Experimental Setup

There are many components to this research, simply looking at the available reward functions makes it clear that reinforcement learning is not an easy topic. A way to start is by tackling each objective step by step, meaning looking at the reinforcement learning objective more specifically. In any reinforcement learning, reward tuning is one of the most important challenges, thus it is important to conduct some tests to compare which reward signals are beneficial and which are not. Aside from reward tuning, another challenge of this project is the state space itself. Due to vision being a huge part of this project, it is very important that some experimental efforts are allocated to investigating the VAE.

Since VAE is the first part of the pipeline, it is extremely critical to first test its performance before doing anything else. The Neptune project produced the original VAE model, which the first round of testing will analyze its performance. Knowing that the Neptune project only flew along the river, it captured very little data of the surrounding state space. For that reason, extra data was collected to train a new VAE model for this project specifically. The second round of VAE testing juxtaposed the performance of the original VAE to the new VAE.

The next round of testings was to identify the characteristics of the PPO algorithm collectively dubbed as the pre-training tests. The first minor test was to identify whether continuous action space was actually more effective than discrete action space. In the literature review, it was discussed that there were indeed more benefits to using continuous action space. However, the discrete network was easier to implement than the continuous network. Thus, a test was conducted to find out if future projects can go the easy route.

The second minor test was to discuss the different reward functions. At the beginning, all the reward functions were utilized to try to encourage the agent to fly along the river until reaching the target, while avoiding collisions. The complicated reward signal includes the spline reward, target reward, collision reward, U-turn reward, and river boundary reward functions. However, this was not the only reward signal tested due to fear of complications. Simpler reward signals using only the forward reward or the spline reward were also tested to identify which reward signal will ultimately be used to train the PPO.

After completing the 2 pre-training tests, the next test was to learn to properly tune the hyperparameters. Since there are very few documentations on similar research, not to mention they also hardly dive into the hyperparameters used, an empirical study to see which hyperparameters would yield the best results was conducted. Some notable hyperparameters that were tested include timestep, epoch, learning rate, and standard deviation. Not every hyperparameter listed will be discussed in the results since most of them are typical to all DRL algorithms; and since PPO is robust to hyperparameters, there is little more to learn.

The culmination of results from the pre-training tests, plus hyperparameter tuning will yield attributes of the final agent. And with those conditions identified, the multiphase training can begin. First, the agent will be trained in the training map only flying forward; at the same time the agent will also need to learn the left turn, as seen in figure 3.14. Once the agent is proficient in making left turns, the agent will be moved farther back and begin learning to turn right as well. After the agent is competent at making both turns, it will need to start flying in both directions of the map in randomized order.

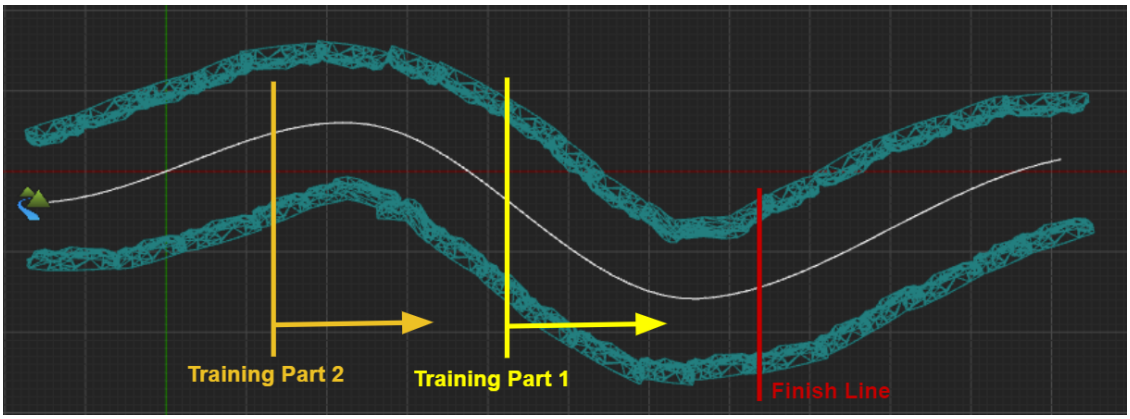

FIGURE 3.14. **First Training Map Setup.** The line in the center is the spline of the river, while the green edges are the scarps of the riverbed. Foliage is not represented in this view as it is not meaningful to display them. Note that each grid is 10x10 meters. There are 2 starting points for this map, which are used to break down the initial training into 2 simple parts. The yellow line is where the agent will be placed at the start of training; it will only need to learn to fly forward and make a small turn before finding the target. The orange line is where the agent will start from once it has converged in training part 1. Note the agent will always face in the tangential to the spline at the beginning of each episode simply to maintain consistency. The Finish line is kept farther from the edge of the map to reduce the impact of overexposing the cameras.

After learning to navigate both directions of the first map, the agent will move onto training in the second map. And subsequently, the agent will then train in a similar manner in the last 2 maps as well. Again, the first 2 maps are shorter and less complicated than the last 2 maps.

After training the PPO agent, the HG-DAgger policy was also trained in the same 4 maps. This time, the HG-DAgger agent will be provided with a relative number of samples that were scaled from the number of samples used to train the PPO agent. A comparable ratio of 250:1 was used to scale the number of samples for HG-DAgger agent. This ratio was synthesized because it only took 2 episodes for the HG-DAgger agent to achieve the same performance as the PPO agent in the first river. Thus, for a total of 4020 episodes of PPO training samples to learn 8 rivers, a total of 16 samples were used to train the HG-DAgger policy.

With the baseline policy well trained, comparison between the two final policies was conducted. The performance tests are conducted in 2 different maps with a 4:1 training map to testing map ratio. The analysis will investigate choke points that present issues for the agents. Metrics such as average elapsed time per episode and maximum distance traveled per episode were investigated.

Note that these two comparisons are averaged between the two maps, plus the episode length depends heavily on the individual map. Therefore these two metrics are relative, and do not speak absolutely of the performance. The last 2 metrics include average distance from the center of the river and average standard deviation of actions taken were also measured. Again, these metrics are averaged among the two test maps. All of the metrics were validated under a 5% significance level.

CHAPTER 4

Experimental Results

4.1. VAE Analysis

A well performing VAE for this project needs to be able to compress visual inputs and produce a meaningful latent representation that can capture enough information about the environment. The images in figure 4.1 shows what a good VAE should do. Although, these are simply the reconstructions, which does not always reflect the quality of the latent states. It is at least a reference one can use to judge the performance of the VAE.

The first part of this discussion will analyze the original Neptune VAE's performance. One big issue identified is that foliage hanging from the sides are usually blind to the VAE. This first test, figure 4.2, shows the VAE completely removing the foliage from its perception. This is the most common way the VAE deals with this unfamiliar state such as overhanging foliage. Other times,

FIGURE 4.1. **Examples Reconstructions of Well-trained VAE.** This is a reference for what a good VAE reconstruction will look like. It should provide a qualitative analysis on the latent representations.

FIGURE 4.2. **Neptune VAE - Overhanging Foliage Test.** The old VAE cannot tell that there are foliage overhanging above the river.

FIGURE 4.3. **Neptune VAE - Complex Front Foliage Test.** The complexity of front foliage, typically around corners, seems to challenge the old VAE in thinking there is a river directly in the bushes. The orange box indicates where the VAE interpreted as the river path itself.

the VAE might do what is seen in figure 4.3. The largest concern here is also the fact that the center could be mistaken as the river as well.

Another big issue of the VAE is that it does not recognize tree trunks. It will often treat it as a complete fuzz, and sometimes, it will instead treat the tree trunk as a piece of the river shown in figure 4.4. This is an issue because if the VAE recognizes some tree trunks as the river, it will make the DRL training slightly more difficult.

FIGURE 4.4. **Neptune VAE - Tree Trunk Test.** The old VAE has trouble understanding tree trunks due to a lack of samples. The worrying thing was that it treated the tree trunks as a river path, much like the overhanging foliage case. An orange box is drawn to show where the VAE mistaken as the river.

FIGURE 4.5. **Neptune VAE - River Bank Test.** Bright patches of grass and brightly lit scarps along the river bank also seems to challenge the VAE. One can clearly observe a river in the reconstruction.

Finally, though there are other issues with the VAE, the last important one to highlight is the treatment of light patches of grass. From the figure 4.5, grass from the river bank was unfortunately mistaken for the river itself, and there is no need to explain why this is an issue for any autonomous agent.

For the reasons above, it was decided that a mild retrain of the VAE will need to be completed for ensuring good DRL training. After incorporating new samples from flying into the forest, up

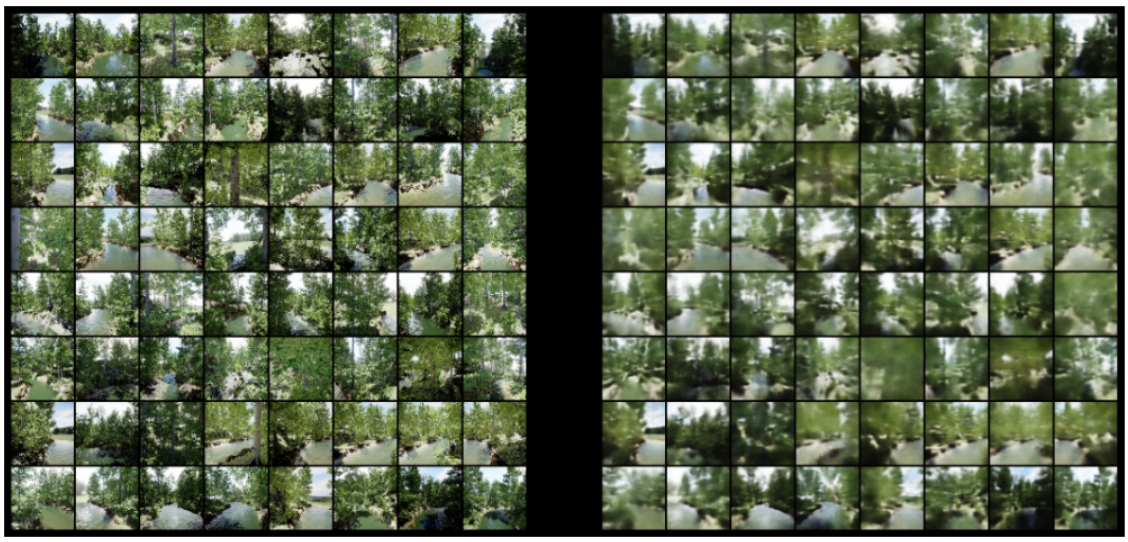

FIGURE 4.6. **New VAE Reconstruction Validation.** This is the performance of the newly trained VAE.

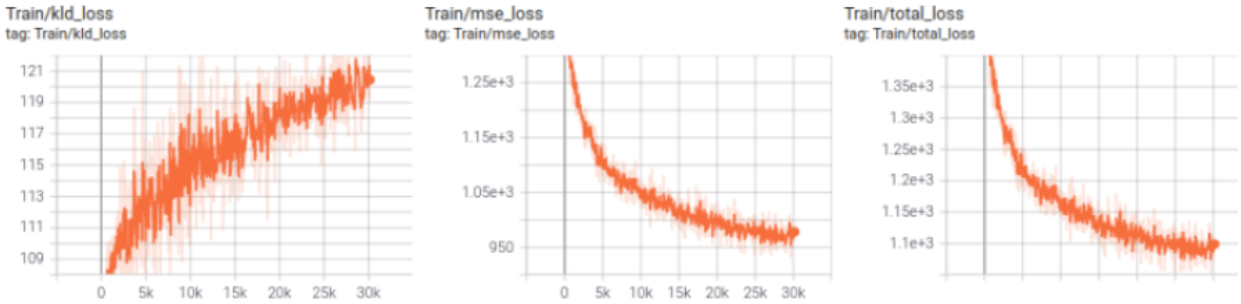

FIGURE 4.7. **New VAE Training Progression.** The new VAE was trained with bonus samples, and the losses at 50 epoch have converged well.

close and into foliage, and under different lighting conditions, some improvements are made to the VAE.

First, the new VAE's general performance can be seen in the reconstruction sample figure 4.6. The figure 4.7 shows a total of 50 epochs was used to train this VAE.

Tree trunks are now somewhat recognized by the new VAE. It will often be treated as a faintly blue hue, mostly a mixture of grey to brown tree trunks, seen in figure 4.8.

Complex frontal foliage can now be somewhat differentiated from the river itself; the box in figure 4.9 indicates a better match for the river body than the figure 4.3 from the original VAE. Thus, the new VAE appears to have a higher probability of recognizing the real river.

FIGURE 4.8. **New VAE - Tree Trunk Test.** The same tree trunk sample was fed into the new VAE. One can observe that the trunk is at least somewhat recognized now. It should no longer interpret it for river states.

FIGURE 4.9. **New VAE - Complex Front Foliage Test.** The same complex sample was used, and one can observe from the orange box that the reconstruction recognized where the river was truly at. Compared to before, this new improvement will have a lower chance of confusing the agent.

There are still challenges with overhanging foliage, unfortunately demonstrated in figure 4.10, most likely due to a lack of samples for overhangs. These images are second most difficult to capture, therefore there is a lack of expert samples here.

Speaking of which, the most difficult samples to capture are light patches of grass; therefore, that issue remains unresolved. To limit the exposure to bad VAE states, artificial boundaries are placed around the rivers to stop the episodes if the agent flies out of the river. This preventive

FIGURE 4.10. **New VAE - Overhanging Foliage Test.** Some foliage blending in with the river can still occur. However, in vision-based learning, and one must expect this type of potentiality.

measure is important since states within the forest can be mistaken as the river due to bright patches of grass and complex foliage. Regardless, the new VAE has improved recognition of complex foliage structures and tree trunks, it will go on to perform well in novel environments.

4.2. Pre-training Tests

The first pre-training test performed was to determine whether a discrete or continuous action space will be used for the PPO agent. The training was performed with the forward reward signal, and the evaluations found similar behavior described by the comparisons performed by a different team [**37**]. The discrete agent had large fluctuating outputs at each timestep and performed relatively shaky, whereas the continuous agent had mostly smoother outputs leading to more stable performance. This result is despite flying straight down the first river; therefore, it was decided that this project will use a continuous action space model.

The second pre-training test performed was to identify an effective reward function that can encourage the desired behaviors. Originally, the complicated reward signal was supposed to communicate to the agent that crashing into foliage is bad, flying away from the river is bad, taking U-turns were bad, staying along the river is good, and reaching the end of a river is good. However, none of the many PPO configurations were able to train under this complicated reward function. The reason for this is because for pure vision-based learning in natural environment, it is can be

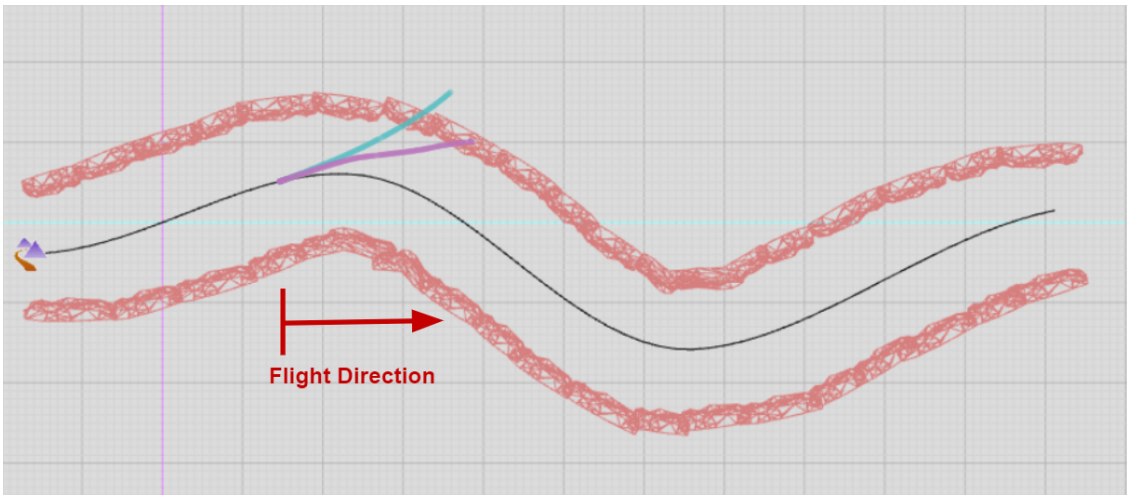

FIGURE 4.11. **Forward Reward Agent Evaluation.** Again the center line is the spline of the river, while the pink edges are the scarps around the riverbed. The colored lines are 2 example trajectories this agent took. The agent collided or flew out of bound in these two demonstrations. The forward reward signal failed to train the agent to fly along the river. The agent was only able to make left turns; and once it had to start farther back, it failed to make a right turn.

difficult to identify certain conditions. For example, how to communicate whether the agent had reached the target. It is perhaps not even necessary to do so in an obstacle avoidance problem. Therefore, following examples from prior works simply does not work for this project. The end result is that the complex reward signal degraded the learned representation of the environment. Resulting in degrading performance over time.

Since the goal was to train the agent to fly along the river for as long as possible, the forward reward, a simple positive reward each timestep may be enough to encourage this behavior. The rationale was that if the UAV crashes or flies away from the river, it will receive less reward than if it had not crashed. Therefore, the agent should be properly motivated to fly forward while avoiding crashing to maximize rewards. However, what was discovered was that this reward function does not do well to encourage the agent to stay along the river. It somehow causes the agent to take the shortest path forward, flying into the forest if it has to, as seen in the figure 4.11. The important thing to note is that this agent also trained from the first starting point in figure 3.14, but it was when it had to start from the second starting point that it began to exhibit this left-veering behavior without being able to find a way out. Moreover, recall that the forest states may look

similar to the river states, which can cause issues for the agent. Thus, it was found to be stuck in some local minimum and only knew how to take left turns.

Based on this result, a different reward function was needed to train the PPO agent. Again, the spline reward function encourages the agent to travel towards the next closest spline point in the target direction. This reward signal was successful in training the same test agent to complete the river without collisions. The same test agent in this case only refers to having the same hyperparameters. Thus, the spline reward function was the one chosen for all the training.

4.3. Hyperparameter Tuning

There are many hyperparameters that are important to the algorithm, however, many of them are fairly standard and without anything new to contribute. Parameters such as epochs, learning rate, ϵ, and standard deviation all mainly affect the rate of convergence; within this project, they all demonstrated expected behaviors. These results are expected since PPO's bound ensures that incremental updates are always in the direction of the advantage. Thus, regardless of how large or small some of these hyperparameters are, they will not have a large effect on the convergence, hence PPO's robustness to hyperparameters. On the other hand, the hyperparameter of timestep turned out to have a major impact on the learning process.

The timestep defines the size of the experience trajectory, and turns out tuning the value of this hyperparameter had the largest effect on convergence rate, success of learning, and stability of training. It was discovered that a static timestep was actually detrimental for training the agent in maps of different length; this was most likely due to the fact that the ends of the river had very different states to the body of the river. Therefore, if the timestep was too small, sometimes the agent will train with trajectories that did not experience the full variety of states, and end up having degrading performance when revisiting those states. As a consequence, the agent will also have a less stable learning process overall, as seen in figure 4.12.

On the other hand, if the timestep of the trajectory was too large, the agent will require a very long training process. If the agent visits an unfamiliar state and performs badly in those states, it will usually require lots of time for the agent to learn how to avoid those states with a long experience trajectory. Sometimes, the agent will just get stuck at a local minimum and visit poor

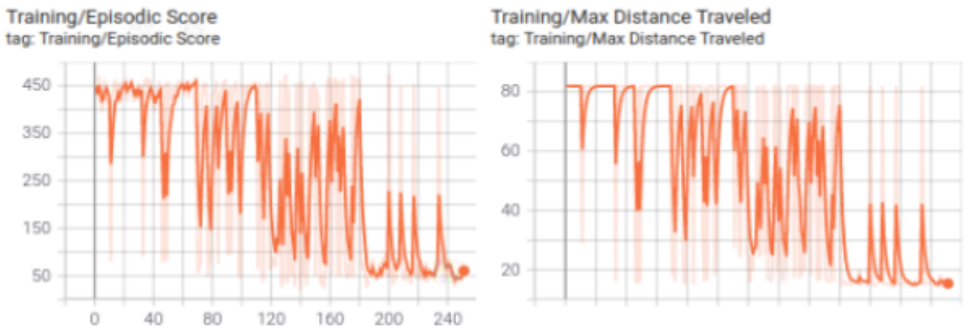

FIGURE 4.12. **Training Results of Using a Very Small Timestep.** The consequence of using a timestep hyperparameter that is too small to adequately capture at least an entire episode will result in degrading performance. This problem occurs because it is necessary to capture all the nuances of an episode for each experience trajectory. Otherwise, the agent will experience distribution shift. Note that similar degrading performance can still happen if the timestep was too large instead. That problem occurs when the agent performs a poor update and begins to obtain an overwhelm distribution of bad data.

states over and over; this can happen when the experience trajectory does not have a majority distribution of good performing states. Conceptually speaking, due to PPO being an on-policy algorithm, it must train on the most recent samples. If the samples were predominantly poor, then there's a distribution bias towards those states, and will be a challenge to recover from those states. Thus, since small timesteps will cause degrading performance and large timesteps will cause convergence issues, the best practice was found to scale the timestep to the length of the river itself.

For shorter training maps that are usually 80 meters in length, the best timestep value was found to be 1000. This value allowed at least 1 full fly-through experience each iteration of training with simulation running at 10 hz. It is important that each iteration observes the river ends, but should have predominant experience flying within the river body. Using this timestep, then training in the same map was now successful, as seen in figure 4.13.

The training was a lot more stable compared to before, and definitely showed signs of improvement when visiting new states instead of degradation. As for longer maps that are double in length, the timestep chosen was also doubled as well. Although, maintaining the same timestep would have been ideal for maintaining consistency, it was found that the original timestep was too small for the longer maps; thus, the best practice was to scale this hyperparameter based on river length. There

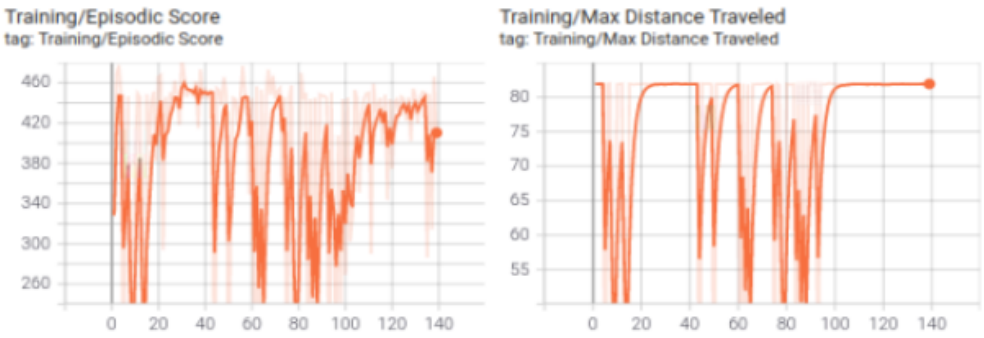

FIGURE 4.13. **Training Results of Using a Correctly Sized Timestep.** Scaling the timestep such that it will capture between 1-2 episodes of experience was found to have the best training results. It lead to more stable learning and better convergence.

were also the considerations to scale other hyperparameters to the river length as well, however tests found them to have negligible impact, especially when compared to the timestep's effect on training.

4.4. Multiphase Training Analysis

The training was conducted in multiple phases similar to curriculum learning, but not strictly adhering to its concepts. Simply, the project trains the agent in increasingly difficult environments based on length of river, complexity of foliage, and lighting. The initial training was conducted from the middle of the first river, from figure 3.14, flying in the positive x-direction. It was decided the agent should collect some good rewards simply flying along the river before making turns. So starting on the straight should be beneficial to its initial training.

The agent will only need to make a tiny left turn to reach the end of the river in each episode. The training result from this PPO agent was recorded below in figure 4.14. Since the agent operated in the continuous action space, it simply traveled forward without applying much input in the beginning; thus, it obtained relatively good rewards even just flying forward along the first sector. As it begins to learn to make tighter turns it obtains higher rewards, but only until it crashes on the left side of the river. After it observes those new states on the left, it corrects its policy to maintain a safer distance and starts obtaining better rewards again. After converging for the left turn, the agent was moved farther away from the target and now has to perform right turns as well.

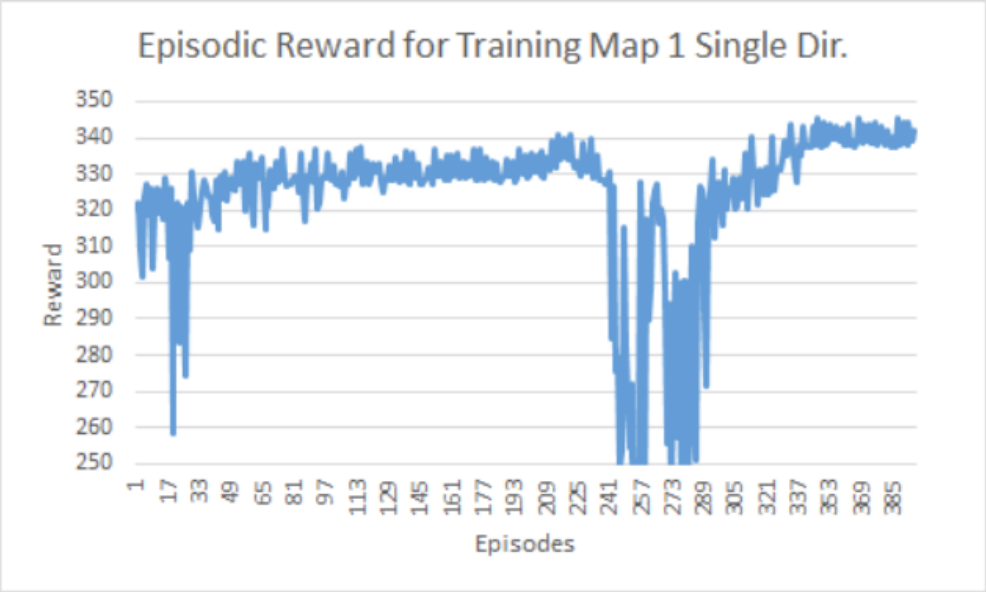

FIGURE 4.14. **Training Results of First Map in only 1 Direction.** This is the first of all training for the PPO agent. It starts from the midpoint of the river as discussed before.

It took the agent some time to explore the new states and figure out how to optimize its policy to do both turns. After training on the forward (positive-x) direction of the map, the backward direction was also added to the training as well. To make this policy more robust, both directions were trained alternatively and randomly with the results displayed in figure 4.15. Notice that the performance at the beginning seemed to fluctuate rapidly. This is due to having to explore both directions at the same time. Again, because this is a vision-based learning problem, it is difficult to have a more steady learning process. It took roughly 1879 episodes for the agent to reach convergence on completing 1 direction of the map with consistent success. On the other hand, it only took the HG-DAgger agent only 1-2 episodes to reach convergence with similar experience. Hence, the ratio of PPO-to-HG-DAgger sample ratio is 250:1. Anyway, it was time to move on to a different map to continue training.

The second map was slightly more complicated than the first river, with a slightly different shape, lighting, and foliage. The policy, at this point, was able to directly train both directions of the map at once; and it took about 793 episodes to reach stable performance as seen in figure 4.16. Notice again that the performance at the beginning fluctuated rapidly. This time, not only

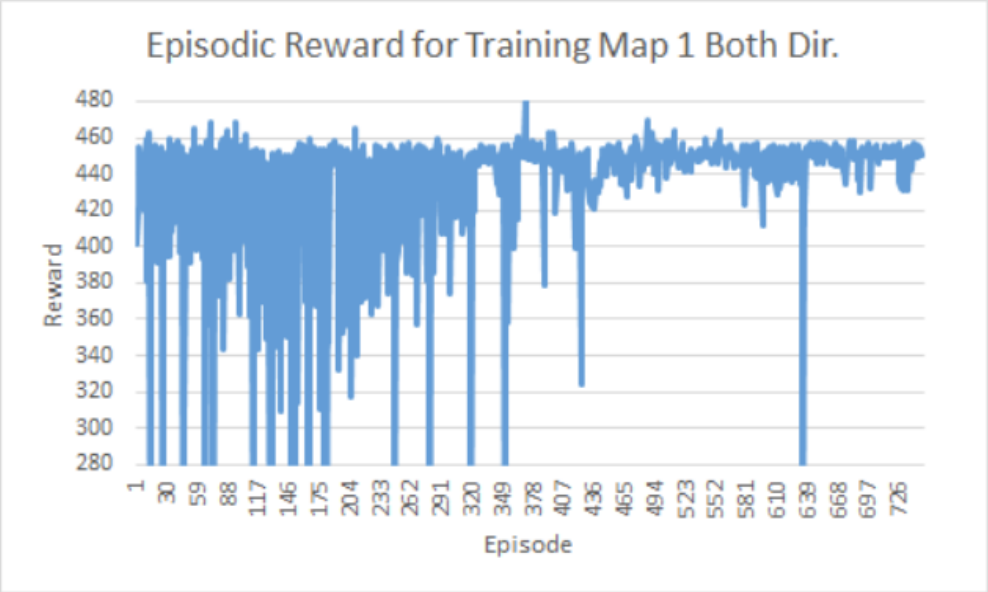

FIGURE 4.15. **Training Results of First Map in Both Directions.** The target direction of the training is randomized. This is the third part of training that took place. There are a bit more fluctuations in performance at the beginning because the agent need to learn both directions at the same time.

does the agent have to learn from both directions of the map, but the map itself is completely new. Thus, the rewards remained rocky for more episodes than the first map.

The final phase of training took place in 2 different maps, each were double in length to the first 2 maps with highly peculiar shapes, different lighting conditions, and greater foliage complexity. The agent was still able to train in both directions of each map at the same time and with less training samples required. It took roughly 291 episodes to train the first long river and 309 episodes to train the second long river, of which one is shown below in figure 4.17 .

What was discovered was that it will take a decreasing amount of samples to train a policy that is already well adapted to a similar task. The cutoff points are rather well defined because clear evidence of actor drift is apparent after training for a long time. For example, the performance will begin to degrade after a certain number of episodes due to overfitting, and that is where the training has passed its peak performance.

There is one other issue, however, which is getting stuck in local minimum after poor updates. When this occurs, it may be easy to misconstrue an earlier checkpoint as the finished agent.

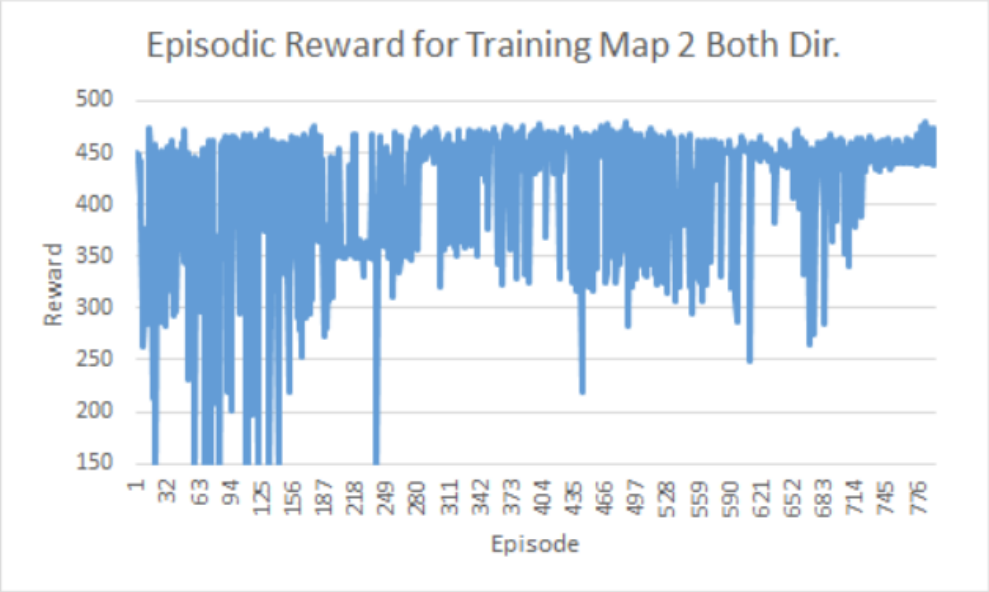

FIGURE 4.16. **Training Results of Second Map in Both Directions.** This is the fourth part of training, where the agent will need to train in both directions of the map at the same time. Once again there are fluctuations in performance at the beginning because it is not only having to train in both directions, but also train in a completely new map.

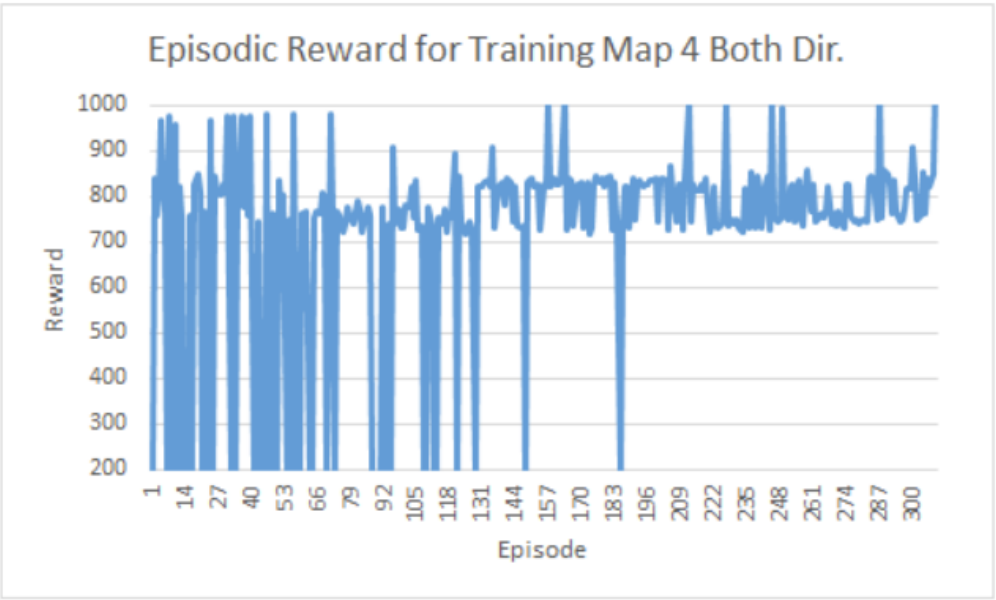

FIGURE 4.17. **Training Results of the Last Map in Both Directions.** This is the final phase of the training process. There is nothing new in the procedures leading up to this point. The rewards also communicated the same insights from before.

Therefore, restarting training from that earlier checkpoint a couple more times was required to confirm whether the policy had actually converged to its peak performance.

Now, one may assume the decrease in sample requirement was only due to the rivers being twice as long, and therefore requiring half the number of episodes. However, recall that the experience trajectory was also doubled in length, thus there are no contradictions to the observation made.

Despite changing the length and shape of the river, lighting conditions, and complexity of foliage, the PPO training clearly benefited from the generalization of the VAE. It is most likely impossible to obtain similar results without some form of state reduction or feature extraction element in the loop if the task complexity were to increase.

Finally, to touch base on imitation learning agent again. It took 4020 episodes to fully train the PPO agent. And using the comparable ratio of 250:1, the HG-DAgger will need a total of 16 samples total to gain a fair comparison. After training the HG-DAgger agent, the performances in training maps were successful just like the PPO agent. Therefore, the HG-DAgger agent is ready to be tested in the performance test against the PPO agent.

4.5. Performance Test

To reiterate, the baseline HG-DAgger agent was trained using the same 4 maps as the PPO agent. Due to the relative training process, it should be a fair comparison to observe whether PPO and HG-DAgger had the same performance. Based on a 4:1 ratio, training maps to testing maps, a total of 2 rivers were used to examine the null hypothesis. Each algorithm ran a total of 10 trials per river for the P-value test. Note for both rivers, the agent starts from the left side and attempts to fly to the end of the right side. A deeper dive into the performance will be discussed as a collective after first viewing some trajectory graphics.

For test map 1, the performance of PPO and HG-DAgger are shown in the figure 4.18; note the vertical-axis is actually flipped. Towards the end of the river, it can be seen that there was a specific choke point where both algorithms found difficulty navigating. It can be seen, in figure 4.19, that this was due to a bright patch of bushes nestled between some dimmer tree branches. This situation is challenging for the agents because it looks similar to the river. However, it would still appear that since PPO was closer to the center of the river, it was able to recognize and avoid

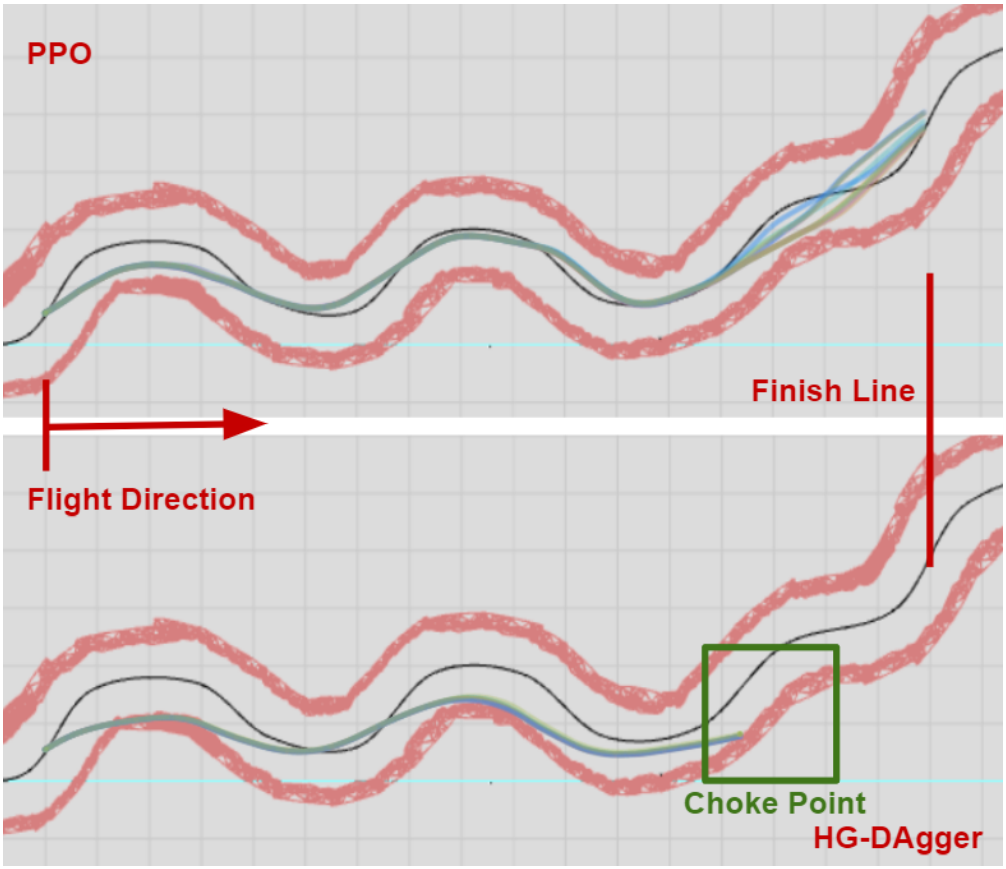

FIGURE 4.18. **Test Map 1 Performance.** A total of 10 trials were performed for each agent. PPO was successful in navigating this environment, while HG-DAgger witnessed collisions at the choke point. Notice that this choke point actually affected both agents.

the turn from a slightly different entry angle. This may have been the result of training with a reward function that encouraged staying close to the center of the river. If that is indeed the case, then that would be a benefit to using DRL. Just note that although only 1 choke hold had been shown here in test map 1, there were many similar locations that both the agents were able to successfully avoid.

Moving on to the performance on the 2nd test map, which can be seen in figure 4.20. This time there appears to be a couple choke points in the map, both of which have caused confusion for the algorithms. The first choke, in figure 4.21, appears to only affect the PPO agent, where 4/10 trials had caused a deviation from the rest of the trials. This first choke is an example of a choke that the agents were able to eventually identify as a dead end after approaching closely. Perhaps it is

FIGURE 4.19. **Test Map 1 Choke Point.** This is the first location that actually affected both agent's performance. The suspect is that there was a very bright patch of bush, boxed in yellow, that highly contrasted its surrounding trees. This challenged the agents to distinguish whether it was the correct path to take. The agents also took slightly different angle of entry towards this choke point, the PPO agent was able to tell the correct path, while the HG-DAgger simply went straight.

due to there being a tree directly above the bright shrub that allowed the VAE to correct its guess within close proximity.

The second choke point, in figure 4.22, caused collisions for 3/10 trials of the PPO agent, and also for the HG-DAgger agent. There are clear similarities between this choke point and the one from the first test map. What was interesting is that of the 4 PPO trials that were headed towards this choke point directly, only 3 of them were collisions. This was due to the slight changes in angle when facing the choke point, which allowed 1 of those 4 trials to recognize that it was actually just foliage and make a recovery. Nevertheless, it can definitely be seen that the angle of entry matters a lot in vision-based scenarios.

The collective performance in both test maps are shown in the figure 4.23 and 4.24, where the records show how long the episode elapsed, maximum x-distance traveled, average distance from the center line, and standard deviations of actions taken. The P-test shows that the results

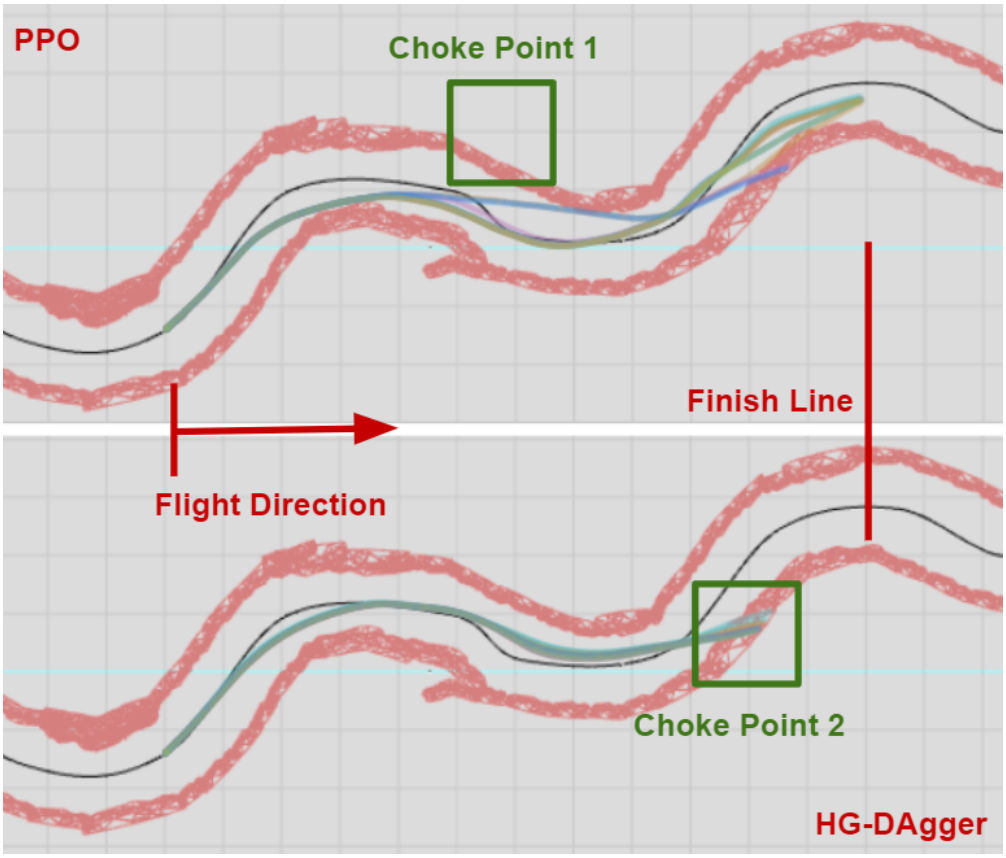

FIGURE 4.20. **Test Map 2 Performance.** A total of 10 trials were performed for each agent. PPO was largely successful in navigating this environment, while HG-DAgger witnessed collisions at the second choke point. Choke point 1 seemed to only affect PPO agent, while the HG-DAgger was able to avoid its influence. Meanwhile, choke point 2 caused collisions for both agents.

obtained are statistically significant, therefore the null hypothesis of PPO and HG-DAgger being the same can be rejected.

The first observation is concurrent with the graphics from above where the PPO agent was able to fly an average of 21.10% farther and 24.82% longer in the same riverine environments. This results shows that PPO agent was more robust to distribution shift than HG-DAgger, which is not surprising since DRL allows the agent to explore a larger state space.

The second observation is that the actions taken by the PPO agent appear to be more aggressive than the HG-DAgger policy. Since both agents trained on the same maps, this observation was not only the result of there being bias on the expert's performance, but also that the PPO agent had

FIGURE 4.21. **Test Map 2 Choke Point 1.** This location is once again defined by the contrast between the bright shrubs to the shady trees. The difference this time is that there's a tree trunk directly behind the shrubs. That may have been the reason why the PPO agent ultimately recognized it apart from the river and steered away from the choke.

more exploration of the state space. Furthermore, training with a reward function that rewarded staying close to the center of the river may have caused the PPO agent to fly closer to the center on average of 58.49%. Subsequently, it was no surprise that the standard deviation of actions were differed by 57.35% because PPO needed to take more aggressive turns in order to stay close to the center of the river.

Finally, the results obtained in this project would not have been possible without the VAE. Despite having some shortcomings still, the VAE still improved the learned representations of the agents. Without this state reduction system, there would have been no way to achieve these results solely relying on vision-based observations. Thus, the trade-off worth considering for future research is really which state reduction models to use. Nevertheless, in this experiment, it was discovered that PPO was significantly different from HG-DAgger after training and performing in the same environments.

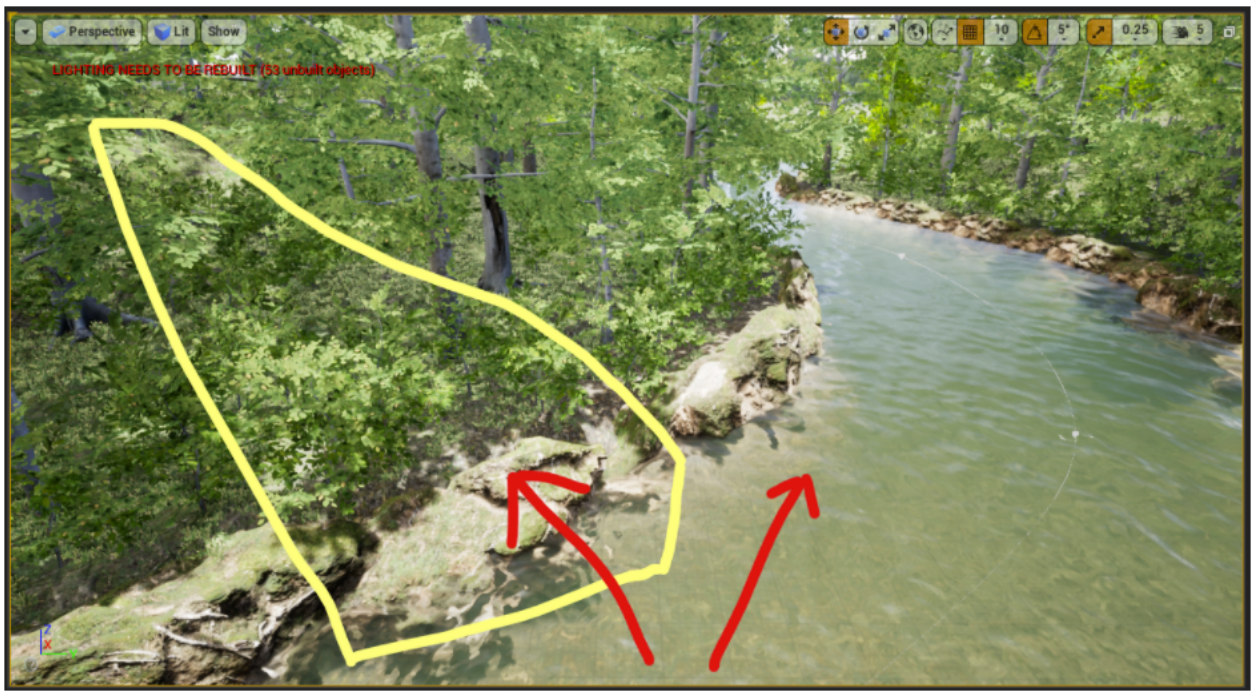

FIGURE 4.22. **Test Map 2 Choke Point 2.** This choke had a long streak of bright foliage as seen in the yellow box. It did not have a tree trunk like the first choke, and was a lot challenging to identify for the agent. Nevertheless, the angle of entry may have had a strong influence on the reaction to this choke point.

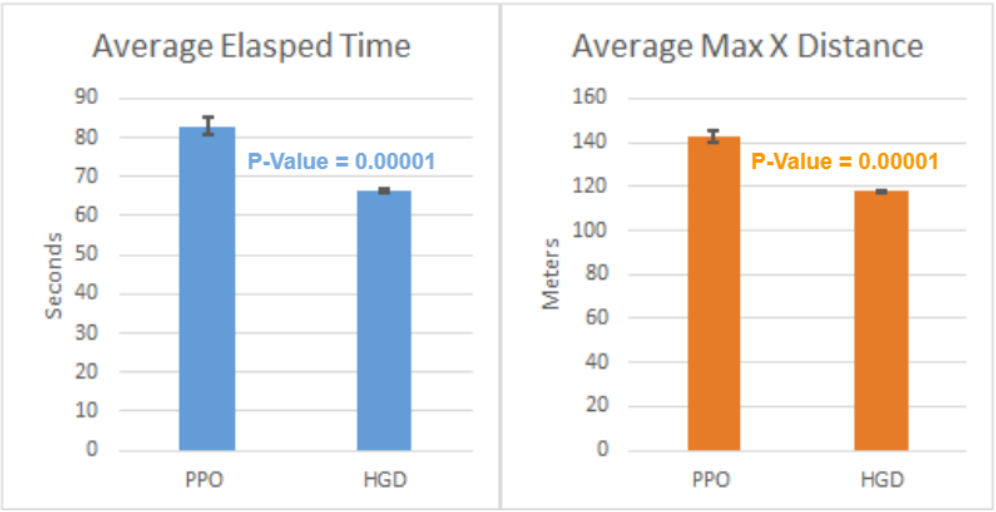

FIGURE 4.23. **Average Episodic Elapsed Time and Episodic Distance Traveled.** The results show mean and standard deviation of all 20 trials. The results are statistically significant with P-Values of 0.00001. Note that these are average episodic values, which are unique to these 2 test rivers.

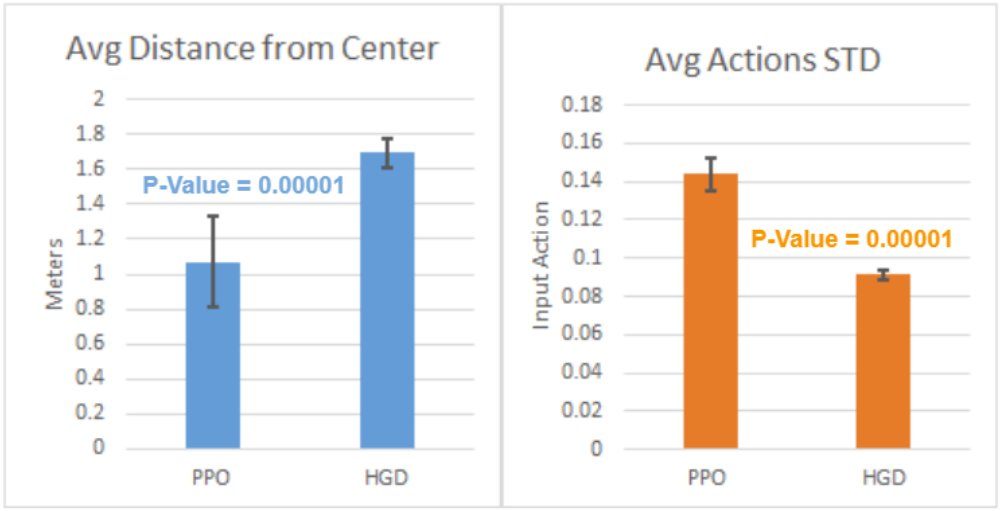

FIGURE 4.24. **Average Distance Away from River Center and Actions Standard Deviation.** The distance away from the center can tell the story of whether the agents were tracing the river or simply flying along. The action STD can determine how aggressive and twitchy the agents behave. The results are also statistically significant with P-Values of 0.00001. These results a more generalized and less unique to specific rivers.

CHAPTER 5

Conclusions

5.1. Summary

In retrospect, the project tested whether DRL is necessarily different to IL in navigating complex synthetic environments, and found it to be significant. The project's first contribution is to use DRL in a novel application that aimed to elevate the complexity and realism. This project also tackled the challenging task of vision-based obstacle avoidance in open synthetic environments, and have found ways to improve sample efficiency and quality of learned representations. The second contribution of this project is to develop a vast amounts of photo-realistic simulations that other students can use for their DRL projects. Over the course of this work, over 50+ high quality maps were developed and shared.

The third contribution of this project is to investigate the challenging task of reward shaping in vision-based learning. It was discovered that for operating in natural-looking environments, complex reward signals can have degrading effects on the agents. This project also analyzed the most important hyperparameter in training on-policy algorithms. Properly scaling the experience trajectory is critical to obtaining proper convergence. The multiphase training process based on increasing levels of difficulty showed benefits of sample efficiency and convergence stability.

The last contribution of this project is to compare PPO with HG-DAgger agent in novel environments after conducting comparable training. There was significant improvement in performance for PPO agent in new environments. PPO showed a higher degree of robustness towards distribution shift as well as validating the benefits of proper reward shaping. The end result of this novel project is a UAV policy capable of performing vision-based obstacle avoidance in complex riverine environments.

5.2. Future Works

The main objective of future research is to transfer the current works to real-life applications. A big concern for this is safety, which is heavily impacted by the performance of the VAE. The importance of good observations cannot be understated, and this project truly unveiled the limitations of the VAE. The findings discovered that complicated reward signals cannot be used if the observations are too noisy. Therefore, a recommendation for future works is to create a more robust state reduction system either with segmentation models [**27**] [**39**], VAE+GAN [**20**], or CMVAE with other observations mixed in [**10**].

Besides dealing with the state reduction issues, there's always the problem of distribution shift. As discussed before, it is possible to train that same system to judge whether new observations are sufficiently different from the training set [**3**] [**16**]. After recognizing a sufficient level of distribution shift, a separate system may be trained to guide the agent back to safety regions of operation.

Bibliography

[1] N. Al-Naami, *Imitation learning using reward-guided dagger*, Master's thesis, KTH, School of Electrical Engineering and Computer Science (EECS), 2020.

[2] J. Altosaar, *Tutorial - what is a variational autoencoder?* https://jaan.io/what-is-variational-autoencoder-vae-tutorial/, Aug. 2016.

[3] A. Amini, W. Schwarting, G. Rosman, B. Araki, S. Karaman, and D. Rus, *Variational autoencoder for end-to-end control of autonomous driving with novelty detection and training de-biasing*, 10 2018, pp. 568–575.

[4] U. Ates, *Long-term planning with deep reinforcement learning on autonomous drones*, 2020.

[5] AurelianTactics, *Ppo hyperparameters and ranges.* https://medium.com/aureliantactics/ppo-hyperparameters-and-ranges-6fc2d29bccbe, July 2018.

[6] O. K. Beng, *We all need overexposure.* https://wikibeng.com/2016/08/16/we-all-need-overexposure-2/, Aug. 2016.

[7] N. Bhushan, *Uav: Trajectory generation and simulation*, University of Texas Arlington, (2019).

[8] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, *End to end learning for self-driving cars*, 2016.

[9] R. Bonatti, R. Madaan, V. Vineet, S. Scherer, and A. Kapoor, *Learning controls using cross-modal representations: Bridging simulation and reality for drone racing*, 09 2019.

[10] R. Bonatti, R. Madaan, V. Vineet, S. Scherer, and A. Kapoor, *Learning visuomotor policies for aerial navigation using cross-modal representations*, 2020.

[11] K. Chilamkurthy, *Off-policy vs on-policy vs offline reinforcement learning demystified!* https://kowshikchilamkurthy.medium.com/off-policy-vs-on-policy-vs-offline-reinforcement-learning-demystified-f7f87e275b48, Nov. 2020.

[12] DavisWiki, *Putah creek riparian reserve.* https://localwiki.org/davis/Putah_Creek_Riparian_Reserve, Nov. 2013.

[13] DearJudge, *Ddpg vs ppo vs sac: when to use?* https://www.reddit.com/r/reinforcementlearning/comments/holioy/ddpg-vs-ppo-vs-sac-when-to-use/, July 2020.

[14] Z. Dukowitz, *What are gps-denied drones and why are they important?* `https://uavcoach.com/gps-denied-drones/`, Sept. 2020.

[15] J. Farris, *Forging new paths for filmmakers on "the mandalorian"*. https://www.unrealengine.com/en-US/blog/forging-new-paths-for-filmmakers-on-the-mandalorian, Feb. 2020.

[16] A. Filos, P. Tigas, R. McAllister, N. Rhinehart, S. Levine, and Y. Gal, *Can autonomous vehicles identify, recover from, and adapt to distribution shifts?*, 2020.

[17] A. Gupta, A. S. Khwaja, A. Anpalagan, L. Guan, and B. Venkatesh, *Policy-gradient and actor-critic based state representation learning for safe driving of autonomous vehicles*, Sensors, 20 (2020).

[18] R. Haque, *Ddpg or dqn - which to use?* https://www.linkedin.com/pulse/ddpg-dqn-which-use-ridhwanul-haque/, 2020.

[19] J. Hui, *Rl the math behind trpo & ppo*. https://jonathan-hui.medium.com/rl-the-math-behind-trpo-ppo-d12f6c745f33, Sept. 2018.

[20] E. Kan, *What the heck are vae-gans?* https://towardsdatascience.com/what-the-heck-are-vae-gans-17b86023588a, Aug. 2018.

[21] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer, *Hg-dagger: Interactive imitation learning with human experts*, in 2019 International Conference on Robotics and Automation (ICRA), 2019, pp. 8077–8083.

[22] S. Kim, J. Park, J.-K. Yun, and J. Seo, *Motion planning by reinforcement learning for an unmanned aerial vehicle in virtual open space with static obstacles*, 2020 20th International Conference on Control, Automation and Systems (ICCAS), (2020).

[23] S. Levy, *Airsim blocks world*. https://www.youtube.com/watch?v=0BFW4bMs7BY, Sept. 2017.

[24] T. Li, X. Zhu, and X. Liu, *An end-to-end network slicing algorithm based on deep q-learning for 5g network*, IEEE Access, 8 (2020), pp. 122229–122240.

[25] X. Li, Q. Wang, J. Liu, and W. Zhang, *Trajectory design and generalization for uav enabled networks:a deep reinforcement learning approach*, in 2020 IEEE Wireless Communications and Networking Conference (WCNC), 2020, pp. 1–6.

[26] Y. Li, H. Li, Z. Li, H. Fang, A. K. Sanyal, Y. Wang, and Q. Qiu, *Fast and accurate trajectory tracking for unmanned aerial vehicles based on deep reinforcement learning*, in 2019 IEEE 25th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2019, pp. 1–9.

[27] L. Lopez-Fuentes, C. Rossi, and H. Skinnemoen, *River segmentation for flood monitoring*, in 2017 IEEE International Conference on Big Data (Big Data), 2017, pp. 3746–3749.

[28] R. Madaan, N. Gyde, S. Vemprala, M. Brown, K. Nagami, T. Taubner, E. Cristofalo, D. Scaramuzza, M. Schwager, and A. Kapoor, *Airsim drone racing lab*, 2020.

[29] G. Matheron, N. Perrin, and O. Sigaud, *The problem with ddpg: understanding failures in deterministic environments with sparse rewards*, 2019.

[30] MIMORALEA, *Comparison & selection of rl algorithms in continuous action spaces.* https://www.reddit.com/r/reinforcementlearning/comments/8w7mn2/comparison_selection_of_rl_algorithms_in/, Nov. 2018.

[31] MIT TECHNOLOGY REVIEW INSIGHTSARCHIVE PAGE, *Self-driving cars take the wheel.* https://www.technologyreview.com/2019/02/15/137381/self-driving-cars-take-the-wheel/, 2020.

[32] NATIONAL HIGHWAY TRAFFIC SAFETY ADMINISTRATION, *Automated vehicles for safety.* https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety, 2021.

[33] H. T. NGUYEN, M. GARRATT, L. T. BUI, AND H. ABBASS, *Supervised deep actor network for imitation learning in a ground-air uav-ugvs coordination task*, in 2017 IEEE Symposium Series on Computational Intelligence (SSCI), 2017, pp. 1–8.

[34] A. PETERSON, *How ai has advanced during the 21st century and where it's headed.* `https://www.pro-sapien.com/blog/how-ai-has-advanced-during-21st-century-and-where-its-headed/`, Feb. 2021.

[35] A. ROBINS, *Stochastic vs deterministic models: Understand the pros and cons.* https://blog.ev.uk/stochastic-vs-deterministic-models-understand-the-pros-and-cons, July 2020.

[36] ROBOTICS SIMULATION SERVICES, *Ros gazebo: Everything you need to know.* https://roboticsimulationservices.com/ros-gazebo-everything-you-need-to-know/, Jan. 2021.

[37] A. E. SALLAB, M. ABDOU, E. PEROT, AND S. YOGAMANI, *End-to-end deep reinforcement learning for lane keeping assist*, 2016.

[38] J. SCHULMAN, F. WOLSKI, P. DHARIWAL, A. RADFORD, AND O. KLIMOV, *Proximal policy optimization algorithms*, 2017.

[39] A. SINGH, H. KALKE, M. LOEWEN, AND N. RAY, *River ice segmentation with deep learning*, IEEE Transactions on Geoscience and Remote Sensing, 58 (2020), p. 7570–7579.

[40] Y. SONG, M. STEINWEG, E. KAUFMANN, AND D. SCARAMUZZA, *Autonomous drone racing with deep reinforcement learning*, 2021.

[41] U. TEWARI, *Which reinforcement learning-rl algorithm to use where, when and in what scenario?* https://medium.datadriveninvestor.com/which-reinforcement-learning-rl-algorithm-to-use-where-when-and-in-what-scenario-e3e7617fb0b1, Apr. 2020.

[42] C. WANG, J. WANG, X. ZHANG, AND X. ZHANG, *Autonomous navigation of uav in large-scale unknown complex environment with deep reinforcement learning*, in 2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP), 2017, pp. 858–862.

[43] C. WU, B. JU, Y. WU, X. LIN, N. XIONG, G. XU, H. LI, AND X. LIANG, *Uav autonomous target search based on deep reinforcement learning in complex disaster scene*, IEEE Access, 7 (2019), pp. 117227–117245.

[44] C. YANG, *Unmanned autonomous systems in complex environments 2021*, Wiley Hindawi Complexity, (2021).

[45] C. Yoon, *Deep deterministic policy gradients explained.* https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b, May 2019.

[46] ———, *Understanding actor critic methods and a2c.* https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f, Feb. 2019.

[47] J. Zhang, Z. Yu, S. Mao, S. C. G. Periaswamy, J. Patton, and X. Xia, *Iadrl: Imitation augmented deep reinforcement learning enabled ugv-uav coalition for tasking in complex environments*, IEEE Access, 8 (2020), pp. 102335–102347.