Study of Learning based algorithms for UAV Motion Planning

Manthan Patel, Jigme Tsering, Min Woo (David) Kong
University of Toronto
AER1516 - Final Report

1 Introduction

Unmanned Aerial Vehicles (UAVs) is a type of an aircraft that has reached to one of top interests due to the usages in a wide range of applications from surveillance and inspection to delivery and transportation as well as circumstances where they are too dangerous or difficult for human pilots. The term UAV itself could have two meanings of 1. remotely piloted by a human operator or 2. fully autonomous either operating with pre-programmed flight plans or with learning-based algorithms to navigate and operate in the environments. This paper is specifically focuses on learning-based algorithms where it can be utilized in a situation with costly, time-consuming, and requiring specialized training. [1][2]

With fully autonomous operating systems, it is important to develop an effective motion planning technique to maximize a successful performance for UAVs because it would maneuver for longer periods without fatigue and should perform tasks more efficiently and accurately than human operators or pre-programmed or classical approaches. In addition, the ability to generate a smooth and precise trajectory is crucial for UAVs to accomplish a complex task. By the way, classical approaches like artificial potential fields, sampling-based, and bio-inspired heuristic methods, have been widely used to solve the motion planning problem. However, most of these methods are ineffective in highly dynamic and high-dimensional configuration space due to the high computational cost and low convergence rates impeding real-time implementations. Recently, learning-based methods have gained considerable attention in tackling the motion planning problem due to their generalization to unseen scenarios and ability to deal with complex issues such as wind and weather changes. [3]

The study of learning-based algorithms for UAV motion planning involves developing algorithms that allow UAVs to learn from their environment and adapt their behavior accordingly. These algorithms typically use machine learning techniques such as reinforcement learning, deep learning, and genetic algorithms to generate optimized paths for the UAVs. In other words, the goal of these algorithms is to enable UAVs to be able to adapt to changing conditions in real-time and make decisions based on data from their sensors, such as cameras and LIDAR sensors. In the main portion of this paper, various reinforcement learning – Proximal Policy Optimization (PPO), Deep Q-Network (DQN), Advantage Actor Critic (A2C), and Deep Deterministic Policy Gradient (DDPG) were implemented using motion planning on Microsoft AirSim simulator software to compare their different performances. To continue to go more in depth different neural network architectures could be tried out to compare its performance with existing network. It is essential to choose the appropriate algorithm based on the specific requirements and constraints of the task at hand.

2 Background and Relevance

Fadi AlMahamid et al. in the paper of "Autonomous Unmanned Aerial Vehicle Navigation using Reinforcement Learning: A Systematic Review" provided a comprehensive survey of the existing literature on the use of reinforcement learning for autonomous UAV navigation. A systematic review of 40 papers published between 2012 and 2019 was conducted and based on the categorization of the type of reinforcement learning algorithm used, the application domain, and the evaluation metrics used were classified.

Four main types of reinforcement learning algorithms used in the surveyed papers were identified: Q-learning, policy gradient methods, actor-critic methods, and deep reinforcement learning. These algorithms were applied in four main application domains: path planning and obstacle avoidance, target tracking, formation control, and search and rescue. The results showed that reinforcement learning has the potential to enable autonomous UAVs navigation in complex and dynamic environments. However, there are still challenges that needs to be addressed, such as scalability, generalizability, and safety of these algorithms. Further research is needed to develop more efficient and effective reinforcement learning algorithms for UAVs navigation, and to ensure the safety of these algorithms in real-world scenarios. [4]

Followed to the first background and relevance paper, Amudhini P. Kalidas et al. in the paper of "Deep Reinforcement Learning for Vision-Based Navigation of UAVs in Avoiding Stationary and Mobile Obstacles" proposed a learning-based approach to motion planning for UAVs in complex and dynamic environments. A combination of deep neural networks and reinforcement learning to enable UAVs to navigate through an environment while avoiding stationary and mobile obstacles; three different reinforcement learning algorithms – Deep Q-network (DQN), PPO, and SAC were evaluated which algorithm was suitable for large 3D environments with both stationary and mobile obstacles created by AirSim to mimic the real world as much as possible.

DQN, PPO, and SAC results in a simulated environments with obstacles were introduced including the analysis of the strengths, weaknesses, and trade-offs. Fundamentally, all three algorithms showed that the mean rewards had been increased as the number of training steps increased. However, DQN and SAC which are off-policy algorithms returned the promising results while PPO which is on-policy algorithm was not able to return the promising results. SAC with the actor-critic reinforcement learning approach performed the best among the three whereas PPO with policy gradient reinforcement learning approach performed the inadequately for effective collision avoidance in large 3D environments with dynamic actors. Overall, the study concluded that off-policy algorithms are more efficient in avoiding collisions in dynamic environments than on-policy algorithms.

Learning-based algorithms have shown the promise in UAVs motion planning as they can adapt to changing environments and complex tasks, without requiring explicit models of the environment or the UAVs dynamics. These algorithms learn by interacting with the environment and receiving feedback in the form of rewards, allowing the UAVs to learn to navigate through complex environments efficiently and effectively. [1]

Lastly, Lun Quan et al. in the paper of "Survey of UAV motion planning" introduced an overview of the state-of-the-art techniques and algorithms for planning the motion of UAVs in a variety of environments. The survey covers a wide range of topics, including path planning, trajectory planning, and control for UAVs. Overall, the survey provides a comprehensive overview of the current state of UAV motion planning and serves as a valuable resource for researchers and practitioners in the field. [2]

With all three papers that were referred for this paper, reinforcement learning has the potential to enable autonomous UAVs navigation in complex and dynamic environments; DQN, PPO, and SAC were used to compare the performances that needs to combine with the current state of UAVs motion planning. Those led this paper show PPO, SAC, A2C, and DDPG reinforcement learning algorithms comparison to study the feasibility in real situation.

3 Literature Review

It is worth noting that a lot of research has been done on optimizing the autonomous navigation of unmanned aerial vehicles (UAVs) and their scheduling. Oleynikova et al. (2018) [5] developed a conservative trajectory-optimization-based local planner with a local exploration strategy for quadrotor navigation through an unknown, cluttered environment. The strategy involved selecting intermediate goals to maximize both the chances of reaching the final goal and potential exploration gain, thus increasing the likelihood of finding a feasible path. This method builds on the team's previous work [13], which proposed a reactive local planning method to avoid obstacles using disparity maps. Both methods require an explicit reconstruction of a local map first, and then perform planning within that map. For map representation, they used Voxblox[8], which allows the system to build maps of any size incrementally from sensor data in real-time. The researchers validated their system with extensive simulations and experiments in highly cluttered environments, demonstrating that it outperformed the standard approach of using an optimistic global planner and also performed better than single exploration step when the local planner was stuck.

Lopez and How (2017) [14] developed an algorithm called the Triple Integrator Planner (TIP) that enables collision avoidance for quadrotors during high-speed navigation by utilizing point cloud data and minimum-time motion primitives. The TIP algorithm adopts a world representation that is easily obtained from instantaneous perception data, such as point clouds, and leverages a closed-form solution for an optimal control problem with state and input constraints to generate high-performance motion primitives that are intelligently sampled for possible collisions. The researchers suggested that the computational burden of map building could be reduced by using a k-d tree, although the method still requires some form of world model representation. The experimental results demonstrated the algorithm's capability to plan and execute aggressive collision avoidance maneuvers.

In another method [15] (Alonso-Mora et al., 2018), neural network control policies are introduced as a computationally lightweight approach, which computes control commands directly from sensor inputs. To reproduce the behavior of the path following control algorithm with collision avoidance, an imitation learning algorithm is employed to generate a policy. The resulting policy exhibits the ability to perform local collision avoidance of obstacles that have not been encountered before while tracking a global reference path, thanks to the generalization capability of neural networks. However, this approach still necessitates guidance from a time-free model-predictive path following controller to evade obstacles.

The vision-based reactive planning algorithm, inspired by model predictive control, was developed by Penin et al. in 2018 [16]. Their approach utilizes multi-objective optimization with the occlusion constraint formulation and an online replanning strategy, which successively solves a nonlinear optimization problem. In order to include obstacles in a nonlinear program formulation, an exact representation of them is required. The researchers demonstrated the robustness and reactivity of the replanning algorithm through realistic simulation results.

In [17] Sadeghi and Levine (2016) utilized an end-to-end learning approach to train a deep RL agent to extract collision avoidance policies from virtual raw monocular images in various low fidelity simulation environments. They employed reinforcement learning to obtain accurate supervision that reflects the actual probabilities of collision, and deep convolutional neural networks were used to predict the probability of future collision from raw monocular images while also outputting velocity commands. Although this method overcomes the challenges of 3D reconstruction and shows promising results in terms of collision avoidance, it is still necessary to incorporate it into a complete path planning framework that enables the quadrotor to navigate to the goal as quickly as possible.

Blukis et al. in 2018[18] propose another learning-based algorithm that uses direct mapping between images, instructions, pose estimates, and control to follow high-level navigation instructions. They use a Grounded Semantic Mapping Network (GSMN) model, which consists of a single neural network that projects learned features from the agent camera frame into the global reference frame, thus explicitly maintaining a semantic map. The map representation is learned from the data and the local-to-world transformation is computed explicitly. The GSMN algorithm was tested in a virtual environment on a realistic quadcopter simulator and showed better performance than strong neural baselines.

Most state-of-the-art navigation methods separate sensing, planning, and control for autonomous navigation to achieve modularity, which allows for design, development, and analysis within their respective scopes. However, if any module in the pipeline fails, it is likely that other modules will fail as well. To overcome this issue, a single module end-to-end reasoning method using deep reinforcement learning was developed by Camci et al. (2020) in [7]. They introduced an RL agent that utilizes only raw depth images and relative position information of a moving setpoint on the initial path to the goal to generate local motion plans. The RL algorithm used is an off-policy, episodic training algorithm that employs DQN (Deep Q-network) to estimate the action-value function governing the agent's policy. To validate the method, the researchers trained the agent in seven virtual environments to acquire diversified retrospective knowledge in the first stage, and then tested the agent in ten unseen virtual environments in the second stage to demonstrate sufficient generalization. The same agent was successfully tested in real flights to exhibit simulation-to-real knowledge transfer and it completed 15 out of 15 flights without a crash while avoiding obstacles and reaching the goal point.

Omar et al. (2020) [11] proposed a framework for autonomous navigation of an Unmanned Aerial Vehicle (UAV) using the Deep Deterministic Policy Gradient (DDPG) approach for the purpose of using the UAV as an Internet of Things (IoT) device to navigate obstacles and reach a desired location. The proposed framework considers the energy constraints of the UAV and employs a customized reward function that aims to minimize the distance between the UAV and the destination while also penalizing collisions. In case of battery depletion, a dynamic energy threshold is used to redirect the UAV towards a charging station. The authors demonstrated that their DDPG approach achieved comparable performance to one of the graph-based Dijkstra's algorithm in simulations.

Bilal Kabas in 2022[10] proposed using cheaper and lighter RGB cameras for autonomous navigation, as sensors such as LIDAR, RADAR, or depth cameras are not feasible due to their cost and weight for micro drones. To handle continuous action spaces and achieve monotonic improvement in policy, he employed the PPO (Proximal Policy Optimization) algorithm, which is a deep reinforcement learning technique. The agent was trained in one virtual environment and tested it in another. The system was trained using three types of inputs, including depth image, single RGB, and multiple grayscale images. The model trained with multiple grayscale images outperformed the one trained with a single RGB image.

The Advantaged Actor-Critic (A2C) algorithm was proposed by Xiao et al. in 2019 [12] to train a decision-making model for multi-UAV obstacle avoidance. They aimed to enhance the A2C algorithm for multiple UAV scenarios by allowing experience sharing among the UAVs, thus accelerating the training process and maximizing performance. The results demonstrated that the Experience-shared A2C algorithm outperformed the traditional approach, with a shorter training period.

Taking inspiration from the methods mentioned earlier, we took on a project to implement a variety of reinforcement learning (RL) algorithms such as Proximal Policy Optimization (PPO), Deep Q-Network (DQN), Advantaged Actor-Critic (A2C), and Deep Deterministic Policy Gradient (DDPG) for end-to-end motion planning on the Microsoft AirSim simulator. It was carefully evaluated and compared the paths and reward plots generated from each algorithm to determine their effectiveness. Additionally, the modifications were introduced to the neural network architecture and observed the performance of the altered models against the baseline models. It was trained the models using multiple RGB images instead of a single RGB image, which led to a noticeable improvement in the policies and paths generated for the quadrotors.

4 Implementation

This is the fundamentals of reinforcement learning algorithms and implementation to broaden understanding of the concepts involved in this paper.

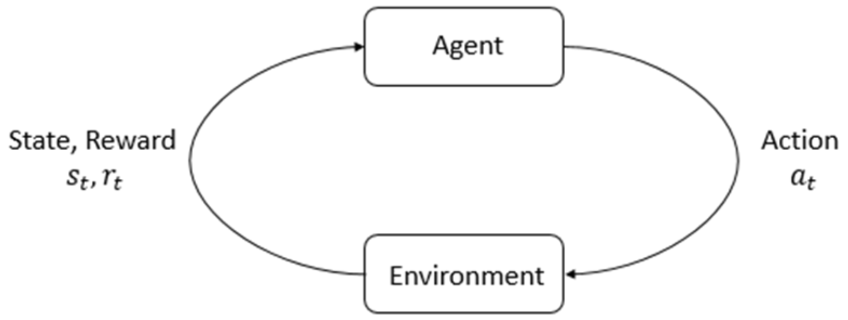


Figure 1 – Reinforcement Learning Block Diagram [24]

The main characters of RL are the agent and the environment. The environment is the world that the agent lives in and interacts with. At every step of interaction, the agent sees a (possibly partial) observation of the state of the world, and then decides on an action to take. The environment changes when the agent acts on it but may also change on its own.

The agent also perceives a reward signal from the environment, a number that tells it how good or bad the current world state is. The goal of the agent is to maximize its cumulative reward, called return. Reinforcement learning methods are ways that the agent can learn behaviors to achieve its goal.

Some of the important terminology related to reinforcement learning problem are.

- states and observations,
- action spaces,
- policies,
- trajectories,
- different formulations of reward,
- the RL optimization problem,
- and value functions.

States and Observations: A state s is a complete description of the state of the world. There is no information about the world which is hidden from the state. An observation o is a partial description of a state, which may omit information. In deep RL, we almost always represent states and observations by a real-valued vector, matrix, or higher-order tensor. For instance, a visual observation could be represented by the RGB matrix of its pixel values; the state of a robot might be represented by its joint angles and velocities. When the agent can observe the complete state of the environment, we say that the environment is fully observed. When the agent can only see a partial observation, we say that the environment is partially observed.

Action Space: The action space of a given environment refers to the set of all valid actions an agent can take. Environments can have either discrete or continuous action spaces, depending on whether the agent can choose from a finite set of actions or a continuous range of real-valued actions. Discrete action spaces are common in games like Atari and Go, whereas continuous action spaces are encountered in tasks such as robotics and control. The choice of action space significantly impacts the choice of deep reinforcement learning algorithm used to solve the task. Some algorithms are designed for discrete action spaces, while others are specialized for continuous action spaces and require function approximators to estimate optimal action-value or policy functions. Choosing the right algorithm for the action space can have a profound impact on the performance of the agent.

Policies: A policy is a rule used by an agent to decide what actions to take. It can be deterministic; in which case it is usually denoted by μ .

$$a_t = \mu(s_t)$$

or it may be stochastic, in which case it is usually denoted by π:

$$a_t = \pi(.\ |s_t|)$$

It is not uncommon to substitute the word "policy" for "agent" because the policy is essentially the agent's brain, e.g., saying "The policy is trying to maximize reward." In deep RL, we deal with parameterized policies: policies whose outputs are computable functions that depend on a set of parameters (e.g., the weights and biases of a neural network) which we can adjust to change the behavior via some optimization algorithm.

It is often denoted the parameters of such a policy by θ or φ , and then write this as a subscript on the policy symbol to highlight the connection:

$$a_t = \pi_\theta(s_t)$$

$$a_t = \pi_\theta(.\ |s_t|)$$

Reward: The reward function R is critically important in reinforcement learning. It depends on the current state of the world, the action just taken, and the next state of the world:

$$r_t = R(s_t,\ a_t\ ,\ s_{t+1})$$

although frequently this is simplified to just a dependence on the current state, $r_t = R(s_t)$, or state-action pair $r_t = R(s_t,\ a_t)$. The goal of the agent is to maximize some notion of cumulative reward over a trajectory, but this can mean a few things. We'll notate all these cases with R(τ), and it will either be clear from context which case we mean, or it won't matter (because the same equations will apply to all cases).

The categorization of reinforcement learning algorithms is based on the type of states and actions used, and choosing the most appropriate algorithm depends on the characteristics of the task environment and objective formulation. Deep reinforcement learning (DRL) algorithms, including deep Q-networks (DQNs) and actor-critic methods, have demonstrated encouraging outcomes in UAV navigation tasks. The selection of suitable state and action representations is crucial for the success of reinforcement learning algorithms in UAV navigation, and different representations may be more appropriate for various tasks. Creating a proper reward function that aligns with the intended goal is critical for the success of reinforcement learning, and the choice of reward function should balance achieving the task objective and ensuring safe operation of the UAV. The learning approach, such as online or offline learning, can significantly influence the effectiveness and stability of reinforcement learning algorithms. The assessment of reinforcement learning algorithms in UAV navigation should consider both objective metrics, such as success rate and task completion time, and subjective metrics, such as user satisfaction and ease of use.

PPO (Proximal Policy Optimization) is a policy gradient algorithm used in reinforcement learning that optimizes the policy function efficiently. It achieves this by using a clipped surrogate objective function that limits policy updates per iteration to prevent drastic changes in policy. PPO's strengths include its simplicity and versatility in handling both discrete and continuous action spaces, making it a model-free algorithm. [19]

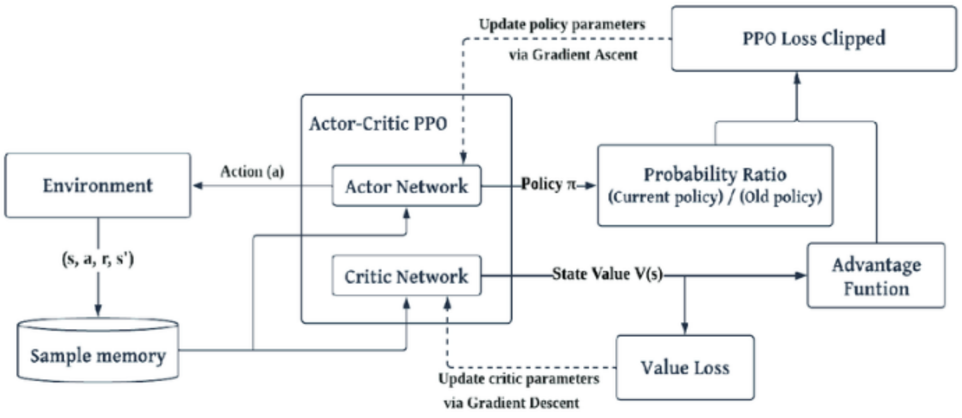


Figure 2 – Architecture of PPO

A2C (Advantage Actor Critic) is an on-policy algorithm used in reinforcement learning that learns both a value function and a policy function. It is a synchronous version of the Asynchronous Advantage Actor Critic (A3C) algorithm, which uses multiple parallel workers to generate multiple trajectories simultaneously. A2C combines the benefits of policy gradient and value-based methods by using an actor-critic architecture, which can improve the stability and convergence speed of the algorithm. The algorithm is relatively simple and stable, making it easy to implement and train efficiently in parallel environments. It can learn from the most recent experiences, making it useful in non-stationary environments. [20]

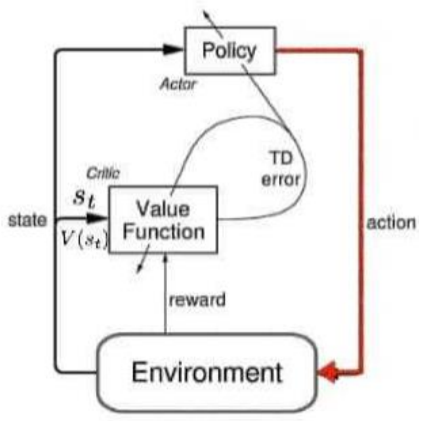


Figure 3 – Architecture of A2C

DDPG (Deep Deterministic Policy Gradient) is a model-free, off-policy algorithm used in reinforcement learning that uses a deterministic policy gradient to optimize the policy function. It is a variation of the actor-critic algorithm, where the actor learns a deterministic policy, and the critic learns the value function. DDPG is designed for environments with continuous action spaces, making it well-suited for robotics and control tasks. The algorithm is stable, can be easily implemented and trained, and benefits from an actor-critic architecture, which can improve the stability and convergence speed of the algorithm. Being an off-policy algorithm, it can learn from old experiences, making it more sample-efficient than on-policy algorithms. [21]

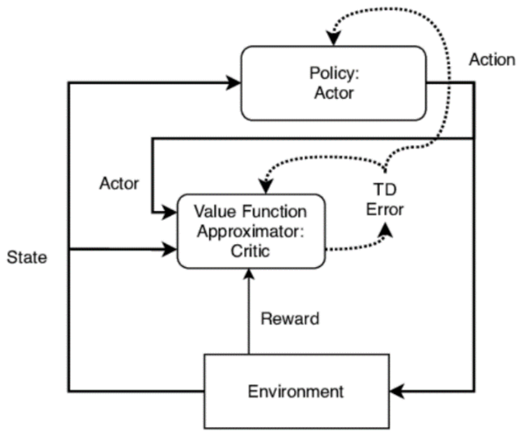


Figure 4 – Architecture of DDPG

DQN (Deep Q-Network) is a type of reinforcement learning algorithm that uses a neural network to approximate the Q-function, which estimates the expected future reward for taking a particular action in each state. DQN is designed to work with discrete action spaces, where the agent can only choose from a finite set of actions. In DQN, the Q-function is represented as a table, where each row corresponds to a state and each column corresponds to an action. The table is updated iteratively based on the agent's experience, and the agent selects actions based on the highest Q-value for the current state. DQN has been used in various applications, such as playing Atari games, controlling robotic arms, and optimizing energy consumption in buildings. [22]

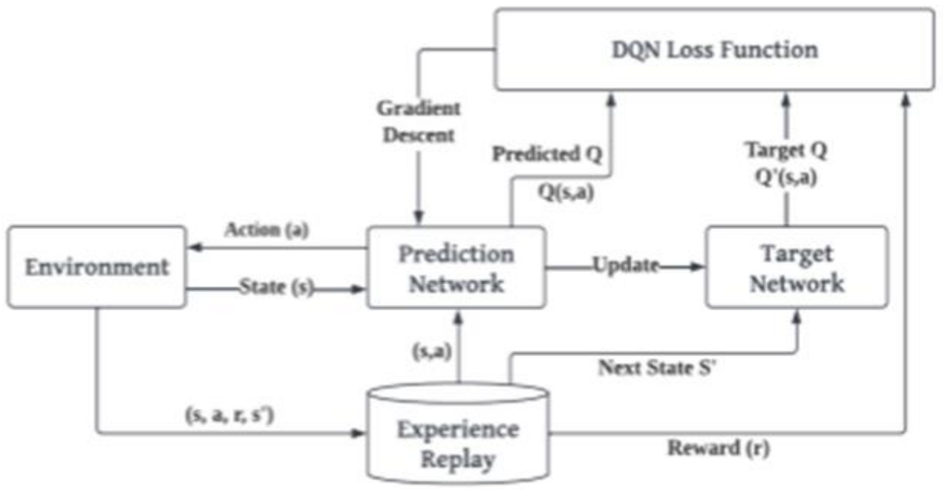


Figure 5 – Architecture of DQN

Table 1 – Reinforcement Learning Process Steps [4]

Steps	Process	Details
1	Define State Type	Essential to comprehend the state that can be obtained from the surrounding environment using grid cell representations, where the agent has a limited and predetermined set of states, whereas in other tasks, the environment can have unlimited states. [23]
2	Define Action Type	Choosing between discrete and continuous action types limits the number of applicable algorithms. For instance, discrete actions can be used to move the UAV in pre-specified directions (UP, DOWN, RIGHT, LEFT, etc.), whereas continuous actions, such as the change in pitch, roll, and yaw angles.
3	Define Policy Type	RL algorithms can be either off-policy or on-policy algorithms – On-policy learning involves using the same policy that the agent is currently following to generate experience and update the policy; Off-policy learning, on the other hand, involves using a different policy than the one the agent is currently following to generate experience and update the policy.
4	Define Processing Type	Select the processing type that suits the application needs and the available computational power – single thread that it can only process one task at a time; multiple threads which allows the agent to process multiple tasks simultaneously; distributed processing that can be even more efficient than multi-threading, as the workload being distributed across many machines.
5	Define Number of Agents	Specifies the number of agents the application should have – a single agent that interacts with the environment to learn a policy that maximizes its cumulative reward; multiple agents that interact with each other and the environment to learn a policy.
6	Select the Algorithms	A collection of algorithms that may be applied to the RL problem at hand.

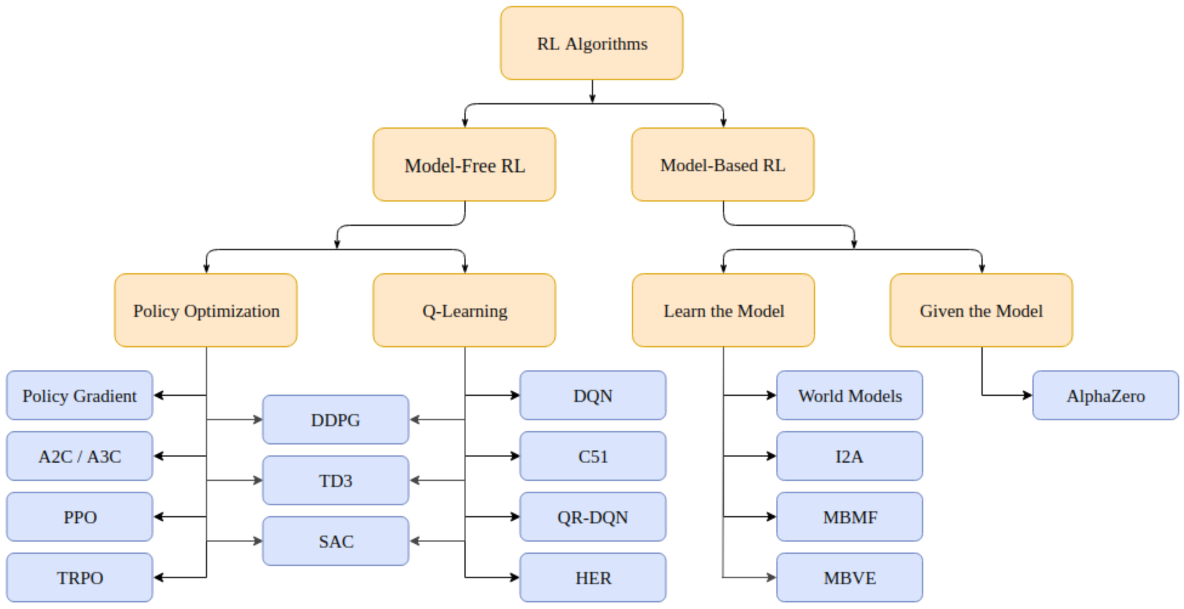


Figure 6 – Algorithm Selection Process

This work explores the effectiveness of PPO, A2C, DDPG, and DQN reinforcement learning algorithms in motion planning of UAVs with obstacles, using available open-source repositories. The study involves generating paths and reward plots to compare the performances of these algorithms and determine which is best suited for different conditions. Our research is influenced by the work of Bilal et al. (2022) and Efe Camci et al. (2019), who used PPO and DQN algorithms, respectively, for UAV motion planning using raw depth and RGB images. To implement our comparison of reinforcement learning algorithms, the Github repository provided by Bilal Kabas for his paper Bilal et al. (2022) was used as a base code structure. Repo: https://github.com/bilalkabas/DRL-Nav

4.1 Environment Setup

Simulation Setup: In this study, simulation environments were created using the Unreal Engine game engine and made possible to use the UAV (Unmanned Aerial Vehicle) in these environments with the Microsoft AirSim [13] plugin. With the API provided by AirSim, sensor data can be obtained from the UAV and various control commands can be sent to the UAV. As shown in Figure 1, two different environments were created for training and testing. The training environment consists of 15 rooms, each separated by circular passages with walls that the UAV can pass through. The test environment, on the other hand, has 16 rooms and 15 passages. The diameters of the passages are fixed, but the positions of their centers are different. Different texture types are used in the same environment. The textures of the walls with passages in the training and test environments are different from each other.

Setup:

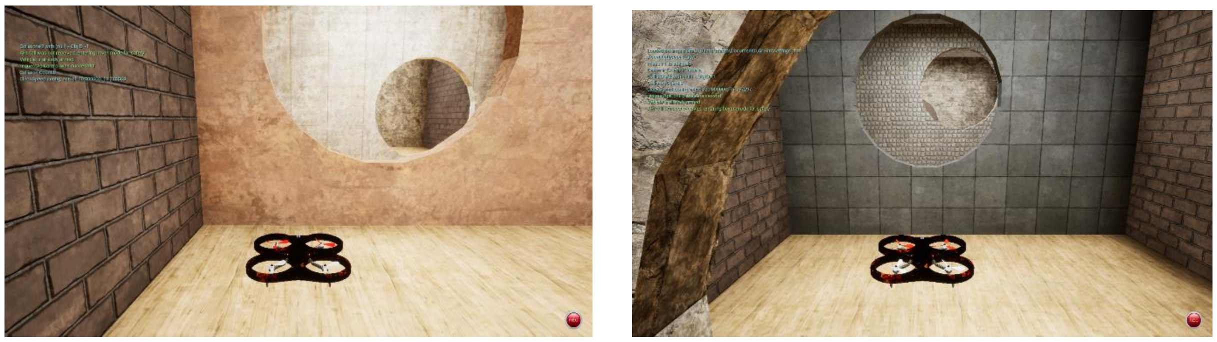


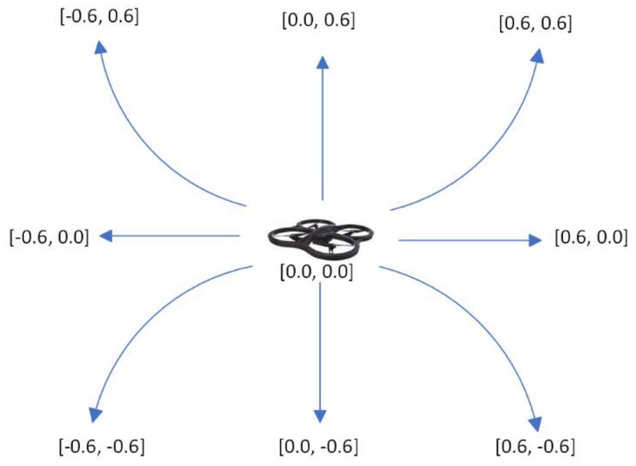
Figure 7 – AirSim Environment

In deep reinforcement learning, the interactions of the robot with the environment are expressed with a triplet of state, action, and reward (s_t,a_t,r_t). The state information is obtained directly from the camera images in the form of $H \times W \times C$, where H represents the height pixel count, W represents the width pixel count, and C represents the number of channels. The state information is the input to the artificial neural network. In this study, three different types of inputs are used, which are depth image, RGB camera image, and concatenated grayscale image. The depth image and RGB image perceived by the UAV (Unmanned Aerial Vehicle) are obtained using AirSim. In the depth image, objects closer to the UAV appear darker. To obtain the concatenated grayscale image, the RGB camera images at time t , $t-1$ and $t-2$ are first converted to grayscale images. These grayscale images are then concatenated to obtain a multi-channel grayscale image with dimensions of H^ × (W^ × 3) × C^.

As an additional modification to above observation space, we have used 5 RGB camera images at a time of t , $t-1$, $t-2$, $t-3$, $t-4$ which are converted to grayscale and then concatenated.

Action Space: As shown in Figure 3, the action of the UAV at time t is represented as $[v_{y,t}, v_{z,t}]$, which determines the speed of the UAV along the y and z axes. The UAV always moves at a constant speed of 0.4 m/s in the x direction (v_x= 0.4 m/s). The values of v_y and v_z are limited to the range of [-0.6, 0.6] m/s, allowing maneuvers of up to approximately 113 degrees in the xy and xz planes. The UAV does not perform any rotation motion in the z axis. As our modification for trials, we also tried training our model with action space of range [-1.0, 1.0] to train on PPO, DDPG and A2C.

Our model is also trained to be worked with DQN network since DQN works with discrete action space modified form [-0.6, 0.6] to 8 separate actions based on below shown diagram.



Reward Function: The reward function consists of two terms as shown in equation below. The first term r_t^d is calculated based on the Euclidean distance of the UAV to the gate on the wall.

$$r_t = r_t^d + r_t^c$$

$$r_t^d = 30 \times e^{-\|p_{a,t} - p_{h,t}\|_2}$$

The value of second term r_t^c is determined based on certain conditions. The function $C_{coll,t}$takes the value of 1 if there is a collision at time t, otherwise 0. Similarly, the function $M_{miss,t}$indicates whether the gate is visible in the camera image of the UAV and takes the value of 1 if it is not visible, otherwise 0. If $M_{miss,t}$ is 1, then r_t^c = -100.

$$r_t^c = \begin{cases} -100, & C_{coll,t} = 1 \\ -100, & M_{miss,t} = 1 \\ 0, & in\, other\, cases \end{cases}$$

The value of indicator function $M_{miss,t}$ is calculated based on the equation below.

$$M_{miss,t} = \begin{cases} 0, & d - r_h > (x_h - x_0 - x)sin(\frac{\theta}{2}) \\ 1, & in\, other\, cases \end{cases}$$

Here, r_h represents the radius of the passageway, $x_h$represents the x-position of the passageway, x_0 represents the x-position of the UAV at the start, and θ represents the field of view of the UAV's camera image.

CNN Architecture:

The architecture of the artificial neural network (NatureNet) trained here is shown in Figure below. The neural network consists of three convolutional and two fully connected layers, with normalization layers between the convolutional layers to prevent the ReLU (rectified linear unit) activation function from negatively affecting learning. Additionally, the stride size(s) in the first convolutional layer is set to two, so that information about the positions of features in the camera images is more accurately transmitted to the subsequent layers.

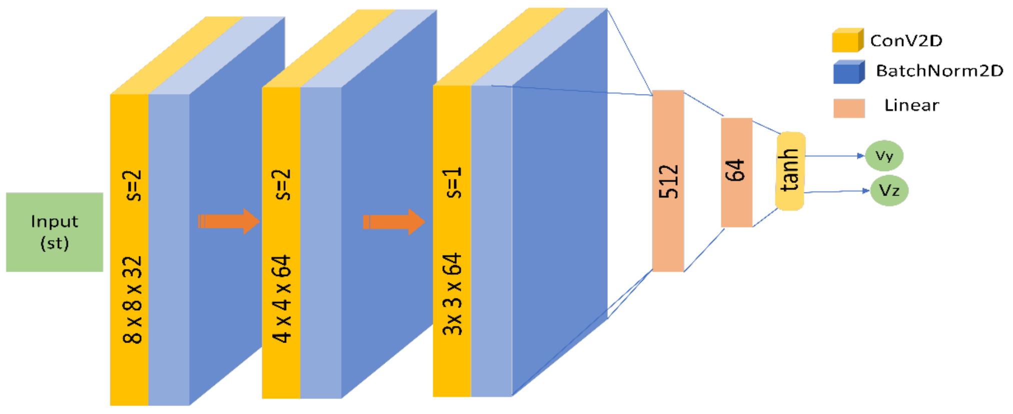


Figure 8 – CNN with 3 images layout filters

Another neural network (BasicNet) that we trained is deeper with 5 convolutional layers and two fully connected layers with normalization and dropout layers between the convolutional layers as shown in the following figure.

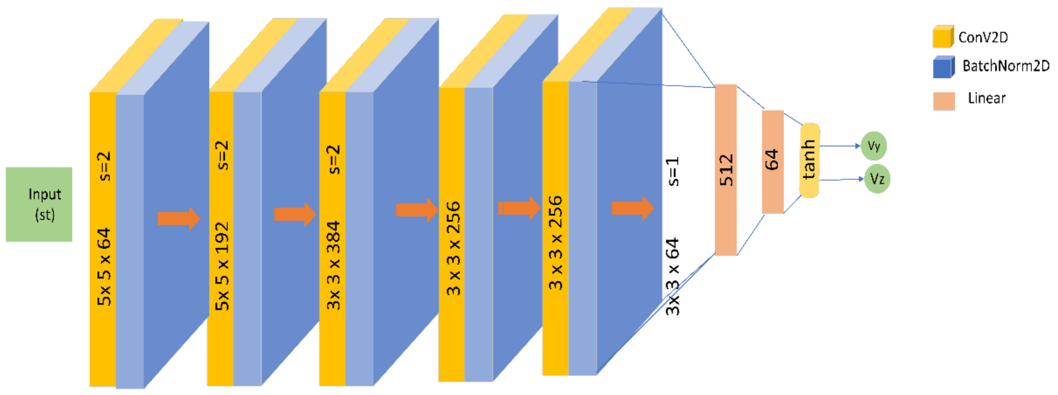


Figure 9 – CNN with 5 images layout filters

4.2 Training

3 PCs with GPU support were used to train models, 3 GPUs with 6GB, 4GB and 4GB of memory were used. Total of 9 training sessions with different algorithms and training conditions were performed. Our training sessions included of:

1. PPO with NatureNet
2. PPO with BasicNet
3. DDPG with NatureNet
4. DDPG with BasicNet
5. DQN with NatureNet
6. DQN with BasicNet
7. A2C with NatureNet
8. A2C with BasicNet
9. PPO with BasicNet, 5 image stack, Updated reward function, Different Action Space

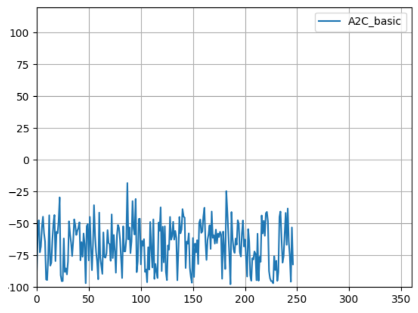
Above sessions were trained for 350,000 timesteps with batch size kept at 128. Our algorithms were evaluated after every 20 episodes and saved our trained model. While performing evaluation of algorithm, the Path generated by each algorithm during testing for comparison was saved.

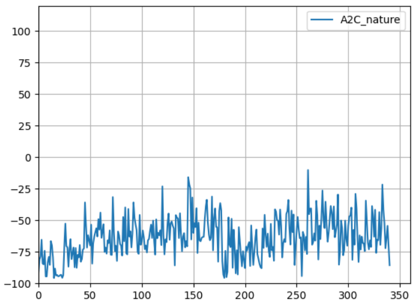
4.3 Results

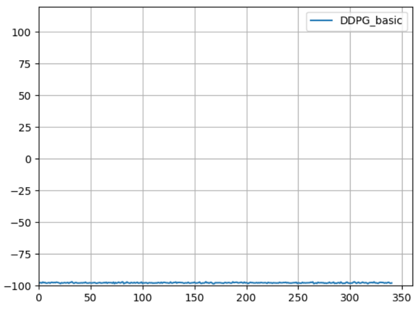
We compared the performance of four algorithms mentioned above based on four different parameters: reward plots generated during training, paths generated by each algorithm during testing, average distance taken by each algorithm to cover five walls, and success rate of algorithms over 100 episodes.

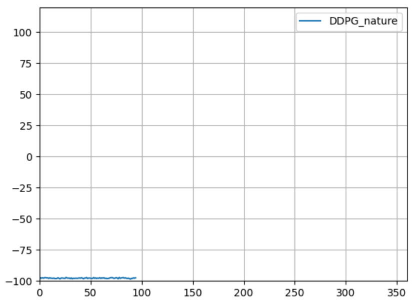
Reward Plot:

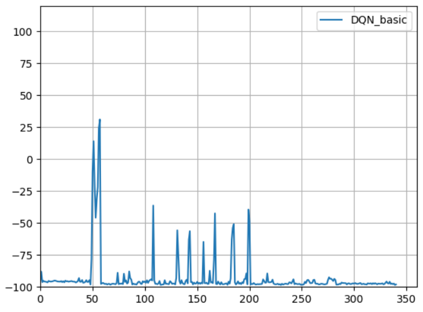
Below are the reward plots generated during the training process.

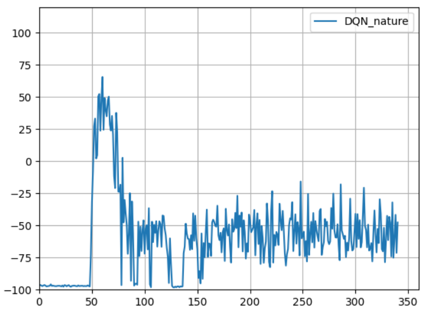












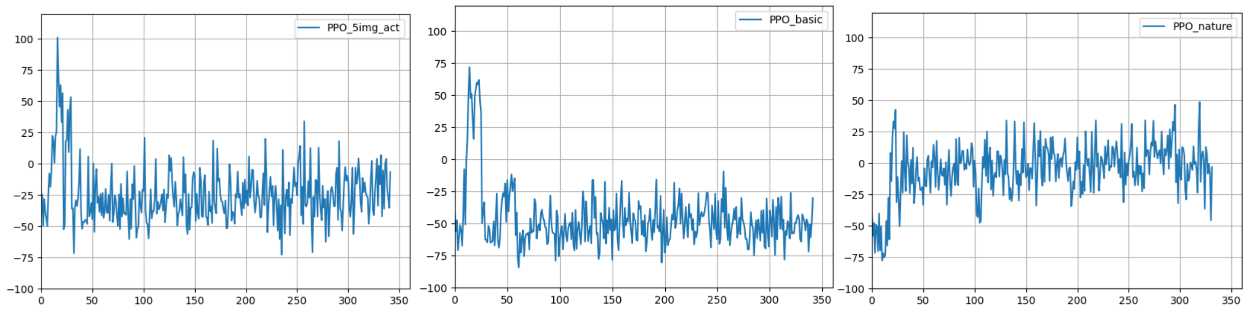


Figure 10 – Reward Plots for each RL algorithm

Based on the Reward plots presented above, we can conclude that the PPO and DQN algorithms have superior performance overall. The PPO algorithm with NatureNet has a higher average reward value than the PPO with BasicNet. However, some episodes of the BasicNet version have higher rewards than those of the NatureNet. It is worth noting that BasicNet has more parameters than NatureNet. Comparing the DQN algorithm with the PPO, we can observe that the average reward value for DQN is approximately -50, while for PPO, it is around 0.

Utilizations of DDPG and A2C algorithms were attempted; however, from the reward plots, we can deduce that these algorithms failed to converge and did not deliver satisfactory performance. During the training process, we randomly generated actions, but from the plots, it is evident that the reward values did not increase with an increasing number of timesteps. The average reward value for A2C was approximately -60, while for DDPG, it was -100. The plots demonstrate that either the On-Policy single thread model (PPO) or the Off-Policy (discrete) single thread model (DQN) are more effective on this environment and problem setup. Furthermore, it is important to note that the performance and convergence of different reinforcement learning algorithms can vary depending on the specific problem and environment setup. Thus, it is crucial to experiment with various algorithms and parameter settings before selecting the most suitable one for a particular application.

The choice between using a continuous or discrete action space in reinforcement learning algorithms depends on the nature of the task at hand. In some scenarios, the action space may be a continuous range of values, such as in robotic control, where the robot's joints can be rotated to any angle within a given range. On the other hand, in a game like chess, the possible moves are finite and discrete. The PPO and DQN algorithms operate on different types of action spaces, with PPO using a continuous action space and DQN using a discrete action space. When testing the DQN algorithm, the limited number of actions to choose from may result in sharp maneuvers as the agent is constrained to choose only from a set of pre-defined actions.

However, it is essential to note that a continuous action space can be challenging to optimize and may require specialized algorithms such as PPO. Conversely, discrete action spaces can be easier to optimize but may require a larger number of discrete actions to be effective, which could lead to increased computation time and memory usage. Thus, the choice of action space should be made based on the specific requirements of the task and the capabilities of the algorithm being used.

To enhance performance, we conducted an experiment in which we input five stacked grayscale images into the CNN architecture and altered the action space from [-0.6, 0.6] to [-1.0, 1.0]. Comparing this trial run with PPO (BasicNet), better performance was observed because of changing the observation and action space, with the improved reward values. However, it is worth noting that the true potential of the algorithms can only be realized with prolonged training and a greater number of timesteps. With additional training, it is likely that the algorithms will continue to improve and yield even better reward values.

Trajectory Plots:

To evaluate another parameter, we utilized the trajectory generated by the algorithm to traverse five distinct walls and collected the resulting path data. By plotting this data, we were able to visualize and compare the performance of the algorithms.

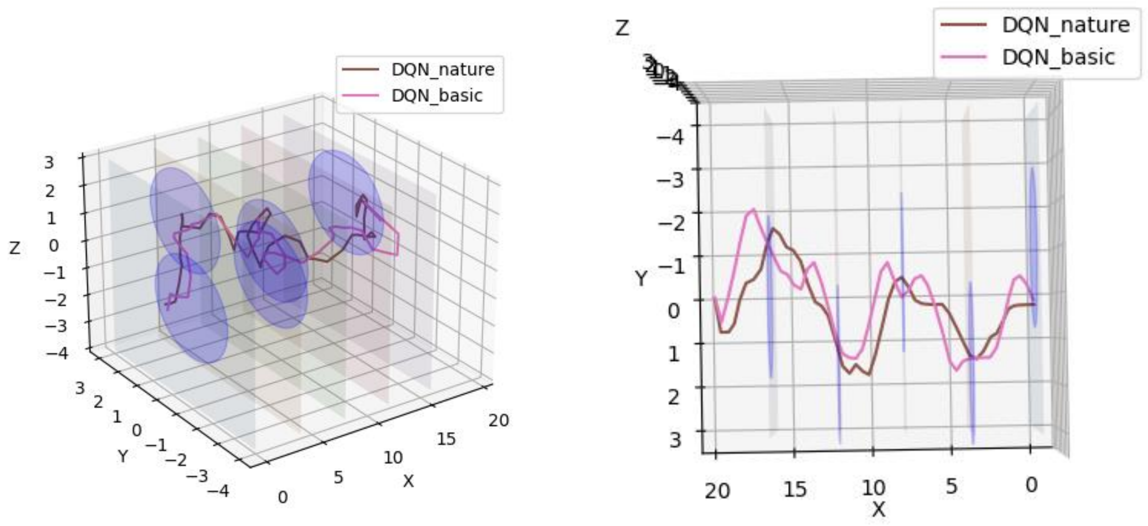


Figure 11 – 3D view and top view of trajectory plots for DQN Nature vs. DQN Basic

Upon examining the plots, it is evident that the DQN algorithm has been trained to a sufficient degree, allowing it to generate the necessary actions to navigate through walls with ease. From the plot on the left-hand side, we can observe that DQN with NatureNet generates a less intricate path, whereas BasicNet generates a more complex one.

Furthermore, when viewing the plot from a top-down perspective on the right-hand side, we can see that the BasicNet-generated path is marginally longer than the NatureNet-generated path. However, both generated paths exhibit sharp maneuvers, as DQN has access to a discrete action space, limiting the range of available actions.

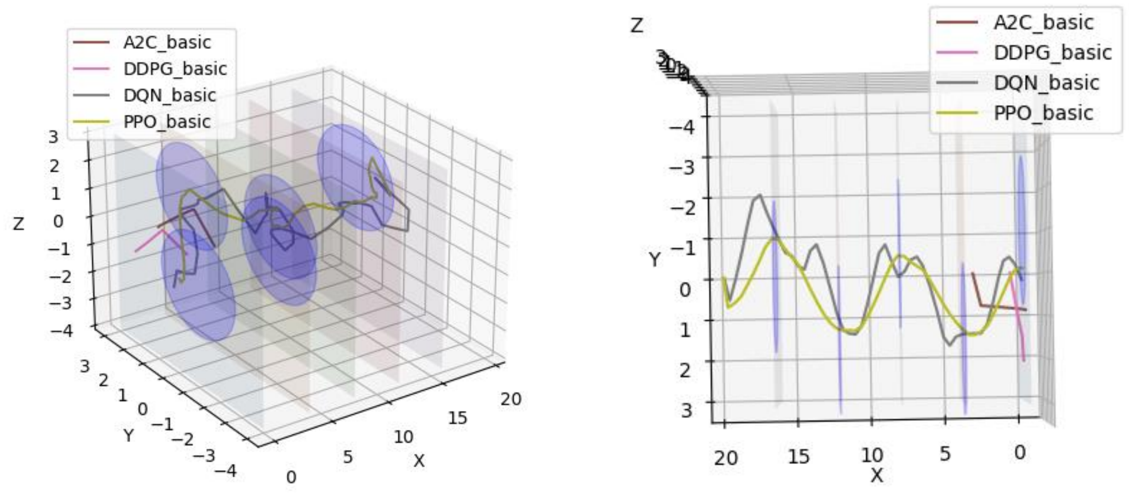


Figure 12 – 3D view and top view of trajectory plots for A2C Basic vs. DDPG Basic vs. DQN Basic vs. PPO Basic

The above plots illustrate the evaluation trajectories of all four algorithms, namely PPO, DDPG, A2C, and PPO trained on BasicNet. As previously noted, DQN and PPO outperform DDPG and A2C, which are unable to traverse any walls. PPO generates a seamless trajectory due to its ability to select actions from a continuous action space. Additionally, the length of the path generated by PPO is shorter than that of all other algorithms.

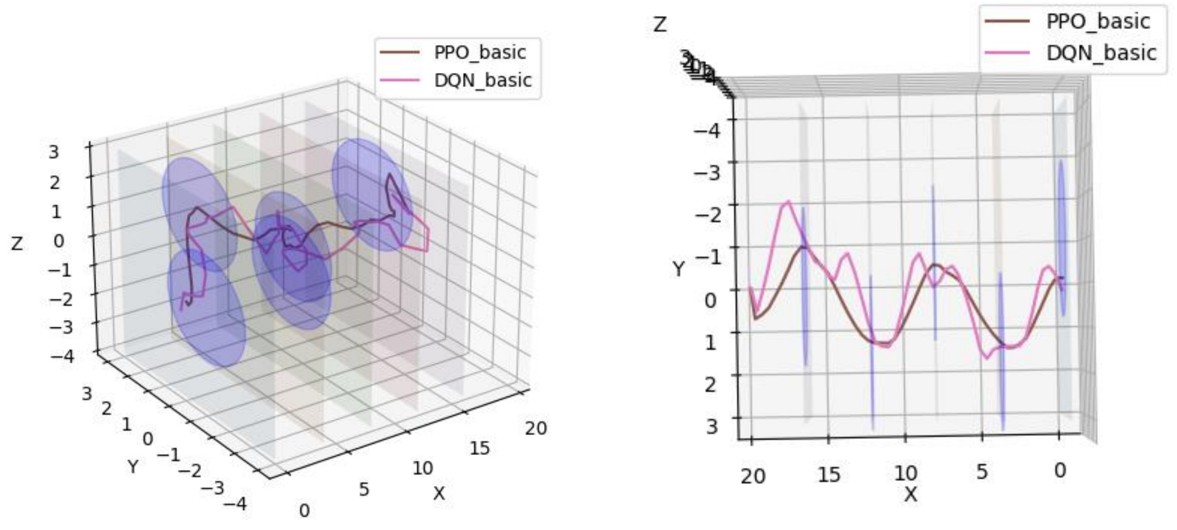


Figure 13 – 3D view and top view of trajectory plots for PPO Basic vs. DQN Basic

After reviewing the performance of A2C and DDPG algorithms, it is evident that further in-depth study is unnecessary. Therefore, let us proceed with a comparison between PPO and DQN. In terms of performance, PPO generates a smoother and shorter path, while DQN produces a more complex and longer path, as clearly depicted in the plots.

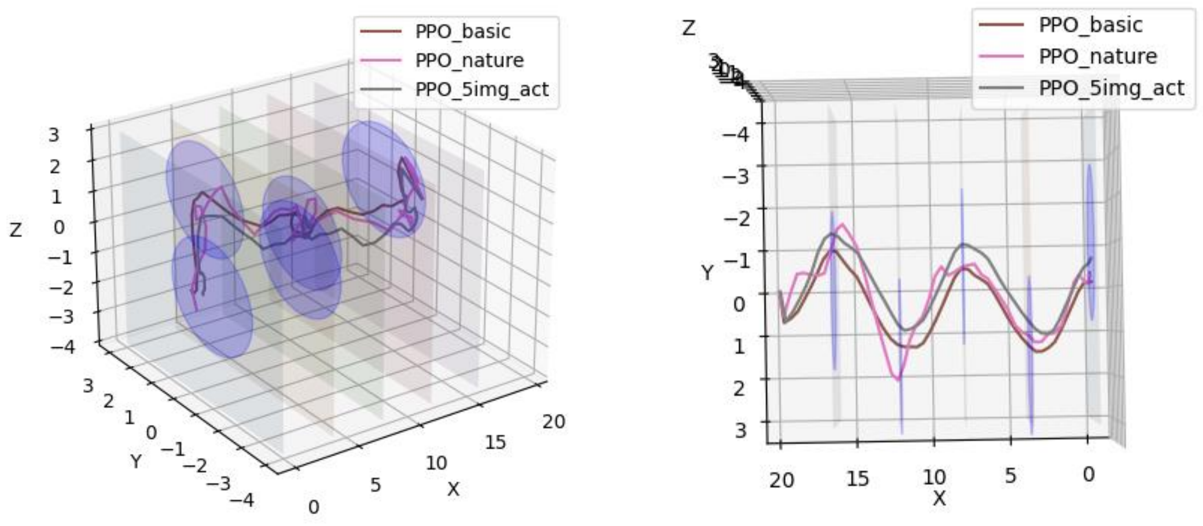


Figure 14 – 3D view and top view of trajectory plots for PPO Basic vs. PPO Nature vs. PPO 5_Img

A comparative evaluation of all the trained PPO models is conducted including our modified version that utilizes 5 grayscale stacked images as input, a modified action space, and reward function. Upon examining the plotted trajectories, it can be observed that our modified PPO model offers a slight improvement over the normal PPO model with BasicNet. Specifically, our modified model generates a more centralized path with fewer collisions with walls, while maintaining a comparable path length. These findings suggest that our proposed modifications could potentially lead to better performance in complex tasks with similar characteristics.

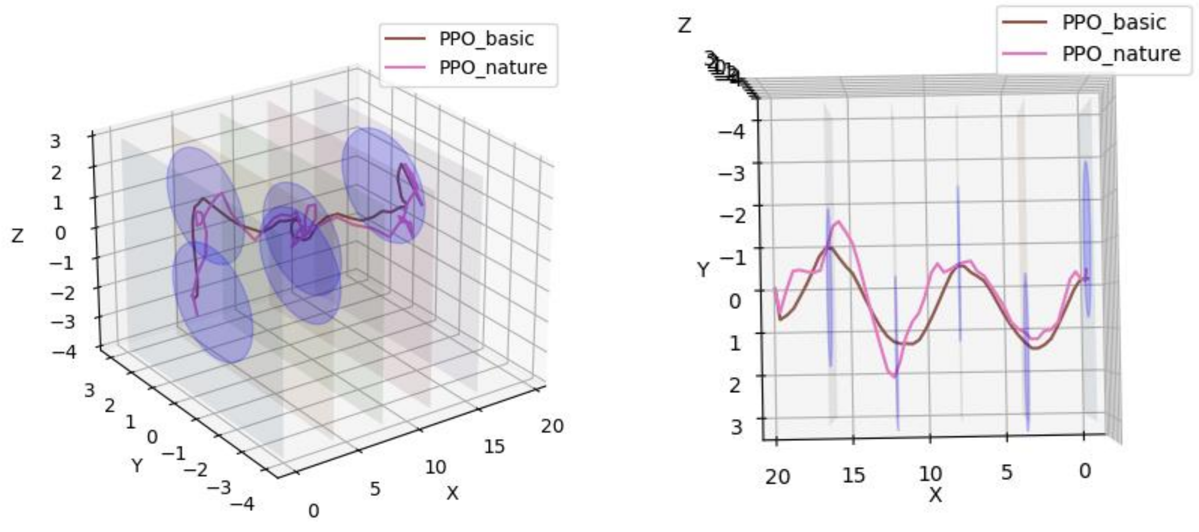


Figure 15 – 3D view and top view of trajectory plots for PPO Basic vs. PPO Nature

After comparing the performance of PPO with BasicNet and NatureNet, it can be concluded that the more complex BasicNet performs better than NatureNet. The generated path is smoother with BasicNet, while NatureNet generates longer paths. However, it should be noted that the choice between the two models ultimately depends on the specific needs of the application and the trade-off between complexity and performance. Additionally, further experimentation and tuning may lead to even better results for both models.

Flight Distance and Success Rate:

Table 2 – Flight Distance and Success Rate based on each Model

Model	**Flight Distance (m)**	**Success Rate (Over 100 test episodes)**
PPO with NatureNet	22.90 m	69 %
PPO with BasicNet	20.30 m	89 %
DDPG with NatureNet	0 m	0 %
DDPG with BasicNet	0 m	0 %
DQN with NatureNet	20.11 m	89 %
DQN with BasicNet	21.85 m	84 %
A2C with NatureNet	0 m	0 %
A2C with BasicNet	0 m	0 %
PPO with BasicNet, 5 image stack, Updated reward function, Different Action Space	20.07 m	98 %

From above table it can be concluded that PPO performs better than rest of the algorithms in terms of flight distance and Success rate.

5 Conclusion and Future Work

The aim of this study was to compare the performance of PPO, DQN, DDPG and A2C deep reinforcement learning algorithms in a virtual environment. According to the results, PPO and DQN were better at navigating through walls than DDPG and A2C. The PPO algorithm with NatureNet had a higher average reward value than PPO with BasicNet, but BasicNet showed higher rewards in some episodes. The average reward value for DQN was around -50, while for PPO it was around 0. The choice between using a continuous or discrete action space depends on the specific task. Altering the observation and action space had a positive impact on the performance of PPO (BasicNet). The path generated by DQN was longer and more complex, while PPO generated a shorter and smoother path. The modified PPO model, which used 5 grayscale stacked images as input, a modified action space, and reward function generated a more centralized path with fewer collisions with walls while maintaining a comparable path length.

There are several potential directions for further research. Firstly, the performance of the algorithms mentioned (PPO, DQN, A2C, and DDPG) can be evaluated in real-world scenarios, such as flight with dynamic obstacles. Additionally, more complex simulation environments can be used to experiment with different deep reinforcement learning algorithms, and the action set and reward function of the system can be improved. One promising area of research is the development of more efficient and effective algorithms for tasks with continuous action spaces. Another area of interest is exploring ways to combine the strengths of both continuous and discrete action spaces to achieve better performance. This is of particular interest because it may lead to smoother and safer motion.

Furthermore, future research could focus on investigating the effects of different network architectures, activation functions, and hyperparameters on the performance of reinforcement learning algorithms. Additionally, evaluating the performance of these algorithms on more complex environments can help researchers gain a better understanding of their capabilities and limitations.

References

[1] Kalidas, A. P., Joshua, C. J., Md, A. Q., Basheer, S., Mohan, S., & Sakri, S. (2023). Deep Reinforcement Learning for Vision-Based Navigation of UAVs in Avoiding Stationary and Mobile Obstacles. Drones, 7(4), 245. https://doi.org/10.3390/drones7040245

[2] Quan, L., Han, L., Zhou, B., Shen, S., & Gao, F. (2020). Survey of UAV motion planning. IET Cyber-Systems and Robotics, 1(1), 1-16. doi: 10.1049/iet-csr.2020.0004

[3] Tamizi, M.G., Yaghoubi, M. & Najjaran, H. A review of recent trend in motion planning of industrial robots. Int J Intell Robot Appl (2023). https://doi.org/10.1007/s41315-023-00274-2

[4] AlMahamid, F., & Grolinger, K. (2022). Autonomous Unmanned Aerial Vehicle Navigation using Reinforcement Learning: A Systematic Review. Journal of Intelligent & Robotic Systems, 102(1), 17-41. https://doi.org/10.1007/s10846-021-01581-4

[5] H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto, "Safe local exploration for replanning in cluttered unknown environments for microaerial vehicles," IEEE Robotics and Automation Letters, vol. 3, no. 3, pp. 1474–1481, 2018.

[6] E. Camci and E. Kayacan, "Learning motion primitives for planning swift maneuvers of quadrotor," Autonomous Robots, vol. 43, no. 7, pp. 1733–1745, 2019.

[7] E. Camci, D. Campolo, and E. Kayacan, "Deep reinforcement learning for motion planning of quadrotors using raw depth images," in Proc. Int. Joint Conf. Neural Netw., 2020, pp. 1–7.

[8] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3D Euclidean signed distance fields for on-board MAV planning," in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2017, pp. 1366–1373.

[9] E. Tal and S. Karaman, "Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness," in 2018 IEEE Conference on Decision and Control (CDC). IEEE, 2018, pp. 4282–4288.

[10] B. Kabas, "Autonomous UAV Navigation via Deep Reinforcement Learning Using PPO," *2022 30th Signal Processing and Communications Applications Conference (SIU)*, Safranbolu, Turkey, 2022, pp. 1-4, doi: 10.1109/SIU55565.2022.9864769.

[11] O. Bouhamed, X. Wan, H. Ghazzai and Y. Massoud, "A DDPG-based Approach for Energy-aware UAV Navigation in Obstacle-constrained Environment," *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, New Orleans, LA, USA, 2020, pp. 1-6, doi: 10.1109/WF-IoT48130.2020.9221115.

[12] X. Han, J. Wang, Q. Zhang, X. Qin and M. Sun, "Multi-UAV Automatic Dynamic Obstacle Avoidance with Experience-shared A2C," *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Barcelona, Spain, 2019, pp. 330-335, doi: 10.1109/WiMOB.2019.8923344.

[13] H. Oleynikova, D. Honegger, and M. Pollefeys, "Reactive avoidance using embedded stereo vision for mav flight," in 2015 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2015, pp. 50–56.

[14] B. T. Lopez and J. P. How, "Aggressive 3-d collision avoidance for highspeed navigation," in 2017 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2017, pp. 5759–5765.

[15] S. Stevˇsi´c, T. N¨ageli, J. Alonso-Mora, and O. Hilliges, "Sample efficient learning of path following and obstacle avoidance behavior for quadrotors," IEEE Robotics and Automation Letters, vol. 3, no. 4, pp. 3852–3859, 2018.

[16] B. Penin, P. R. Giordano, and F. Chaumette, "Vision-based reactive planning for aggressive target tracking while avoiding collisions and occlusions," IEEE Robotics and Automation Letters, vol. 3, no. 4, pp.3725–3732, 2018.

[17] F. Sadeghi and S. Levine, “Cad2rl: Real single-image flight without a single real image,” arXiv preprint arXiv:1611.04201, 2016.

[18] V. Blukis, N. Brukhim, A. Bennett, R. A. Knepper, and Y. Artzi, “Following high-level navigation instructions on a simulated quadcopter with imitation learning,” arXiv preprint arXiv:1806.00047, 2018.

[19] Yan, P., Bai, C., Zheng, H., & Guo, J. (2021). Flocking Control of UAV Swarms with Deep Reinforcement Learning Approach. IEEE Access, 9, 127558-127567. https://doi.org/10.1109/ACCESS.2021.3102889

[20] Han, X., Qin, X., Wang, J., Sun, M., & Zhang, Q. (2019, June). Multi-UAV automatic dynamic obstacle avoidance with experience-shared A2C. In Proceedings of the Second International Workshop on Mobile Edge Networks and Systems for Immersive Computing and IoT (pp. 17-22).

[21] O. Bouhamed, X. Wan, and H. Ghazzai, "A DDPG-based Approach for Energy-aware UAV Navigation in Obstacle-constrained Environment," in 2020 IEEE 6th World Forum on Internet of Things (WF-IoT), 2020, pp. 1-6, doi: 10.1109/WF-IoT48130.2020.9221115.

[22] Z. Huang and X. Xu, "DQN-Based Relay Deployment and Trajectory Planning in Consensus-Based Multi-UAVs Tracking Network," *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, Montreal, QC, Canada, 2021, pp. 1-7, doi: 10.1109/ICCWorkshops50388.2021.9473735.

[23] Z. Cui, Y. Wang, UAV Path Planning Based on Multi-Layer Reinforcement Learning Technique, IEEE Access 9 (2021) 59486–59497.

[24] Key concepts in RL, https://spinningup.openai.com/en/latest/spinningup/rl_intro.html