Autonomous navigation of Unmanned Aerial Vehicles in GPS-Denied Environments using Reinforcement Learning

*Abstract*— With the increasing usage of Unmanned Aerial Vehicles, their indoor applications have also seen a rise. The indoor environments, however, are challenging to navigate using the traditional methods, which rely on GPS signals. This requires the development of alternate information sources that can gather the required information from the environment to perform navigation without collisions. In this study, we put forth a method to use deep reinforcement learning to perform indoor navigation of Unmanned Aerial Vehicles utilizing AirSim and a simulated environment. The deep reinforcement learning system captures input from the indoor environments using the inputs from the cameras onboard. It performs the navigation using the proposed system, which implements DQN, an adaptive learning rate, a discrete decision system to perform independent decision-making for each camera, and a combinatory neural network to choose the best action from both inputs. The proposed system leads to a better learning model than the implemented systems, the only system that solved all the environments while achieving a 92% increase in average episode reward and reducing the number of iterations required by 96%.

Keywords—Reinforcement learning, UAV, Drones, Non-GPS

# Introduction

Unmanned Aerial Vehicles (UAVs) have been growing in popularity and widening in applications. Almost all applications require the drone to manoeuvre around places with space constraints. Multiple applications, including surveillance, reconnaissance, policing activities, and commercial applications like delivery, require the drone to be able to navigate and route the path of the drone. These pose a colossal problem since UAVs need high stability and control to do the task effectively.

The current solutions heavily depend on the coordinate inputs from the GPS sensors and, possibly, a combination with a LIDAR sensor to detect obstacles. This setup is suitable for navigation of unmanned aerial vehicles in outdoor environments where the GPS coordinates can be accurately acquired. This technique, however, poses a challenge when the navigation needs to be performed in an environment without support from a GPS. Navigation in such environments may need to occur in operations like mapping an indoor factory or industrial floor. Therefore, a better technique is required to navigate such environments without support from GPS or LIDAR systems. In these complex environments, an alternate input is required to perform autonomous navigation. A suitable alternative is to use the camera inputs from the drone. This method is chosen in this work, and the cameras provide a 3D depth image to the drone, allowing it to navigate the environment.

When it comes to navigation using cameras, many studies are showing and commending the work of Reinforcement Learning to navigate UAVs in non-static or unknown environments. Recent findings and the emergence of various parts of Artificial Intelligence, namely Deep Learning and Reinforcement learning techniques based on convolutional neural networks or artificial neural networks, have shown great results in all phases of its domain. For the sake of navigating, using these advanced technologies on a drone can be challenging but very helpful. The integration of machine learning algorithms has opened a new sphere of work wherein models prebuilt on large chunks of datasets can be used directly with drones, increasing the area of effect and the application range of drones. The reinforcement learning model performs exploration using real-time learning methodologies instead of pre-trained learning.

The aspect of onboard sensors and running an expensive computation on the drone is a difficult task. It also is a problematic issue to troubleshoot and adjust. To reduce the complexity, increase the margin for continuous change, and make precise adjustments, the solution is done on AirSim. This simulator simulates indoor and outdoor environments for the drone to navigate. This simulator's advantage is that it can run on a high-performance GPU and use the Unreal engine, plus it has multiple environments to test and train the UAV.

Using the reinforcement learning technique for performing navigation, unmanned aerial vehicles can learn to navigate the environments themselves without having to be trained on datasets that may not be related to the target environment. In this technique, the simulation can also be modified to mimic an environment that must be solved. The proposed system can, therefore, allow drones to navigate in an environment based solely on camera inputs without requiring GPS coordinates. Training the agent to learn using reinforcement learning benefits from adaptive, intuitive learning, where the models can learn with minimal supervision.

# Related Works

The drone navigation system proposed by Xue et al. [1] mentions the following shortcomings in previous systems

### A few solutions use supervised learning for navigation, which is feasible, but implementing the same in drones would be difficult. This is due to the large amount of data required to be generated and the datasets required to train the models, which would be expensive and difficult to collect. Reinforcement learning is a possible solution to this shortcoming. Still, many solutions are said to have used discrete action spaces, i.e., the action is pre-selected, and the drone only has control of the direction but not the velocity. To overcome these problems, authors proposed a method that uses continuous action spaces, i.e. These reinforcement learning agents will be able to control the direction and velocity of the drone in the environment. The proposed work uses AirSim with Unreal Engine to simulate the environment and train the reinforcement learning agent. For the current environment, only continuous actions are used, and the suitable algorithms for this are PPO, Soft Actor-Critic and DDPG. The three algorithms were evaluated, and the results are presented in Table 1.

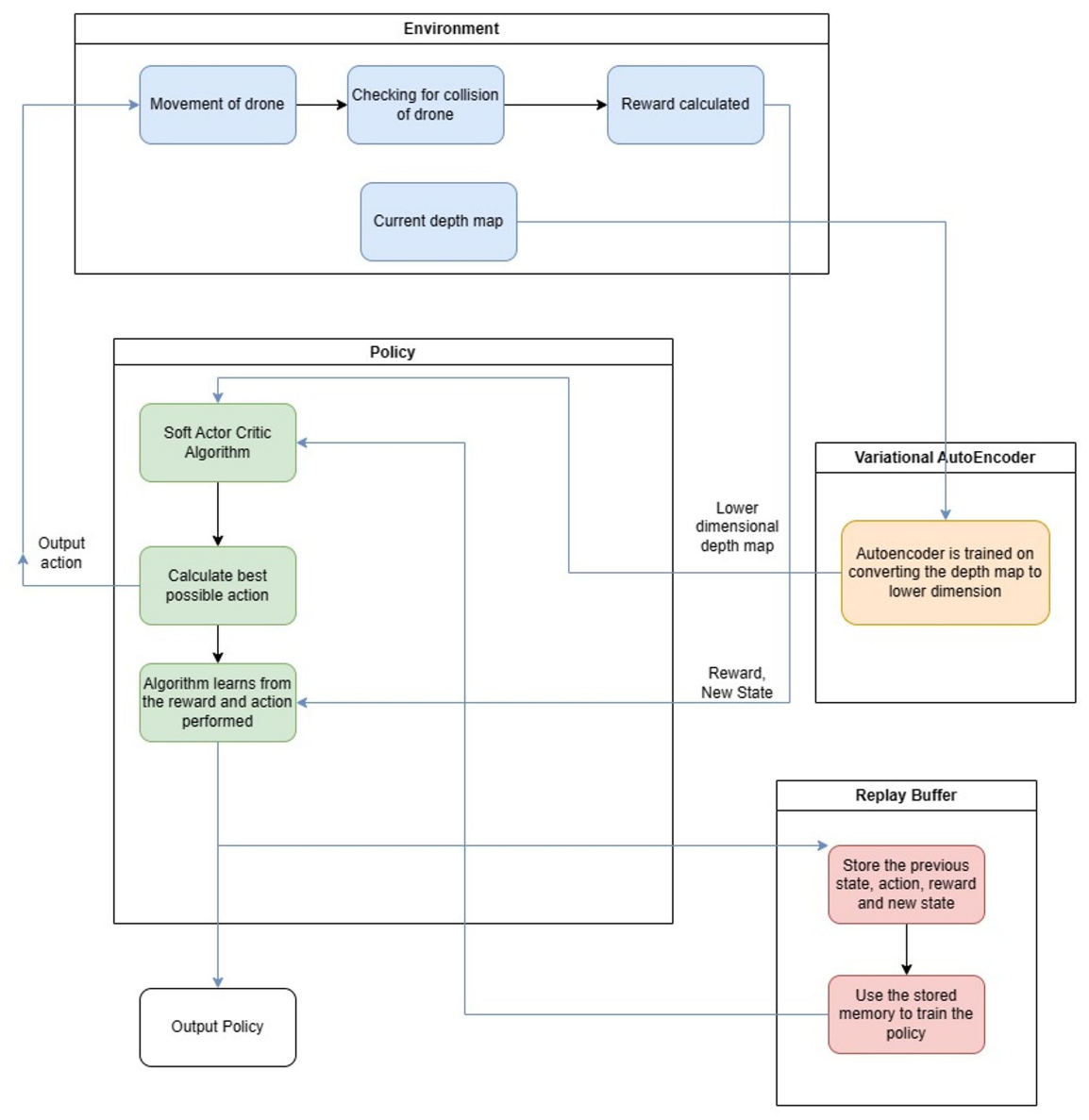
**Table 1.** Algorithm and the description of the systems

|  |  |
| --- | --- |
| **Algorithm** | **Description** |
| PPO | The algorithm required a large number of samples for considerable learning The algorithm may not capture the entire complex process of controlling drones |
| DDPG | DDPG is complex enough to be implemented for drone control. The algorithm however is a deterministic strategy and only chooses the best possible action in each state. This reduces the exploratory characteristic of the drones and reduces the chances of finding the most optimal path. |
| Soft Actor-Critic (SAC) | The soft actor critic is a modification on the actor critic method and has higher exploration compared to the traditional actor critic methods. This is ideal in case of the drone navigation as there are a large number of states and exploration could lead to higher chances of selecting the ideal path. |

For the agent's training, the Soft Actor-Critic algorithm is used with input as the depth maps generated from the unreal engine environment. The depth maps are accessed from the environment and then passed through a Variational Auto Encoder, which generates a more straightforward representation of the depth maps, which is given as input. Once the Variational Auto Encoder can generate simpler depth maps with good accuracy, the more straightforward depth maps are fed as input to the Soft Actor-Critic algorithm, which learns to navigate the drone across the obstacles.

The system also used a replay buffer, which can store rewards, actions, states and next states. These are used for sampling during training and are split into two halves: one for storing the episodes or actions where the agent avoided the obstacles successfully and the other half for storing all the actions in general. Having a separate buffer for the actions that led to avoidance of obstacles is helpful as during the initial training episodes, the drone cannot avoid obstacles efficiently, and if the replay buffer only consists of episodes with collisions, the agent will not be able to learn efficiently and could lead to much slower convergence.

The setup is trained for about 8000 epochs, and the drone achieved a 90% obstacle avoidance rate after the training. The setup was also compared to the TD3 algorithm by training for 4000 epochs, and the average reward of the trained TD3 algorithm is around three while the average reward of the proposed setup is about 7, almost two times the TD3 algorithm, showing the effectiveness of the proposed setup. Fig. 1 depicts the working of the system using the Variational autoencoder.



**Fig. 1** Flowchart of the system to navigate using support from Variational Autoencoder

A Deep Learning-based method to train drones in a simulated and mixed-reality environment was proposed by A. Devo et al. [2]. The mixed-reality environment bridges the gap between the simulation and real-life environments for better portability and adaptability to real-life navigation problems. The agent is not provided with complete information about the environment and its state at any particular time, and therefore, the problem comes under the partially observable Markov decision process.

The environment provides only a single RGB image at each timestep to the agent as state, and the agent has 11 discrete actions that it can perform. The reward is granted based on the number of visited cells or the Map Entropy, which keeps track of the visits to each cell. This is used to promote exploration of the map while providing a higher reward if the number of unvisited cells is less. Due to the ample state space, which cannot be represented in tabular format, a Deep Neural Network is to be used to perform the role of the RL agent. For this, the Agent-Critic method is used where there are two neural networks, the Actor and the Critic.

The agent was trained in three different environments,

* 1. Standard - A small, simple environment.
  2. Large - A large size environment with more obstacles.
  3. Realistic - The environment consists of realistic obstacles and realistic rooms using a mixed-reality framework.

The agent was trained on all three types of environments and could converge to almost 100% coverage of the maps. It took about 1800 epochs for Standard, 7000 epochs for Large and 1800 epochs for the realistic environments to achieve the 100% coverage rate.

The novelty of the work is the use of only a single RGB image to train the drone to explore environments with varying numbers of obstacles and sizes. It also uses a custom-defined actor-critic method, which utilizes a multiple input for the critic network to better estimate the value of Vπ.

The system implemented by Kalidas et al. [3] compares the performance of different reinforcement learning models for teaching an agent to avoid moving and static objects in the drone environment. Proximal Policy Optimization, Deep Q Network and Soft Actor-Critic are the algorithms considered. For the environment setup, AirSim was implemented with the Unreal Engine simulator. Five different environments are used in the training and testing process. For training, three environments are used, and for testing, two environments are used. The five environments consist of simple and complex environments with only static or moving obstacles like one of the environments has moving humans, which the drone must avoid. There are four discrete actions: move front, left, right, and back. In the case of rewards, reaching a goal is +100, and collision is -100. Moving away from the goal gives a negative reward, and moving towards the goal gives a positive reward. The input state or observation is the drone's front camera depth map. This depth map is then passed to a convolutional neural network for the policy, which outputs the ideal action.

The other method uses the DQN to take the depth map as the input and output the best action by estimating the Q value of the actions. This method also implements a replay buffer to store experiences and retrain the model on these experiences during training. The flowchart of the system is given in Fig. 2

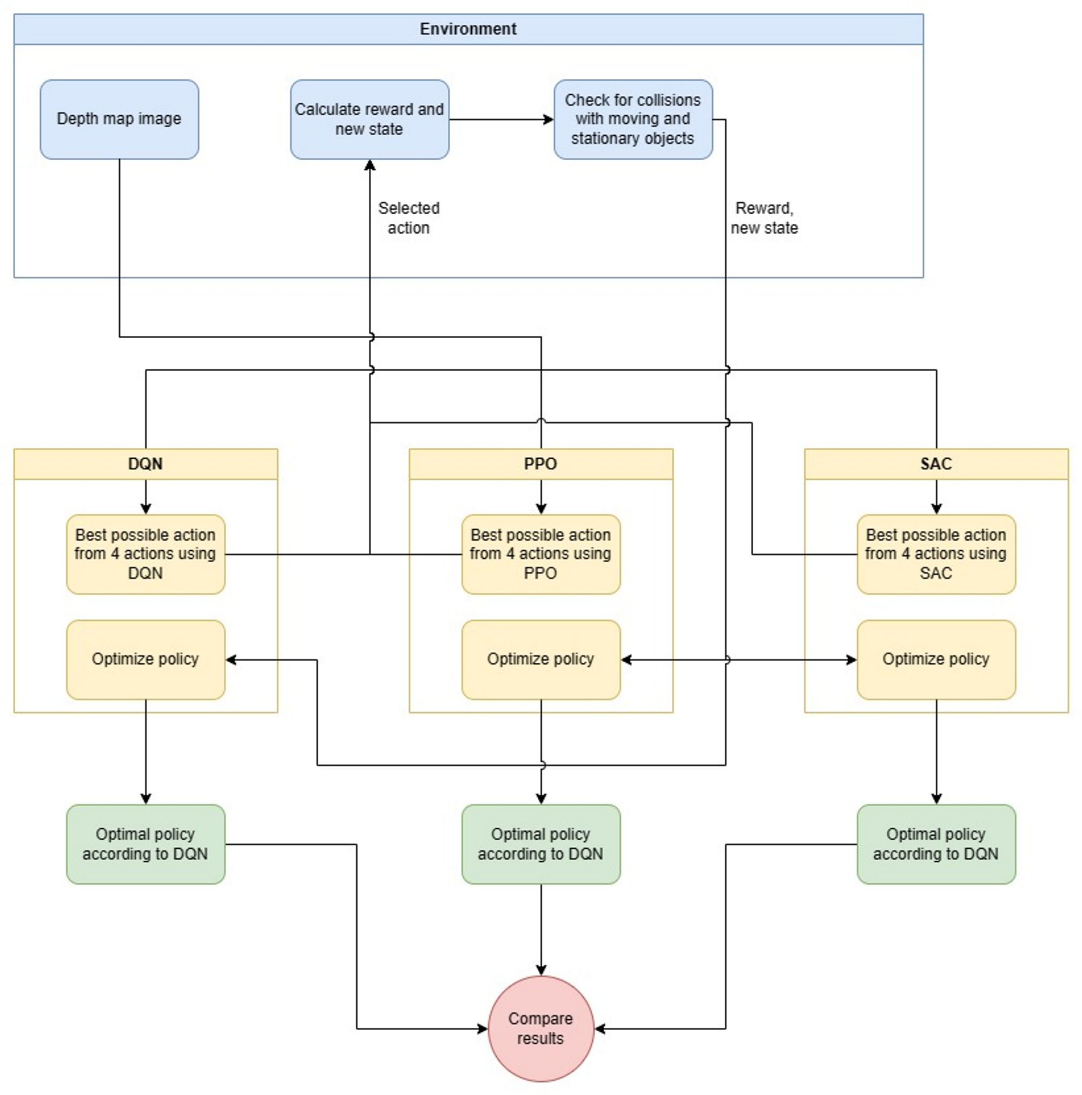


Fig. 2 Flowchart of the ensemble model to utilize multiple models

Another method implemented is proximal policy optimization, which trains an actor-critic pair to generate a new policy that is not significantly different from the previous policy to have continuous but stable improvements. The soft Actor critic method is also implemented to solve the problem, and the algorithm is similar to the soft actor-critic method implemented by the previous works, which uses the experience replay to train the model. These three algorithms were trained for 100,000 steps, and the results showed that each of the algorithms was able to learn the difficulties in the environment and overcome them over time, as all the algorithms showed an improvement in average reward during the training. The Soft Actor-Critic was the best-performing of the three, primarily due to the adversarial setup and the algorithm's off-model feature. The PPO performed the worst due to its slow learning rate and slower adaptation to newer and better policies. Overall, it could be observed that the algorithms could perform navigation through complex environments with stationary and moving obstacles and could be promising baseline metrics for the navigation of drones in environments that only used depth maps.

The system implemented by Tu et al. [4] uses reinforcement learning to perform tasks of path planning and obstacle avoidance in environments with obstacles. The environment does not use a GPS but is trained to work outdoors, too. The first method implemented is Deep Q Learning with experience replay to train the drone to navigate the environment. The drone has 25 available actions in a grid format in front of it; each action cell is calculated by deriving the pitch and yaw angle based on the cell's position in the grid. Therefore, the drone can move towards any of the cells in the 5x5 grid by appropriately modifying the yaw and pitch angles. For the input, the drone is fed an RGB image or a depth image, which the agent can use to derive the 5x5 grid with Q values assigned to each action, and the action with the highest Q value is chosen.

This environment was implemented using AirSim and Python, and a custom environment was created using Unreal Engine. Three experiments were performed in the work; the first was the path planning test, where the drone’s capability to perform path planning was tested using the SARSA algorithm and the Q-learning algorithm. After 5000 iterations, the results were compared, and the Q learning algorithm performed better than the SARSA algorithm in terms of the number of steps performed to reach the goal. The second experiment tested the capability to avoid obstacles. For the input, the agent is provided a processed depth image, which represents the depths using yellow and pink colors primarily where yellow represents closer obstacles and pink represents far away obstacles. The intensity of the colors was decided based on the distance of the drone from the obstacles. From the given input, the agent must learn to avoid the obstacles in the environment, which are trees in this case, and select an appropriate action from the 5x5 grid of action spaces. The algorithms compared were the Q learning and the SARSA algorithm. In this case, the Q learning algorithm outperformed the SARSA algorithm as it was 30% lower in computation time while taking fewer steps to achieve the same objective as the SARSA. An ultrasonic sensor also outputs the distances from the objects directly in front of the drone. The distance from the ultrasonic sensor is fed to the algorithm to avoid any obstacle that may have gone unnoticed or to change course in case an obstacle is detected. The Q learning algorithm and the classical SARSA algorithm were compared for path planning and obstacle avoidance tasks. The Q learning algorithm with the Experience replay performed better than the SARSA algorithm in these tasks and is proposed as a better alternative to the SARSA algorithm.

The system proposed by Miera et al. [5] implemented a reinforcement learning technique to perform drone navigation using LIDAR data from the sensor on the drone. The environment was implemented in Gazebo and using ROS. For the drone training, two different simulators were used; the first one was custom-built using Python to generate a 2D map with trees as obstacles represented by circles of varying radius. The drone would be spawned on the map, and the LIDAR information is simulated in the environment by calculating the distance to each circle from the drones. The second simulator uses Gazebo to create a 3D simulation of the environment where each tree is a long cylinder with a different radius, and the drone is required to navigate the obstacles. The agents were trained on a Software In the Loop mode of PX4 and could be moved to the actual hardware if required. From the algorithms that could be used for this purpose, the PPO algorithm was chosen for implementing the proposed system. The LIDAR readings would be fed to the PPO algorithm, which would then navigate the environment. The agent aims to navigate the X-axis with the highest possible velocity while not crashing into the obstacles. The reward system for the environment is as follows.

-0.25 for going very close to an obstacle

-1.5 for colliding with a tree

-0.1 \* dy if the drone moves away from the intended axis of movement, where dy represents the distance between the main axis and the current axis.

0.8 \* vx Reward is given for travelling on the intended axis, with vx representing the drone's speed.

The agent was trained in the environment using the PPO algorithm with a Multi-Layer Perceptron. After training, the test flights were conducted using software and hardware, such as running the agent on an NVIDIA Jetson and using an RPLIDAR fitted onto the drone. In the simulated test flights conducted, the highest successful rate is 91%, where a flight is considered successful if it travels the map without getting close to an obstacle. In real-world test flights, the highest successful flight rate is 80%, where the drone did not collide or get close to the trees. The work demonstrates the feasibility of using a simulator to train the drones and then migrating it to a real-world drone. The results show that it is possible to train a drone for the real world using simulators, but there is a significant gap in the results. In the simulated flights, the maximum achievable accuracy was 91%, whereas, in the real world, the drone had a maximum success rate of 80%, showing a large gap that would still have to be overcome to be perfectly suitable.

The proposed system by Chen et al. [6] is a well-designed signal sensor system and robust terrain understanding model that infers both the flatness and the safeness of the ground to overcome the problems due to the complexity of environments. Using a UAV platform equipped with a high precision LIDAR system and proposing a model named Terrain Net to achieve a terrain understanding and guarantee the safe landing of UAV in unknown environments. The model was trained using 10 actual flight records in different environments using onboard sensors. The model is trained using Adam optimizer with the backbone of ResNet-18, which has a depth of 18. The comparative study against the existing methods stated that with and without LIDAR input, the RMSE consistently decreases. As the density of the LIDAR increases, every metric improves, and the depth map is more fine-grained. ResNet-101 gave better results with a maximum accuracy of 98%. TerrainNet's landing site selection strategy improves accuracy significantly (from 86% to 98%) by incorporating semantic information. This helps the UAV avoid obstacles and choose reliable ground. Precise depth mapping aids in identifying small obstacles and altitude changes, further enhancing safety in complex environments.

A Reinforcement Learning-based system to calculate a UAV swarm’s optimal flight path was proposed by Puento-Castro et al.[7]. The methodology includes building two types of ANN models, one global and one ANN per layer. on chosen configuration and then employ and train according to the map. The base algorithm used was Hill-Climbing, and rewards were provided based on it. The optimizer used was RMSProp. The battery estimation and performance measures were also calculated; no realistic difference was found. The paper proposes a system capable of calculating the shortest flight time using Q-learning techniques. Experimental results indicate that a global ANN is the better choice because of its high speed on larger maps but slower on smaller maps. The larger the map, the more path combinations the ANN can evaluate. The paper does not work on 3D maps in which movements such as pitch, yaw and roll are possible. All the models are generic systems with future commercial potential.

A system proposed by Fu et al.[8] focuses on developing an intelligent agent that can navigate in the 3D environment using only visual input in an end-to-end manner. There is an introduction to a goal-conditioned reinforcement learning framework for vision-based UAV navigation. The authors developed an Enhanced DRL agent with dynamic relative, extra action penalty and non-sparce reward to tackle UAV navigation. vanilla DQN and MEDQN in Blocks and Neighborhood environments. The reward system is based on distance parameters. The proposed memory-enhanced DRL model allows memory to escape from obstacle dilemmas by considering observations and historical data. By performing experimental evaluation in AirSim, they showed a higher success rate with fewer training steps. The concepts not covered by this paper were the UAV’s better and smoother path planning on real-time drones and how accurate the model would be in case of highly dense objects and hurdles.

The main crux of the system implemented by Villota et al.[9] is to apply artificial intelligence to autonomously guide a drone to a certain point. The project is based on AirSim powered by Unreal Engine, on which many parallel simulations were run, like threads and the reward functions used to evaluate the performance. The algorithm is A3C, which uses neural network architecture that asynchronously updates the shared model. It is also efficient since no GPU is required to run it, and the CPU performance is maximized. The successful implementation of the A3C algorithm applied to the UAV field in low-performance servers led to simulations launching without costly GPUs. Using this is the proposed strategy for future experiments. The control carried is very limited, and since it only uses positions, there is no robust drone control. No image processing is involved, and the different modifications of the training strategies have not been experimented on.

A learning-based algorithm to solve the problem of routing the drone with recharging stops was implemented by Ermağan et al.[10], which can be used in multiple applications, including agriculture and surveillance. The proposal is to formulate an approach to improve the numerical efficiency of drone routing. Three major contributions from the paper were a new integer programming formulation, an innovative approach that learns from features of high-quality solutions, and numerical experiments to test the suggested method’s effectiveness. The results of the numerical experiments with four groups of instances show that the classification algorithms can effectively identify the features that determine the timing and location of the recharging visits, and L&F generates energy-feasible routes in a few seconds with around a 5% optimality gap on the average

# Proposed System

Reinforcement learning is a powerful machine learning technique successfully applied to many domains, including robotics and drone control. We have utilized the system to solve the problem of navigation in GPS-denied environments, where the drone cannot depend on position coordinates for movement. As an input to guide navigation, two cameras are used, one on the front and the other on the back, which provide depth images from the camera’s perspective. The parameters for the cameras used in the simulation are shown in Table 2.

**Table 2.** Camera parameters

|  |  |
| --- | --- |
| Parameter | Value |
| Image Size | (144,256) |
| Field of vision (FOV) | 120º |
| Image Type | Depth Perspective |
| Auto Exposure Speed | 100 |
| Motion Blur Amount | 0 |

AirSim is used to set up the drone control framework with the environment built using Unreal Engine. A custom gym class implementation is built using Python to interface with the drone in the environment. The class is responsible for performing actions and returning the state inputs to the model for learning. Using the custom Gym class, it can be used to perform learning using predefined algorithms. The proposed system was compared to algorithms implemented in the StableBaselines3 package to form a baseline metric. The description of the algorithms implemented along with their working are given below

## Q-Learning Algorithm

### Initialize the Q-Table: The first step in the Q-Learning algorithm is to initialize the Q-table with zeros or small random values.

### Observe the State: The agent observes the current state of the environment.

### Select an Action: The agent selects an action to take in the current state using an exploration-exploitation strategy, such as the epsilon-greedy strategy.

### Observe the Reward and Next State: The agent receives a reward and observes the next state of the environment based on the selected action.

### Update the Q-Table: The Q-table is updated based on the reward received and the maximum expected cumulative reward for the next state, which is calculated using the Bellman equation.

### Repeat: Steps 2 to 5 are repeated until the agent reaches a terminal state or a predetermined number of iterations.

### Evaluate the Policy: Once the Q-Learning algorithm has converged, the policy can be evaluated by selecting the action with the highest Q-value for each state.

Pseudocode:

Initialize (s, a) arbitrarily

Repeat (for each episode):

Initialize S

Repeat (for each step of episode):

Choose a from S using policy derived from Q (e.g., E-greedy)

Take action a, observe I,S'

Q(s, a) <-- Q(s, a) + a [r + y max,Q(s', a') α - Q(s, a)]

s <-- s';

until s is terminal

The parameters used in the Q-value update process are:

α - the learning rate, set between 0 and 1. Setting it to 0 means that the Q-values are never updated; nothing is learned. Setting a high value such as 0.9 means that learning can occur quickly.

γ - discount factor, also set between 0 and 1. This models the fact that future rewards are worth less than immediate rewards. Mathematically, the discount factor must be set to less than 0 for the algorithm to converge.

maxγ - the maximum reward attainable in the state following the current one. i.e the reward for taking the optimal action thereafter.

The Q-Learning algorithm can be represented as a graph, where each node represents a state-action pair, and each edge represents the update to the Q-value based on the reward received and the maximum Q-value for the next state. The graph can help visualize the learning process and identify patterns and relationships that may be difficult to discern from raw data alone. Overall, the Q-Learning algorithm is a powerful tool for learning the optimal policy for an agent in an uncertain environment, and the Q-Table graph can help visualize and understand the learning process.

## Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm that has gained popularity recently due to its simplicity and efficiency. PPO is a policy optimization algorithm that iteratively updates a parameterized policy by maximizing the expected cumulative reward.

### Initialize the Policy: The first step in the PPO algorithm is to initialize the policy with random or pre-trained weights.

### Collect Trajectories: The agent interacts with the environment by following the current policy and collecting a set of trajectories, which are sequences of state, action, and reward pairs.

### Compute Advantage Estimate: The advantage estimate estimates the value of taking a particular action in a particular state compared to the average value of all possible actions in that state. This can be computed using various methods, such as generalized advantage estimation (GAE).

### Compute Surrogate Objective: The surrogate objective measures how well the updated policy performs compared to the old policy. The surrogate objective is computed by taking the ratio of the updated policy probability to the old policy probability, multiplying it by the advantage estimate, and applying a clip function to prevent large updates.

### Update the Policy: The policy parameters are updated using an optimization algorithm, such as stochastic gradient descent (SGD) or Adam, to maximize the surrogate objective.

### Repeat: Steps 2 to 5 are repeated until the policy has converged or a predetermined number of iterations have been completed.

Pseudocode

Input: initial policy parameters θ0, initial KL penalty β0, target KL-divergence δ

for k = 0, 1, 2, ... do

Collect the set of partial trajectories Dk on policy πk = π(θk)

Estimate advantages Â(t)πk using any advantage estimation algorithm

Compute policy update

θk+1 = arg max θ Lθk (θ) − βkD̄KL(θ||θk )

by taking K steps of minibatch SGD (via Adam)

if D̄KL(θk+1||θk ) ≥ 1.5δ then

βk+1 = 2βk

else if D̄KL(θk+1||θk ) ≤ δ/1.5 then

βk+1 = βk /2

end if

end for

PPO can be visualized as a graph that shows the optimization process of the policy. The graph typically includes the policy parameters, the loss function, and the optimization algorithm used to update the policy parameters. Overall, PPO is a powerful and efficient reinforcement learning algorithm that can be used to learn the optimal policy for an agent in various environments. Using the advantage estimate and surrogate objective makes PPO more stable and efficient than other policy optimization algorithms.

## Maskable PPO

Maskable PPO is a variant of the Proximal Policy Optimization (PPO) algorithm that was introduced by Jacob Buckman et al. in 2018. The basic idea behind maskable PPO is to use a masking mechanism that allows the agent to ignore certain observations or actions during training selectively. In traditional PPO, the agent receives observations and chooses actions based on those observations. However, certain observations or actions may be irrelevant or harmful to the agent's performance in some environments. In such cases, training on these irrelevant features can lead to slower convergence and suboptimal performance.

Maskable PPO addresses this issue by allowing the agent to mask certain observations or actions during training selectively. The mask is a binary vector that indicates which observations or actions are included in the current step. During training, the agent uses the masked observations to update its policy and value functions while ignoring the unmasked observations.

In addition to the PPO algorithm, the following step is performed.

### Mask observations and actions: The agent applies a binary mask to each observation and action in the batch. The mask is determined based on which observations or actions are relevant to the current task.

Therefore, it is a powerful variant of the PPO algorithm that can improve the performance and speed of convergence in environments with irrelevant observations or actions. It has been used successfully in various applications, including robotics, game-playing, and autonomous driving.

## Recurrrent PPO

Recurrent PPO (Proximal Policy Optimization) is a variant of the PPO algorithm designed for reinforcement learning problems with sequential data, such as time-series data or video frames. Recurrent PPO uses a recurrent neural network (RNN) as a function approximator to handle the sequential nature of the data.

In addition to the PPO algorithm, the following step is performed.

#### Process observations with RNN: The observations in each trajectory are processed by an RNN to capture the temporal dependencies in the data. The RNN outputs a hidden state, which is used as input to compute the policy and value losses.

Therefore, it is a powerful algorithm for handling sequential data in reinforcement learning problems. Using an RNN allows the agent to capture the temporal dependencies in the data, leading to improved performance and faster convergence.

## Advantage Actor-Critic(A2C)

Advantage Actor-Critic (A2C) is a reinforcement learning algorithm that combines the advantages of policy-based and value-based methods. A2C is an extension of the Asynchronous Advantage Actor-Critic (A3C) algorithm, which allows for faster learning by training multiple agents in parallel.

### Initialize the Policy and Value Function: The first step in the A2C algorithm is to initialize the policy and value function with random or pre-trained weights.

### Collect Trajectories: The agent interacts with the environment by following the current policy and collecting a set of trajectories, which are sequences of state, action, and reward pairs.

### Compute the Advantage Estimate: The advantage estimate estimates the value of taking a particular action in a particular state compared to the average value of taking all possible actions in that state. This can be computed using various methods, such as generalized advantage estimation (GAE).

### Compute the Policy Gradient: The policy gradient is the gradient of the logarithm of the policy with respect to the policy parameters. The policy gradient is computed using the collected trajectories and the advantage estimate.

### Compute the Value Function Loss: The value function loss is the mean squared error between the estimated value function and the target value function, which is the sum of the reward and the estimated value of the next state.

### Update the Policy and Value Function: The policy and value function are updated using an optimization algorithm, such as stochastic gradient descent (SGD) or Adam, to minimize the policy gradient and the value function loss.

### Repeat: Steps 2 to 6 are repeated until the policy and value function have converged or a predetermined number of iterations have been completed.

A2C can be visualized as a graph that shows the computation graph for the policy and value function. The graph typically includes the state, action, reward, advantage estimate, policy gradient, and value function loss. Overall, A2C is a powerful and efficient reinforcement learning algorithm that can be used to learn an agent's optimal policy and value function in various environments. The use of the advantage estimate and the combination of policy-based and value-based methods make A2C more stable and efficient than other reinforcement learning algorithms.

## Deep Q-Network (DQN)

Deep Q-Network (DQN) is a reinforcement learning algorithm that uses a deep neural network to approximate the Q-value function. The Q-value function is the expected total reward for taking a particular action in a particular state and following the optimal policy thereafter.

### Initialize the Q-Network: The first step in the DQN algorithm is to initialize the Q-network with random or pre-trained weights.

### Collect Experience: The agent interacts with the environment by following the current policy and collecting a set of experiences, which are tuples of state, action, reward, and next state.

### Compute the Target Q-Value: The target Q-value is the reward sum and the next state's discounted maximum Q-value. This is used as the target for the Q-network during training.

### Compute the Q-Loss: The Q-loss is the mean squared error between the estimated Q-value and the target Q-value. This is used as the loss function during training.

### Update the Q-Network: The Q-network is updated using an optimization algorithm, such as stochastic gradient descent (SGD) or Adam, to minimize the Q-loss.

### Repeat: Steps 2 to 5 are repeated until the Q-network has converged or a predetermined number of iterations have been completed.

DQN can be visualized as a graph showing the Q-network computation graph. The graph typically includes the state, action, reward, next state, target Q-value, and estimated Q-value. Overall, DQN is a powerful and widely used reinforcement learning algorithm that has been applied to various environments and tasks. Using a deep neural network to approximate the Q-value function allows DQN to handle high-dimensional state spaces and learn complex policies.

## Quantile Regression DQN (QRDQN)

Quantile Regression DQN (QRDQN) is a variation of the DQN algorithm that incorporates the concept of quantile regression. Quantile regression is a statistical technique that estimates the conditional quantiles of a distribution, which can be used to estimate the uncertainty of the Q-value estimates in reinforcement learning.

### Initialize the Q-Network: The first step in the QRDQN algorithm is to initialize the Q-network with random or pre-trained weights.

### Collect Experience: The agent interacts with the environment by following the current policy and collecting a set of experiences, which are tuples of state, action, reward, and next state.

### Compute the Target Quantile Value: The target quantile value is computed using the Bellman equation and the quantile function. The quantile function estimates the distribution of the Q-value given the state and action, which allows for a better estimation of the uncertainty of the Q-value.

### Compute the Quantile Regression Loss: The quantile regression loss is the quantile regression loss between the estimated and target quantiles. This is used as the loss function during training.

### Update the Q-Network: The Q-network is updated using an optimization algorithm, such as stochastic gradient descent (SGD) or Adam, to minimize the quantile regression loss.

### Repeat: Steps 2 to 5 are repeated until the Q-network has converged or a predetermined number of iterations have been completed.

where,

QRDQN can be visualized as a graph showing the Q-network computation graph. The graph typically includes the state, action, reward, next state, target quantile value, and estimated quantile value. Overall, QRDQN is a powerful and promising reinforcement learning algorithm that can be used to learn an agent's optimal policy and value function in various environments. Using quantile regression allows for better estimation of the uncertainty of the Q-value, which can lead to more robust and reliable policies.

An important optimization in our system is the usage of the replay buffer. Replay buffer is a key component in many reinforcement learning algorithms, including DQN, DDPG, and TD3. The replay buffer is a memory buffer that stores past experiences of the agent. Each experience is a tuple of state, action, reward, and next state. The replay buffer allows the agent to learn from past experiences by randomly sampling experiences from the buffer and using them for training.

The benefits of using a replay buffer in reinforcement learning include the following -

### Efficiency: The replay buffer allows the agent to learn from past experiences without continuously interacting with the environment. This can save time and resources, especially when interactions are time-consuming or expensive.

### Memory: The replay buffer allows the agent to store and reuse past experiences, which can help the agent remember rare events and avoid forgetting important information.

### Learning Stability: Randomly sampling from the replay buffer during training helps to decorrelate the training data and reduce the impact of sequential correlations. This can lead to more stable and robust learning.

### Experience Replay: Experience replay allows the agent to learn from a diverse set of experiences, which can improve the agent's ability to generalize and adapt to new situations.

The replay buffer can be implemented as a queue or circular buffer data structure, where the oldest experiences are overwritten by the newest experiences once the buffer is full. The size of the replay buffer is an important hyperparameter that can affect the agent's performance. A larger buffer size can increase the diversity of experiences and improve learning stability, but it can also increase the memory requirements and training time.

Multiple environments and setups are created to test the robustness of the learning of each of the algorithms, which would allow for better insight into the performance. The descriptions of the environments are given in Table 3.

**Table 3.** Implemented features in the setups

|  |  |
| --- | --- |
| **Setup No.** | **Implemented Features** |
| 1 | Indoor environment with the goal directly in front |
| 2 | 1. Indoor environment with the goal directly in front 2. The reward is given based on distance from the goal, which is negative. Moving away from the goal gives a higher negative reward. |
| 3 | 1. Adaptive Learning rate where the learning rate is changed as the training progresses 2. Unbiased memory to store positive and negative steps 3. Dynamic replay memory training which performs offline training on the model for a random memory step between each episode. 4. Discrete Decision systems where a separate model is used for each of the inputs, front and back camera. The Q value of front movement is not affected by the inputs from back camera and vice versa to maintain control over the specific directions. |
| 4 | 1. Adaptive Learning rate where the learning rate is changed as the training progresses 2. Unbiased memory to store positive and negative steps 3. Dynamic replay memory training, which performs offline training on the model for random memory steps between each episode. 4. Discrete Decision systems where a separate model is used for each of the inputs, front and back camera. The Q value of front movement is not affected by the inputs from the back camera and vice versa to maintain control over the specific directions. 5. A combinatory neural network is used, which takes in the predicted Q values from the DQN networks of the camera inputs along with the reward gained in the previous step and the action taken in the previous step. The model then outputs the action to be taken. 6. The neural networks in both networks were also modified to make use of LSTMs for better memory. |

In order to improve the performances of the algorithms implemented, there were a few additions made on top which are as follows

## Adaptive Learning rate

The learning rate of the environment is made adaptive, and it helps achieve two objectives.

### Improving learning from the environment as the initial episodes mostly fail. A higher learning rate at the start can help the model learn more quickly and decrease the training time required.

### As the learning rate is adaptive, as training progresses, the learning rate is decayed to a small constant, thus preventing major changes to the model once a large training batch is completed.

## Unbiased memory

The memory module used to store the previously experienced episodes has an equal number of positive and negative episodes. This is done to use learnings from negative episodes while also learning to perform better using the positive ones.

## Dynamic memory training

During training, the memory is used to train the agent between episodes, i.e., offline training. In the offline training phase, the minimum number of memory steps trained is 32 but the agent is also trained randomly on a larger number of steps, as given in Table 4.

**Table 4.** Probability of the number of steps chosen in offline training

|  |  |
| --- | --- |
| **Number of steps trained on** | **Probability of selecting the number of steps** |
| 1000 | 2% |
| 128 | 10% |
| 64 | 20% |

## Discrete decision systems

There are two inputs to the agent, one from the front camera and one from the back camera. Ideally, the decision to move forward must be taken by input from the front camera; similarly, only the back camera must be responsible for making the decision to move backwards. To implement this, separate models are implemented for each of the inputs, and the resultant predicted Q values from the networks are used to choose the actions depending on the input, i.e., if Q values are predicted based on the inputs from the front camera, these Q values will not be affecting the Q value of selecting the action of moving back.

## Combinatory Neural Network

In these environments, as there are multiple sub-goals with rewards being given on movement towards the goal, a combinatory neural network was experimented with. The neural network would output the final Q values from the following.

### Q values of length 6 from each of the input models for the front and back camera

### Reward gained in the previous step

### Action taken in the previous step

This was done in the hopes that the neural network would be able to read the Q values and also utilize the reward and action information from the previous episode to judge the location of the sub-goal better while also preventing collisions. To aid the combinatory neural network, the environment returns information about the best action to be taken for the current state to achieve the goal. This is used to train the assimilating model.

# Experimental Setup

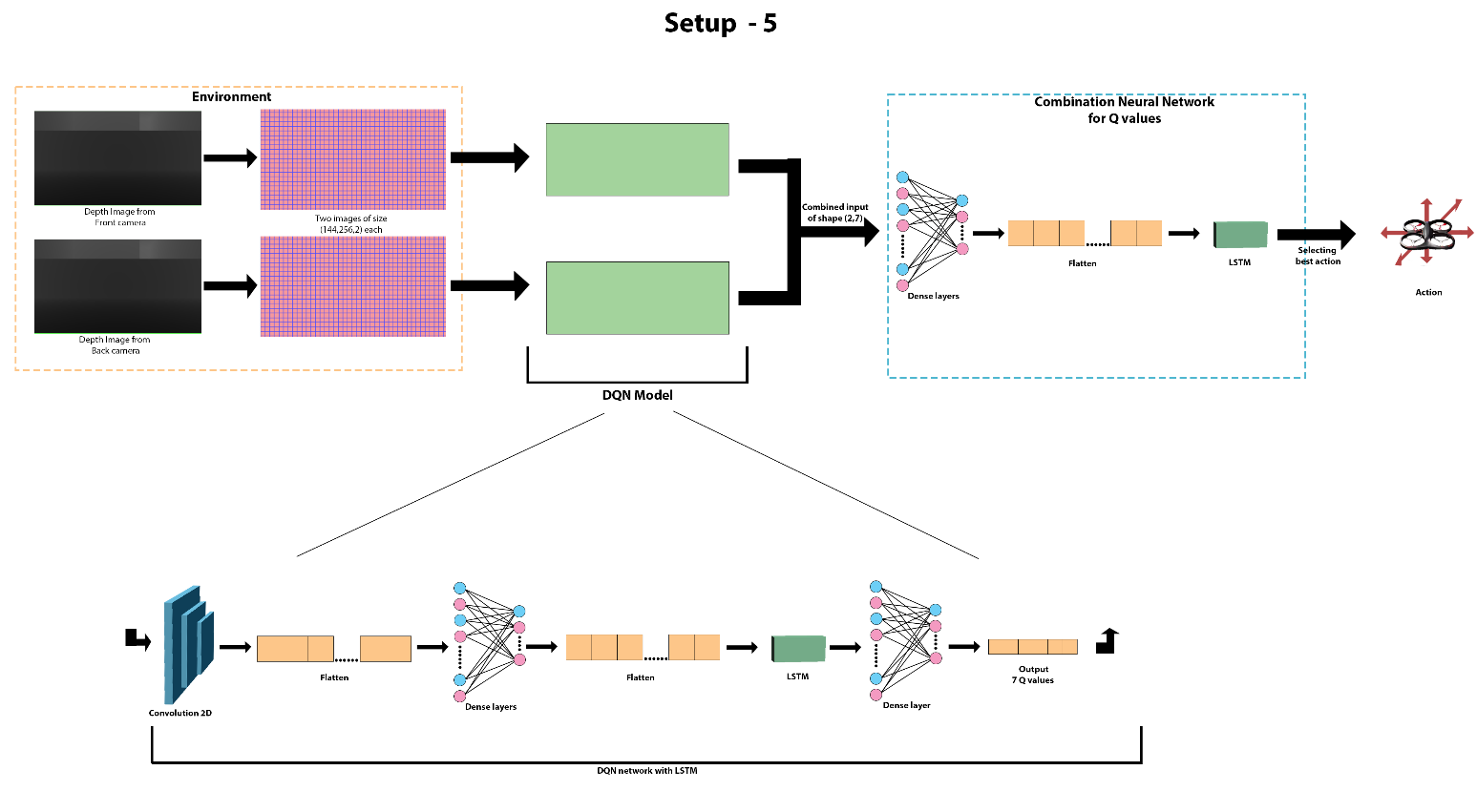
Our experimental setup involves the usage of AirSim along with simulated indoor environments. The simulated environments were made using Unreal Engine, and multiple environments with varying environment setups and configurations were used to allow the reinforcement learning algorithms to adapt to the different environments. The deep-learning -based algorithms are implemented using TensorFlow, and our setup also used Reinforcement learning algorithms from the Stable Baselines Python. A custom AirSim wrapper made with OpenAI gym in Python was used to implement the environment class to train the agents using the various reinforcement learning algorithms.

To carry out the proposed work, multiple setups were created, as described in Table 5. Fig. 3 and Fig. 4 depict the working of the Setup Nos. 3 and 4.

**Table 5.** Description of the Environment Setups

|  |  |  |  |
| --- | --- | --- | --- |
| **Setup No.** | **Difficulty** | **State Space** | **Reward definition** |
| 1  Indoor Environment | Simple | Only camera information from front and back in Dictionary format | -1 for all actions  -1000 for collision  1000 for reaching goal |
| 2  Indoor Environment | Simple | Array of 22 length with   * Current state – (x,y) * Goal state – (x,y) * Front image – (9,1) * Back Image – (9,1) | -x for distance x from goal  -1000 for collision  1000 for reaching goal |
| 3  Obstacle Environment - I | Medium | Inputs from two cameras passed onto a separate model | 10 – Movement towards goal  100 – Reach goal  - Distance – Movement away from goal  -1000 - Collision |
| 4  Obstacle Environment - II | Complex | Inputs only from both cameras used | 2 \* (level / No of levels) – Achieving sub-goal  -2 – Collision  5 – Movement towards goal |

# 

**Fig. 3** Flowchart of Obstacle Environment – I

**Fig. 4** Flowchart of Obstacle Environment - II

To measure the performance of the models, the environments are made slightly harder, and the algorithm is tried on the first environment. If it solves the environment, it is moved to the next environment and is trained similarly. A simple indoor environment with minimal obstacles during movement was used for the first environment. The drone was trained using the algorithm to perform the movement in the environment according to the setup information given above. The drone was given sub-goals which it could achieve by moving straight and then to the left. This was done to encourage drone exploration towards the intended goal and test the navigation accuracy of the algorithm being implemented. The environment used in setup 1 and 2 is an indoor environment and the images of the environment are given in Fig. 5 and Fig. 6.

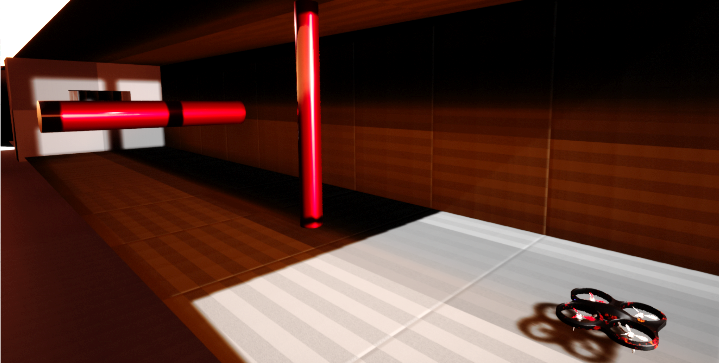
# 

**Fig. 5** Environment used in setup 1 and 2

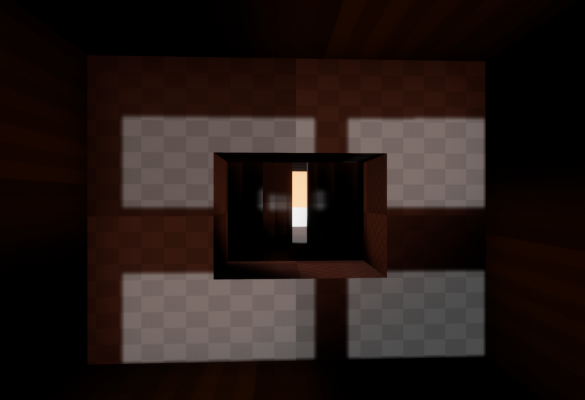
# 

**Fig. 6** Environment used in Setup 1 and 2

The second environment used is an environment with obstacles only in the front. This environment consists of horizontal and vertical bars, a hole in the wall and curving pillars which the drone has to manoeuvre around without colliding. If the drone can gain a good average reward or reach the goal state, it is classified as solved and moved to the next environment. The drone is only required to move straight in this environment. On moving through each set of obstacles, the drone is given a reward, and on traversing all obstacles, it is given a positive reward, and the episode ends. The obstacle environment used in setup 3 and 4 is shown in Fig. 7 and Fig. 8.

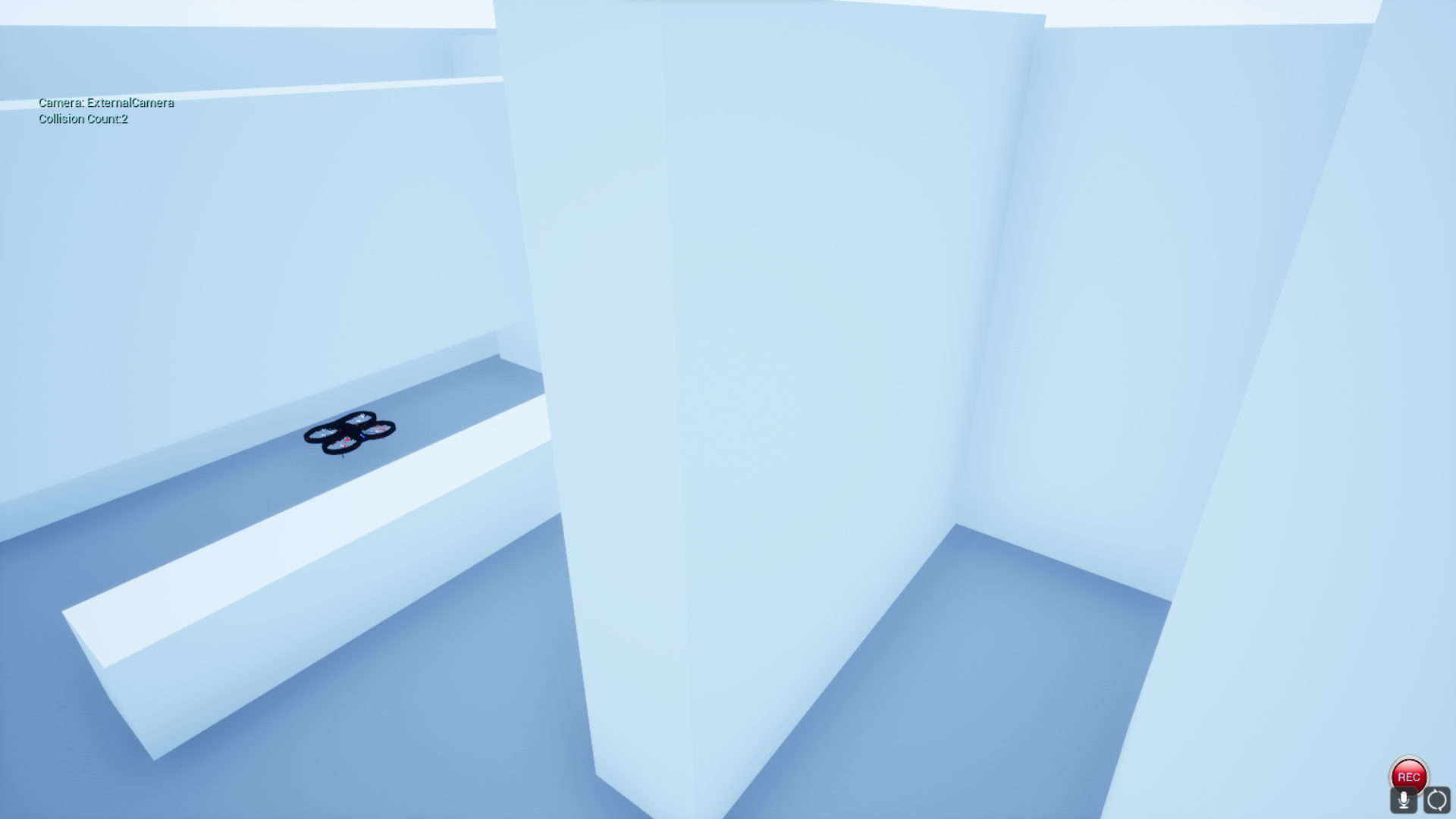


**Fig. 7** Environment used in Setup 4

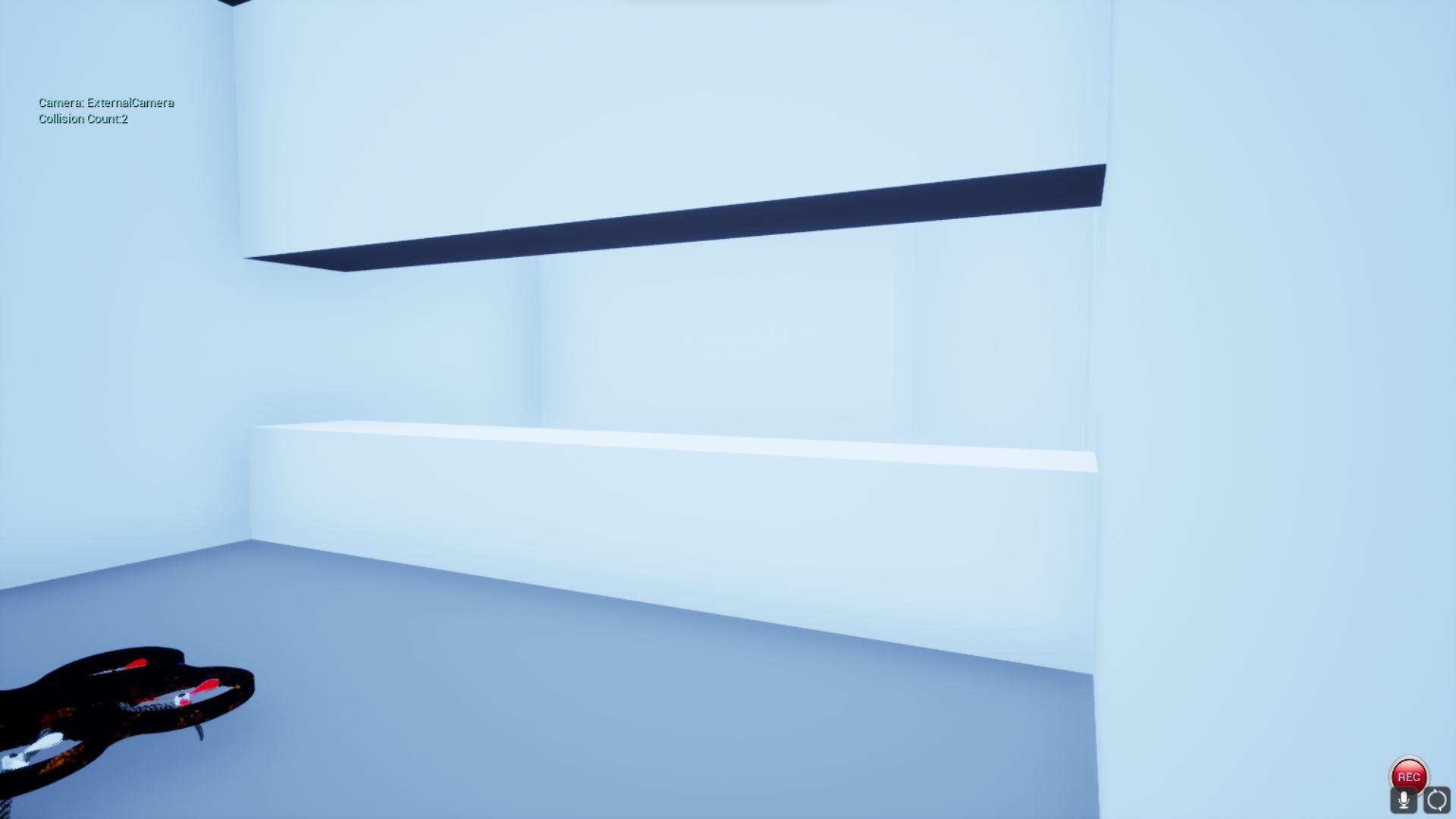


**Fig. 8** Environment used in Setup 3

The final environment is a custom-made environment which consists of a slit in the wall and then a small passage which is surrounded by walls. This environment is the most complex as there are multiple sub-goals to be achieved and requires higher control in all directions. As almost all wrong moves lead to a collision in the narrow passages, the drone requires precise control during the navigation. The environment used in setup 4 is shown in Fig. 9 and Fig. 10.



**Fig. 9** Environment used in Setup 4



**Fig. 10** Environment used in Setup 4

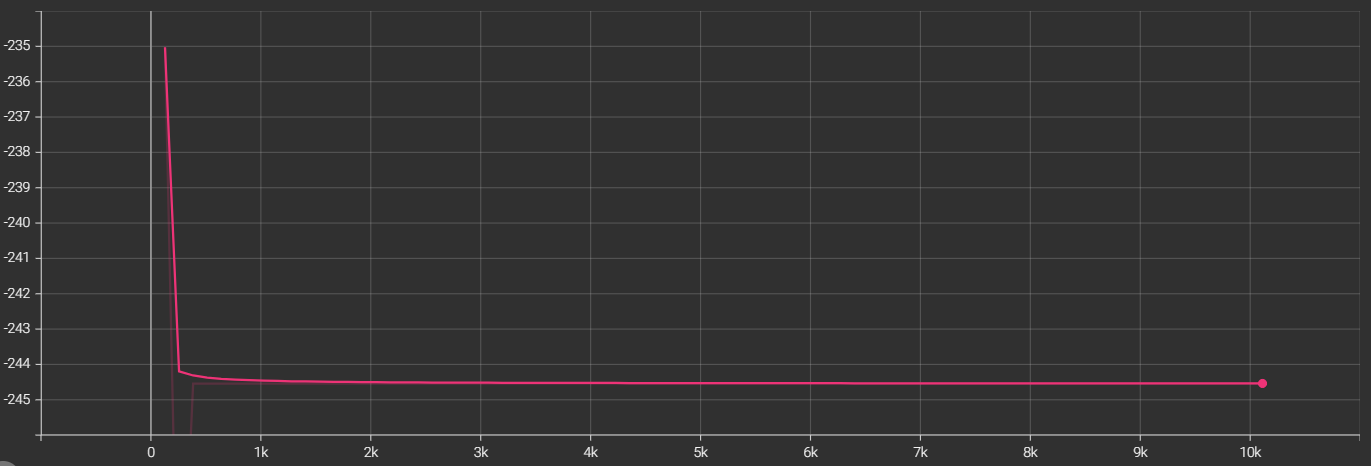
# Results

For the experimentation, we performed 10 runs as described in Table 6 using different algorithms and environments. The algorithms which are successful on the easier environments are moved to the tougher environment and the cycle continues. If the algorithm cannot perform in one of the environments, it is not considered for the next environment.

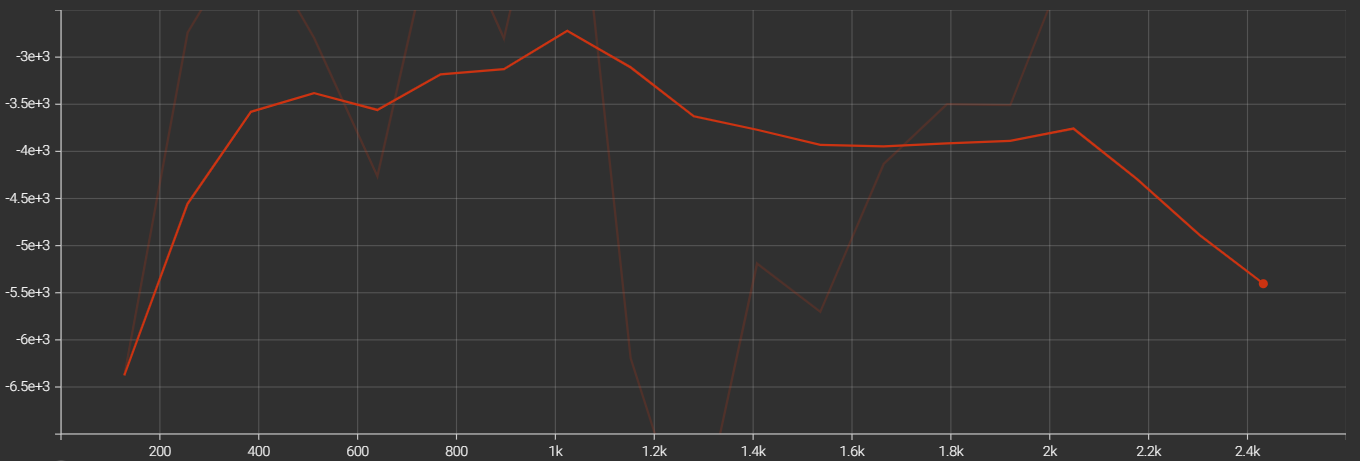
**Table 6.** Description of the experiments performed

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Run No.** | **Algorithm** | **Difficulty** | **Setup No.** | **Iterations** | **Mean Episode Reward** |
| 1 | A2C | Simple | 1 | 10,000 | -245 |
| 2 | Maskable PPO | Simple | 1 | 2400 | -14535 |
| 3 | Quantile Regression DQN | Simple | 2 | 10,000 | -819 |
| 4 | Deep Q Network | Simple | 2 | 10,000 | -825 |
| 5 | Deep Q Network | Medium | 3 | 10,000 | -819 |
| 6 | Proposed Deep Q Network | Simple | 2 | 208 | -134.3 |
| 7 | Proposed Deep Q Network | Medium | 3 | 323 | -60 |
| 8 | Proposed Deep Q Network | Complex | 4 | 80 | -17 |

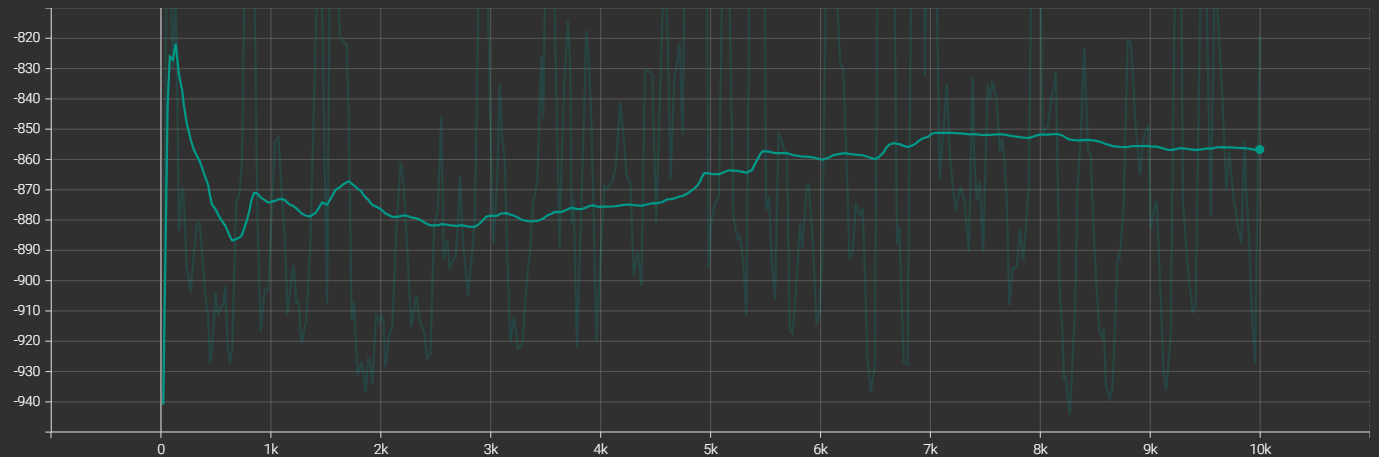
We plot the average episode rewards of each of the runs using Tensorboard to understand the effectiveness of the algorithm. The figures from Fig. 11 to Fig. 16 display the average episode reward plots for the different experiments that are performed according to Table 6.



**Fig. 11** Average episode reward for A2C in Run No.1



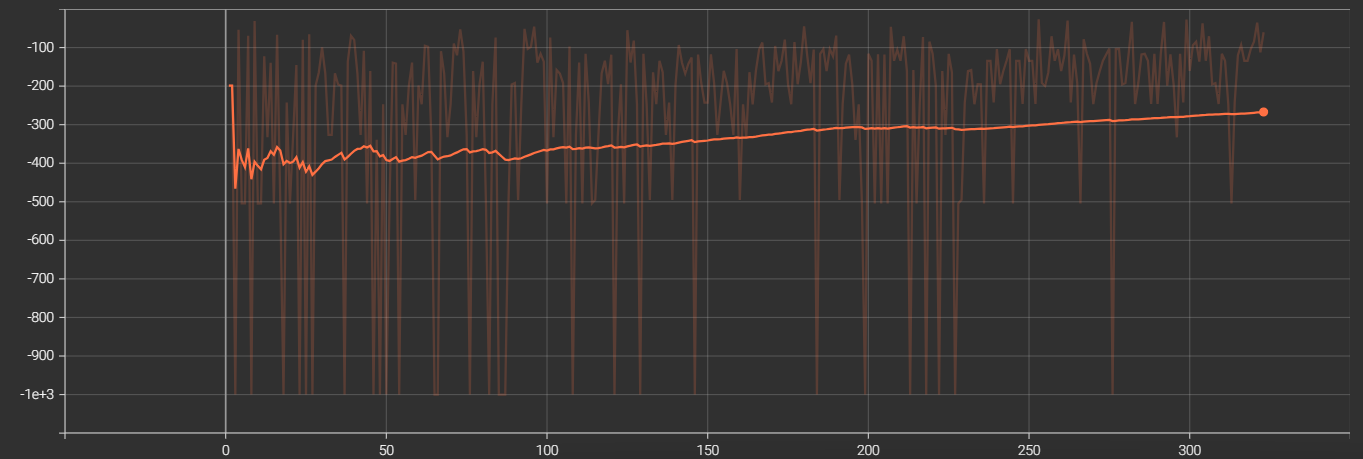
**Fig. 12** Average episode reward for Maskable PPO Run No.2



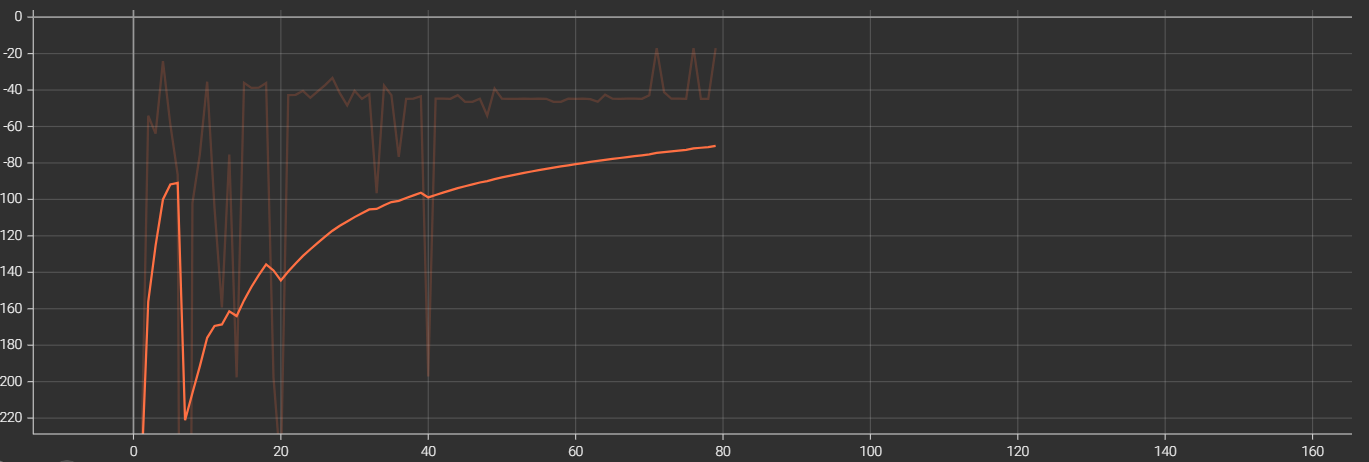
**Fig. 13** Average episode reward for DQN Run No.5



**Fig. 14** Smoothened graph showing the increase in average episode reward for Proposed DQN Run No.6



**Fig. 15** Smoothened graph showing the increase in average episode reward for Proposed DQN Run No.8



**Fig. 16** Smoothened graph showing the increase in average episode reward for Proposed DQN Run No.8

The graphs above denote the average episode rewards of each run through the training. The occasional dips in the average episode reward can be attributed to the exploration actions of the drones, where the drone gradually reduces the exploration rate through the training course. The dip occurs mostly as the drone collides due to the exploratory action leading to a very low reward (approx. -1000).

For Setup No. 3, the DQN model was implemented, with each camera input having an associated DQN model. The Q Values of the two networks were then averaged, and the best action was chosen from the resultant Q values. The custom-implemented DQN solved the environment by reaching the goal within 100 episodes, whereas the StableBaselines algorithms could not solve it despite being trained for 3000 episodes. Increasing the complexity of the environment, a slightly more challenging environment was chosen, and the proposed DQN model solved the environment, whereas the StableBaselines algorithms were not successful in replicating similar results in a higher number of episodes. The average reward mean of the algorithms is negative in the cases as the reward for collision is given as -1000, which is much greater than the other rewards that the drone can achieve.

Comparing the results of the custom-implemented DQN with the other methods; it can be inferred that the average reward is comparatively higher in each of the implemented setups, and the algorithm achieves better accuracy in fewer timesteps. This can be attributed to the additional features added to the algorithm, which makes the algorithm learn faster and more accurate.

Summarizing the performances of the algorithms can be displayed as given in Table 7, which shows the models that were able to solve the environment and those that were not able to. This gives a combined view of the performance of each of the models. Similarly, Table 8 shows the algorithms and the corresponding difficulty of the environments solved by them.

**Table 7.** Performance of the models in each setup

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S. No** | **Algorithm** | **Setup 1** | **Setup 2** | **Setup 3** | **Setup 4** |
| 1 | A2C | ✓ | - | - | - |
| 2 | Maskable PPO | 🗶 | - | - | - |
| 3 | Quantile Regression DQN | - | 🗶 | - | - |
| 4 | Deep Q Network | - | ✓ | 🗶 | - |
| **5** | **Proposed Deep Q Network** | **✓** | **✓** | **✓** | **✓** |

**Table 8.** Performance of model in each environment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S. No** | **Algorithm** | **Easy** | **Medium** | **Tough** |
| 1 | A2C | 🗶 | - | - |
| 2 | Maskable PPO | 🗶 | - | - |
| 3 | Quantile Regression DQN | 🗶 | - | - |
| 4 | Deep Q Network | ✓ | 🗶 | - |
| **5** | **Proposed Deep Q Network** | **✓** | **✓** | **✓** |

# Conclusion

Reinforcement Learning for drones is an appropriate technique, and the learning methodology is suited for drone applications which require drones to overcome unpredictable situations. Drones often operate in real-world environments, and applying reinforcement learning allows the drones to be better suited for applications in these real worlds as the algorithm allows drones to be trained and learn from close to real-world environments, giving better outcomes when deployed. By implementing the reinforcement learning training modifications, the networks learned faster while achieving better results in the same environments as the StableBaselines algorithms.

The implementation of the custom algorithms demonstrated a significant improvement over the other algorithms considering the similar factors. The additional features implemented, like the Unbiased memory and adaptive learning rate, demonstrated further improvement to the models. These changes made it possible for the drone to be able to travel the environment for longer and achieve its goals in the environment with a low chance of collision.

# Future work

Current work limits reinforcement learning to only a few environments, which could be increased and allow the drone to be trained in more diverse environments, including agricultural fields, factory environments, and other environments. The compute limitation limits the inputs that can be passed onto the algorithm during training due to the increased training time. More powerful computing could allow for more complex inputs to the algorithm, possibly leading to better performance. Neural networks could also be experimented with to improve performance or decrease the training time while maintaining the performance.

The changes made in the traditional DQN by implementing Adaptive learning rates and discrete decision systems could be further explored and integrated with other algorithms. Changes in the environment definition, reward functions, and other parameters of the environments, which mostly remain constant, can be explored.

##### References

[1] Xue Z, Gonsalves T. Vision Based Drone Obstacle Avoidance by Deep Reinforcement Learning. AI. 2021; 2(3):366-380. <https://doi.org/10.3390/ai2030023>

[2] A. Devo, J. Mao, G. Costante and G. Loianno, "Autonomous Single-Image Drone Exploration With Deep Reinforcement Learning and Mixed Reality," in IEEE Robotics and Automation Letters, vol. 7, no. 2, pp. 5031-5038, April 2022, doi: 10.1109/LRA.2022.3154019.

[3] Kalidas AP, Joshua CJ, Md AQ, Basheer S, Mohan S, Sakri S. Deep Reinforcement Learning for Vision-Based Navigation of UAVs in Avoiding Stationary and Mobile Obstacles. Drones. 2023; 7(4):245. <https://doi.org/10.3390/drones7040245>

[4] Tu G-T, Juang J-G. UAV Path Planning and Obstacle Avoidance Based on Reinforcement Learning in 3D Environments. Actuators. 2023; 12(2):57. <https://doi.org/10.3390/act12020057>

[5] Miera, P., Szolc, H., & Kryjak, T. (2023). LiDAR-based drone navigation with reinforcement learning. arXiv preprint arXiv:2307.14313.

[6] Chen, L., Yuan, X., Xiao, Y., Zhang, Y., & Zhu, J. (2020). Robust autonomous landing of UAV in non-cooperative environments based on dynamic time camera-LiDAR fusion. arXiv preprint arXiv:2011.13761.

[7] Puente-Castro, A., Rivero, D., Pazos, A., & Fernandez-Blanco, E. (2022). UAV swarm path planning with reinforcement learning for field prospecting. Applied Intelligence, 52(12), 14101-14118.

[8] Fu, C., Xu, X., Zhang, Y., Lyu, Y., Xia, Y., Zhou, Z., & Wu, W. (2022). Memory-enhanced deep reinforcement learning for UAV navigation in 3D environment. Neural Computing and Applications, 34(17), 14599-14607.

[9] Villota Miranda, D., Gil Martínez, M., & Rico-Azagra, J. (2021). A3C for drone autonomous driving using Airsim. In XLII Jornadas de Automática (pp. 203-209). Universidade da Coruña, Servizo de Publicacións.

[10] Ermağan, U., Yıldız, B., & Salman, F. S. (2022). A learning based algorithm for drone routing. Computers & Operations Research, 137, 105524.

[11] AlMahamid, Fadi, and Katarina Grolinger. "Autonomous unmanned aerial vehicle navigation using reinforcement learning: A systematic review." Engineering Applications of Artificial Intelligence 115 (2022): 105321.

[12] Serrano, Will. "Deep reinforcement learning with the random neural network." Engineering Applications of Artificial Intelligence 110 (2022): 104751.

[13] Hodge, Victoria J., Richard Hawkins, and Rob Alexander. "Deep reinforcement learning for drone navigation using sensor data." Neural Computing and Applications 33 (2021): 2015-2033.

[14] Pieczyński, Dominik, et al. "A fast, lightweight deep learning vision pipeline for autonomous UAV landing support with added robustness." Engineering Applications of Artificial Intelligence 131 (2024): 107864.

[15] Fu, Chenchen, et al. "Memory-enhanced deep reinforcement learning for UAV navigation in 3D environment." Neural Computing and Applications 34.17 (2022): 14599-14607.