Autonomous Drone Control using Reinforcement Learning

Kian Parnis
University of Malta
kian.parnis.20@um.edu.mt

Josef Bajada
University of Malta
josef.bajada@um.edu.mt

ABSTRACT

This project aims to develop a system for autonomous drone control that focuses on the problem of drone obstacle avoidance. This is crucial for safe drone deployment in industries like search and rescue, package delivery, and infrastructure inspections. The focus is on developing a reinforcement learning-based solution for unmanned aerial vehicles (UAVs) to navigate through cluttered environments with static or moving obstacles. To achieve this, AirSim simulated drone physics, and the UnReal engine created the virtual environment. A depth sensor captured environmental data, used as input for the reinforcement learning algorithm. The algorithm learned from observations, with depth imagery being the most significant. A Convolutional Neural Network (CNN) processed the depth imagery to extract relevant features. Additional inputs included velocity, distance from the goal, and action history. An Artificial Neural Network (ANN) processed the actions before combining them with the imagery. Four RL algorithms were trained in a static environment, and the best two were trained in a dynamic environment. These algorithims were: Deep Q-Network (DQN), Double Deep Q-Network (DDQN), Proximal Policy Optimization (PPO), and Trust Region Policy Optimisation (TRPO). The best performance came from the DDQN algorithm, reaching the target goals 93% of the time in environments with static obstacles and achieving an average of 84.5% in environments with dynamic obstacles.

1 INTRODUCTION

1.1 Problem Definition

The specific problem this thesis is addressing is developing a solution for unmanned aerial vehicles (UAVs) to navigate unmapped, dynamic, and static environments. The UAV must rely on onboard sensors to gather real-time information due to the absence of prior knowledge. The solution should effectively handle uncertainties, variations, and unpredictable obstacles, including both dynamic elements (e.g., moving objects) and static elements (e.g., fixed structures). Considering the presence of pre-existing static objects, such as buildings, enhances the UAV's situational awareness and allows for informed decisions to ensure safe and efficient flight. This becomes particularly crucial in situations such as natural disaster scenarios, where the lack of prior knowledge is prominent, as events like avalanches or earthquakes can introduce new static elements or modify existing ones.

1.2 Motivation

The need for autonomous drone control is motivated due to the sheer number of different applications that can utilize such technology. In [1] the use of Reinforcement Learning with drone control is highlighted in different problems such as path planning, navigation, and control. The following is a breakdown of some of the different applications highlighted by [1] which are Altitude control, Obstacle avoidance, Drone delivery, and Unmanned Rescue. Respective research in these fields can be found in [2, 3, 4, 5]. This project focuses on one of the most noticeable tasks UAVs need to overcome which is Obstacle Avoidance. The motivation behind focusing on Obstacle Avoidance is due to its overall importance in its involvement in various critical tasks. When carrying out a search and rescue operation for example. The autonomous drones' knowledge of navigating an adaptive environment is paramount to the overall success of the operation.

1.3 Challenges

This solution depends on the agent being able to navigate through static or dynamic environments that the agents had no prior knowledge of, depending on the limited data such as its sensor data, as well as its GPS signals to reach its targeted goal. The traditional path-planning algorithms, such as A* and RRT*, face several challenges in the context of the problem being addressed [6, 7]. A* relies on graph-based approaches and requires the drone to navigate the environment to gain knowledge about its surroundings. This can be problematic as it may involve flying large distances and backtracking, which is inefficient and risky in hazardous environments. Additionally, A* would need to recalculate its trajectory from scratch if the environment shifts, such as when obstacles move or wind interference occurs. On the other hand, RRT* is unable to handle dynamic environments and needs to reconstruct its mapped tree if the agent encounters wind interference.

To overcome these challenges, Reinforcement Learning (RL) can be employed. RL enables the agent to learn from its surroundings and make informed decisions based on acquired knowledge. By interacting with the environment, RL algorithms like Q-Learning [8] and Deep Q Network (DQN) [9] can learn which behaviors yield favorable outcomes. In the case of path planning for a drone in a dynamic environment, an RL agent can adjust to changes such as moving obstacles or wind interference by gathering data from its sensors and modifying its policies accordingly. Unlike A* or RRT* algorithms, this approach eliminates the need for backtracking or recalculating trajectories from scratch. Instead, the RL agent can continuously update its policy to adapt to the changing environment.

1.4 AIMS AND OBJECTIVES

The aim of this project is to develop algorithms for autonomous drone navigation that are effective in obstacle avoidance in unmapped environments. More specifically the goal is to train agents using different state-of-the-art Reinforcement Learning Techniques which are capable of navigating through environments that contain both static and dynamic obstacles. This will be accomplished by fulfilling the following objectives:

1. **Objective (O1):** Integrate a drone environment simulator within a Reinforcement Learning framework such that the drone can be programmatically controlled.
2. **Objective (O2):** Identify what sensor information to use for the observations and evaluate which algorithms and configurations perform obstacle detection accurately.
3. **Objective (O3):** Train four RL algorithms to navigate environments with static obstacles, with two algorithms using discrete actions and two algorithms using continuous actions, and evaluate them by the number of successful runs without collision.
4. **Objective (O4):** Train the best two RL algorithms on environments with dynamic obstacles, using adapted observations and rewards to capture obstacle movement information and evaluate them by the number of successful runs without collision.

2 BACKGROUND & LITERATURE REVIEW

2.1 Kinematics and Dynamics

Figure 1 shows the different variables of the kinematic and dynamic state of the drone. These states are the drone's position and orientation in space, represented by Cartesian coordinates (x, y, z) for position and Euler angles (ϕ, θ, ψ) for orientation. The dynamic state of the drone includes variables such as $v(x), v(y), v(z)$ for linear velocity along the x, y, and z axes, respectively. The drone's angular velocity is represented as the Roll(ϕ) rate, Pitch(θ) rate, and Yaw(ψ) rate. These states provide information about the drone's position, orientation, velocity, and angular velocity.

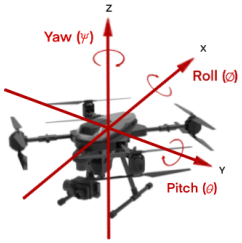


Figure 1: Kinematic and Dynamic State Variables of a drone

2.2 Reinforcement Learning

Reinforcement Learning (RL) is the process of learning what to do, by interacting with the environment. This involves the recognition of similarities between the characteristics of different situations and mapping these situations to actions, with the overall goal being to maximise some utility that is of interest. RL is typically broken down into an interaction loop (see Figure 2), this involves our agent taking some action in an environment based on some policy that transitions its current state to a new state while collecting some reward as well as an observation.

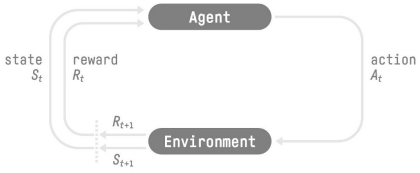


Figure 2: RL Interaction Loop [10]

The policy typically denoted as π, is a function that maps a state s_t to an action A_t (i.e., how the agent acts in each different environmental state by taking a decision to move into another state s_{t+1}, where t is the current time step). Rewards, are represented as numerical values given back to the agent as a result of the action taken in a state as R_{t+1}. Rewards R_t are used to improve the agent's policy in deciding what action to take in a particular state to reach its target goal, the optimal policy π^* is achieved by maximizing an expected cumulative reward (The sum of all rewards achieved after a particular time step) [11]. The environment is a scenario that is typically modeled around aiding the agent to train efficiently to reach its desired objectives/goal.

Deep Reinforcement Learning (Deep RL) is a sub-field of machine learning that combines Reinforcement Learning (RL) and Deep Learning. Deep RL is often used when the state-space is not feasible to engineer due to space constraints. For example, the game of chess has 10^{47} states while continuous actions are considered to have an infinite state space. Deep Learning aids by utilizing an approximation function which is parameterized by a vector of weights and is typically characterized as Artificial Neural Networks (which includes CNN's and other variants of Deep Neural Networks).

2.3 Convolutional Neural Networks

Drone imagery can be used as one of the observations that can be fed to a Reinforcement Learning algorithm to receive information about its surrounding environment. However, If a regular feed-forward neural network were to be used for this imagery, its large size would result in a huge number of neurons being needed. For example, an image with size 300x300 px and 3 color channels would result in the network having 270,000 weights (300x300x3), which is not only computationally expensive but leaves the network prone to over-fitting (learning to memorize its data set) due to a large number of parameters.

Convolutional Neural Networks (CNN) are based on the idea of Artificial Neural Networks but aim to use the idea of mathematical convolution to perform well with image, speech, or audio signal inputs. CNNs gained popularity based on the ImageNet architecture during 2012 [12]. Convolutions layers utilize Kernels (Also referred to as Filters and Feature Detectors) which are small matrices known as Edge Detectors that stride across the image starting from the top and progressively going downward per stride. For each stride, a dot product between the Kernel and the current position on the image is taken and saved into a Feature Map. This Feature Map would then reduce the image (depending on the kernel size) with specific patterns of information in an image being highlighted.

Pooling Layers are used to down-sample the feature map along its spatial dimensionality [13] to further reduce parameter size while maintaining important features which helps with the previously described over-fitting problem.

2.4 Observation Data

LiDAR (Light Detection and Ranging) is a sensor that has been studied for UAV obstacle avoidance. One example, [14], presented a navigation solution for UAVs in forests. It used a localization algorithm that combined LiDAR-based odometry and GNSS information. Another example, [15], described a system for determining safe landing areas using a 2D LiDAR-based ground system. It used a servo motor to generate a 3D point cloud from 2D laser distance data. However, LiDAR is computationally expensive and costly to implement due to the complexity of the software and processing resources required.

Another common sensor that can be used is the traditional camera. Cameras can provide visual data that can be used for tasks such as tracking, and classification. Unfortunately, cameras have well-known limitations in terms of their ability to perceive depth and achieve low performance in dimly lit environments making them unsuitable for obstacle avoidance. Yet another sensor that can be used in autonomous UAVs is radar. In [16] authors use millimeter wave (MMW) radar for vehicle obstacle avoidance. With their system, the distance between the object and the vehicle is computed for object detection and object tracking by observing the echoes created by radar signals. Also, the performance is assessed at various distances and weather conditions. Despite its appeal, similar to LiDAR radar-based systems are either too expensive or too heavy to be a payload for smaller robots like battery-powered UAVs [17].

Depth imagery has become a valuable sensor for autonomous UAVs. For instance, in [18], depth imagery was used with RL to teach drones independent action in suburban neighborhoods. Using Q-Network and Joint Neural Network (JNN), the drone learned obstacle detection and task completion. The environment had fixed and moving obstacles and a geo-fence. Depth images from the front camera and performance metrics boosted efficiency. Results showed high success rates in obstacle avoidance and reaching the goal. Another study [4] trained drones to deliver packages in neighborhoods with obstacles using stereo-vision front camera and a geo-fence. Depth information improved obstacle navigation during package delivery.

2.5 Reinforcement Learning Algorithms

In [3], a study compared Reinforcement Learning techniques for drones with obstacle avoidance capabilities. They evaluated discrete actions using DQN [9], Double DQN [19], Dueling DQN [20], and Double Dueling DQN (D3QN) algorithms, along with RGB and Depth imagery, in three environments: a woodland with random trees, a block world with 3D objects, and a curved trajectory block world with yaw control. The result of this first experiment showed that DQN's varients gave better performance in each environment. The reason why this is the case is that each of these algorithms features improvements on the vanilla DQN.

In the second experiment, continuous actions were employed to address the unnatural movement caused by discrete actions. Policy gradient algorithms (Trust-Region Policy Optimization [21], Proximal Policy Optimization [21], and Actor-Critic using Kronecker-Factored Trust Region [21]) were used with a U-Net-Segmentation model for vision-based segmentation. This approach eliminated the need for manual labeling of data. The results showed that continuous actions led to smoother flights in the third environment, resulting in better completion rates. Additionally, agents trained with PPO outperformed a fully tuned PID controller across various metrics.

2.6 Dynamic environments

Existing research primarily focuses on static environments in reinforcement learning (RL). However, RL algorithms encounter challenges in dynamic environments due to changes in the environment over time, such as non-stationarity and partial observability. Further investigation is necessary to assess the effectiveness of RL algorithms in dynamic environments.

In [22], a Deep Reinforcement Learning approach was proposed for path planning of Unmanned Aerial Vehicles (UAVs) in dynamic environments with potential threats. The approach utilized the Dueling Double Deep Q-networks (D3QN) algorithm to approximate Q-values for candidate actions based on global situation information. The proposed method's performance

was demonstrated in both static and dynamic scenarios using the STAGE Scenario software. The D3QN network effectively handled dynamic enemy threats by employing a set of situation map stacks as observations, instead of using a single RGB map. This stacking approach of image frames along the channel was also employed in [9] while using a Deep Q-Network (DQN). By concatenating the current and previous three frames of the Atari game screen, the DQN created a single input image for the deep neural network. This technique enabled the agent to observe object movements and interactions over time, providing important contextual information for decision-making and achieving high performance on Atari games.

An alternative method, presented in [23], involved using Recurrent Neural Networks within Deep Q-Networks to enhance perception and action in partially observed environments. The Deep Recurrent Q-Network (DRQN) replaced the first fully-connected layer with a recurrent LSTM, allowing the integration of information through time. DRQN performed well on Atari games with flickering screens, adapting better during evaluation when the quality of observations changed compared to DQN. Recurrency offered a viable alternative to frame stacking in the input layer of DQN, but no systematic advantages were observed between the two approaches.

3 METHODOLOGY

3.1 O1: Constructing the RL framework

One of the major challenges in RL-based drone navigation is the risk of collisions with obstacles, which can lead to significant damage to the drone and endanger people and property in the vicinity. Although training an RL-based drone obstacle avoidance system in the real world is feasible, it can be a time-consuming, expensive, and a hazardous process. However, simulating the environment and drone behavior can offer a safer, more cost-effective, and more efficient alternative. By doing so, the agent can learn the specified task of obstacle avoidance in a controlled environment before being deployed in the real world. To achieve this goal, we are utilizing AirSim and UnReal Engine to simulate our environment. This section provides the details of how these environments were constructed.

3.2 Creating the static and dynamic environments

For the static environment, a closed corridor was constructed with a length of 61.3 m, width of 21.8 m, and height of 15 m. Obstacles in the form of columns with a diameter of 1m were placed every 10.8 m for a total of four levels. The drone had to navigate through the corridor, avoiding the obstacles and gaps between them to pass the test. The environment was designed to be randomized at each episode to achieve generalization. The vertical and horizontal columns could be shifted uniformly between -3m and +3m programmatically from their original location dictated by a parent column. Figure 3 shows a sample of these generated environments.

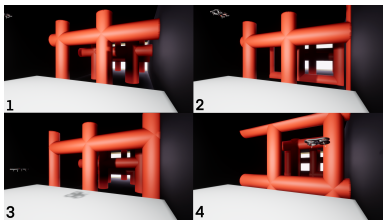


Figure 3: Four randomly generated static environments

For the dynamic environment, the same corridor dimensions were used, but instead of columns, cubes and squished cylinders of various sizes were used as obstacles. The obstacles were programmed to move from one point to another at a constant speed, taking 20 seconds in real time to reach their destination. Two approaches were taken to test generalization: one with randomized obstacle order and one with a familiar training environment but shuffled obstacle order. Figure 4 shows these two environments.

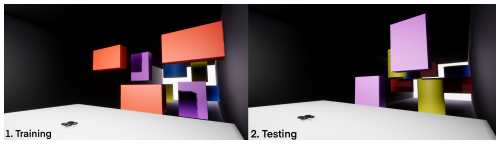


Figure 4: Original Training and Shuffled Testing Environments

4 O2: Sensor Information

As discussed in Section 2.4, Depth imagery uses active sensors such as stereo cameras or time-of-flight cameras to create 3D point clouds that represent the depth information of the environment. In conjunction with Convolutional Neural Networks (CNNs), depth information can be coupled to improve the accuracy of object detection and recognition tasks. Depth information provides accurate data even in low light conditions compared to the traditional camera, and its cost in terms of computation and the overall price is relatively low compared to both LiDAR and Radar, making it the preferred sensor for autonomous UAVs that require accurate depth perception for the task of obstacle avoidance. Moreover, as detailed in Section 2.3, a CNN architecture can be used to process this imagery to lower the computational expense as well as avoid over-fitting that occurs when passing imagery straight through an ANN. In order to solely focus on the classifier's capability to predict obstacles the problem was broken down and changed to a classification problem which was trained via Supervised Learning. Breaking down the whole problem into individual parts helps better verify if, for example, the retrieved sensor data is sufficient for Obstacle Avoidance.

4.1 Data Collection

To train a collision prediction model, sufficient training data is essential. We obtained this data by running the UAV multiple times through our environments and collecting imagery using the AirSim API. Each episode ended either with a collision or when the goal was reached, and the data was labeled as Collision (1) or Non-Collision (0) based on a collision flag returned by the API. To ensure the drone takes evasive action before getting too close to obstacles, we maintained a stack of images taken at each time step. The last few images before a collision were labeled as collisions, while the rest were labeled as non-collision. We conducted 500 episodes in static and dynamic environments, resulting in datasets of 10,392 and 11,949 images, respectively. These datasets were split into Training, Validation, and Test sets (80%, 10%, and 10%, respectively) to evaluate the model's performance and generalization ability. We trained a Convolutional Neural Network (CNN) using the TorchVision library, aiming for a lower parameter CNN to accommodate drones' limited processing power, faster processing time, and reduced memory usage.

5 O3, O4: Navigation in Static and Dynamic Envrionments

For our final two objectives, we train four RL algorithms (two using discrete actions, two using continuous actions) in a static environment and the best-performing two in dynamic environments. Time constraints necessitated this approach, as each algorithm required extensive tuning and substantial training to attain optimal performance. During the training process, we made adaptations to the action space, observation space, and rewards based on whether the agent operated in a static or dynamic environment and whether the action space was discrete or continuous. Thus in this section, the action spaces, observations spaces, and rewards shall be outlined.

5.1 Actions

Discrete actions involve a limited set of choices, while continuous actions offer an infinite range of possibilities. In the context of drone obstacle avoidance, discrete actions include specific movements like "move left," "move right," "move forward," and "move backward." On the other hand, continuous actions pertain to controlling the drone's speed and direction. For encoding discrete actions, the agent is given six options: move right, move down, move back, move left, move up, and hover. These actions correspond to movements along the x-axis, y-axis, and z-axis. When executing a discrete action, the drone's speed is set to 1 m/s along the chosen axis and 0 m/s for the remaining axes during hovering. Regarding continuous actions, the agent can select velocities within a floating-point range of -1.0 m/s to 1.0 m/s for each axis. This allows the drone to move along the x-axis, y-axis, and z-axis with varying degrees of velocity or even stay stationary along a specific axis by choosing velocities within this range.

5.2 Observations

Regarding the observation space, in the case of discrete actions, an additional observation is included to retain a record of prior actions taken. During training, it was observed that the agent would occasionally become trapped in an action loop directly in front of an obstacle, resulting in poor

performance. This looping behavior was caused by the agent's insufficient prior event history. To address this issue, a queue containing the 10 most recent actions was implemented. This enabled the agent to develop a more comprehensive understanding of previous events, thereby preventing it from becoming trapped in the aforementioned loop. On the other hand, when training using continuous actions, since the agent wasn't limited to movement along the x, y, and z axis this problem was not encountered.

Another difference in observation spaces occurred when dealing with static and dynamic environments. This is highlighted in Section 2.6, whereas for static environments one frame observation per step is sufficient, on the other hand, a stack of observation frames along the channel frames is required for the agent to be able to observe how obstacles are moving. These frames are collected between each step, starting at the beginning before an action is taken, in the middle of the action being taken, and a final frame when the action concludes. The agent received supplementary observations that are used across various action and obstacle types, including the agent's velocity, its relative distance to the goal, and its prior relative distance to the goal. Relative Distance is calculated using Euclidean distance for each axis as shown before:

$$d(x,y,z) = \sqrt{(x_{agent} - x_{goal})^2}, \sqrt{(y_{agent} - y_{goal})^2}, \sqrt{(z_{agent} - z_{goal})^2}, \quad (1)$$

When these observations are fed into their respective Reinforcement Learning Algorithms, as mentioned in Section 2.3 it is important that the observation imagery is passed through a Convolutional Neural Network. The rest of the observations are passed through a linear layer of size 16, followed by a ReLu activation function. Each observation is then concatenated together before being fed to the feature extractor class.

5.3 Rewards

While the structure of rewards in the environments remained mostly consistent, the rewards varied slightly when changing actions spaces from discrete to continuous. The primary objective of the agent is to maximise its cumulative rewards over time. To achieve this objective, negative rewards were employed as a penalty to discourage undesirable behavior, while positive rewards were used to reinforce desired behavior. The largest rewards and penalties were given either when the agent collides (-100 penalty) or when the agent reaches the desired objective (100 reward). An additional -100 penalty is given if the agent fails to reach the objective after a substantial amount of time. Once any of these conditions are met the episode ends, resetting the whole environment.

The agent receives a interim reward of 20 every time it 'levels-up'. Level-ups are achieved every time an agent successfully crosses an obstacle. This reward encourages the agent to avoid obstacles and move closer to the goal. Furthermore, euclidean distance is also used to encourage the agent to move towards the goal on a more frequent basis, giving minor penalties or minor rewards at each step depending on whether the agent is moving towards or away as follows: This reward is calculated by the following:

$$reward = (d_{previous} - d_{current}) - d_{pos} \quad (2)$$

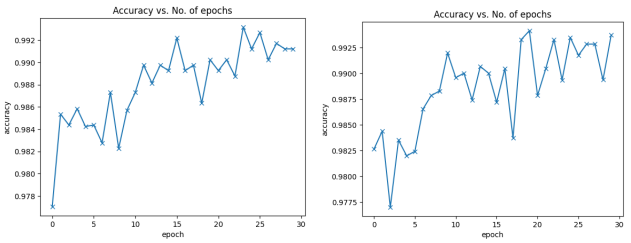
where $d_{previous}$ is the euclidean distance of the agent's previous position from the target goal, $d_{current}$ is the distance of the agent's current position from the target goal, d_{pos} is the distance between the agent's previous position and current position. The only difference in rewards for each action space is a penalization if the agent hovers for too long to discourage staying idle. For discrete actions, hovering is designated as a specific action, which incurs a penalty of -0.5 when selected. On the other hand, for continuous actions, the agent hovers if it selects a specific range of values for each axis, this range is specified between a speed of -0.2 m/s and 0.2 m/s for all three axes incurring a penalty of -1.5.

6 EVALUATION AND RESULTS

6.1 Obstacle Detection Evaluation

In our initial experiment, we evaluated the classification accuracy of our trained classifiers for collision detection. We collected training data by flying a UAV through our environments multiple times from different starting positions. This data was used to train CNN models of varying complexity to classify collisions. Our goal was to enable navigation through both static and dynamic environments, so we created two datasets. The first dataset consisted of 150 x 150 x 1 images for classifying predictions in static environments accurately. The second dataset included stacked 150 x 150 x 3 images to address the non-stationarity challenge in dynamic environments.

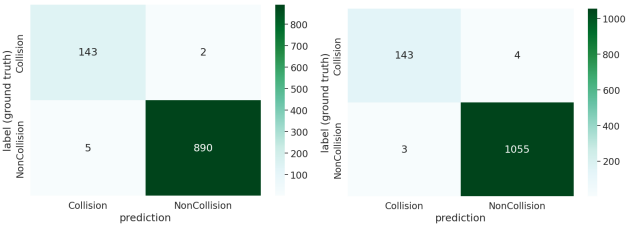
All architectures were trained effectively and achieved high accuracy. Even the most complex architecture, with 86 million parameters and a size of 401 MB, showed only a negligible drop in performance compared to the least complex architecture with 4 million parameters and a size of 15 MB. This held true for both single and stacked imagery. Therefore, we integrated the architecture with the least computational expense while maintaining high classification accuracy. In Figure 5a and Figure 5b, we showcase the accuracy achieved on this architecture over the validation set for thirty epochs. The CNN trained on single imagery achieved a maximum accuracy of 99.17% on our validation set, while the same architecture trained on stacked imagery achieved an accuracy of 99.37%.



(a) Static Accuracy (b) Dynamic Accuracy

Figure 5: Training Accuracy of the chosen CNN Architecture on Dynamic and Static Environments

After training, the trained models were run on an unseen test set, and with these predictions, the confusion matrices were formed. From these matrices, we can then infer our model's Accuracy, Precision, Recall, and F1-Score. These formed matrices for the architecture used are illustrated in Figures 6a and 6b.



(a) Static Confusion Matrix (b) Dynamic Confusion Matrix

Figure 6: Confusion matrix of the chosen CNN Architecture on Dynamic and Static Environments

The results in Figures 6a and 6b demonstrate the effectiveness of our CNN architecture in obstacle detection for both static and dynamic environments. In static environments, we achieved an adequate 99.3% accuracy, 96.6% precision, 98.6% recall, and 97.6% F1-score. Meanwhile, in dynamic environments, we achieved an accuracy of 99.4%, with 97.9% precision, 97.2% recall, and 97.5% F1-score. Recall is the most important metric for our obstacle detection system. This is because Recall is a measure of how many actual obstacles the drone properly identified and avoided. With a recall of 98.6% in static environments and 97.2% in dynamic environments, we have demonstrated the high performance of our depth sensor-equipped drone in detecting obstacles. These results indicate that our CNN architecture, combined with a depth sensor, is well-suited for obstacle detection in various environments.

6.2 Navigation and Obstacle Avoidance Evaluation

In our second experiment, we trained four algorithms in a Static Environment: two continuous and two discrete. Each algorithm ran for 50k time steps, with early stopping at 40k time steps. In terms of real-time performance, the DDQN algorithm converged the fastest in 10 hours. The DQN algorithm took 11 hours and 30 minutes to converge, followed by TRPO at 11 hours and 40 minutes. The PPO algorithm required 12 hours to reach convergence. However, when considering the number of time steps, the PPO algorithm demonstrated the quickest convergence, only requiring 42k time steps. TRPO followed with 45,500 time steps, while DDQN took 46,500 time steps. The DQN algorithm reached the maximum time steps

of 50k without triggering early stopping. This difference between real-time convergence and time-step convergence is due to the mean episode length. Continuous algorithms took longer due to a speed of approximately 0.5m/s for safety, while discrete algorithms had a fixed forward speed of 1m/s. Figure 7 shows the disparity in episode length between the four algorithms.

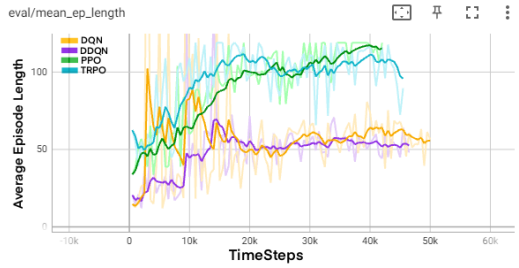


Figure 7: Mean Episode Length between the two continuous and two discrete agents

Regarding rewards, a direct comparison between Discrete and Continuous actions is challenging due to differences in their reward structures. Nevertheless, we can still compare the algorithms within each set. Among the discrete algorithms (Figure 8), DDQN exhibited a slight improvement over DQN, with a higher average reward over time. However, it is worth noting that DQN achieved a slightly higher maximum reward of 186.2 compared to DDQN's 183.2. For the continuous algorithms, (Figure 9), both showed a steady increase in rewards during training, but at around 20k time steps, their policies experienced a temporary drop in performance. This could be due to factors such as policy updates causing divergence from the optimal solution or a trade-off between exploration and exploitation, where the agent starts effectively exploring the learned policy over time. In the first half, TRPO outperformed PPO in terms of average reward, but PPO recovered with a steep increase in average reward and achieved a higher maximum reward of 152.4, while TRPO showed a gentler increase in average reward with a slightly lower average reward of 151.1.

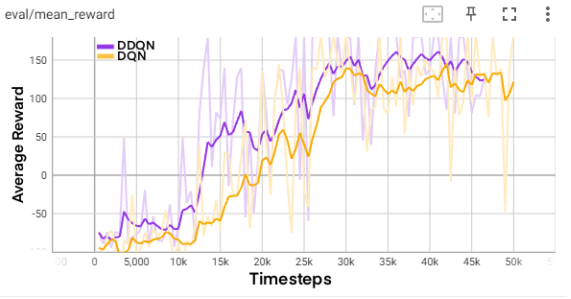


Figure 8: Mean Reward vs Time steps for DQN & DDQN in a Static Environment

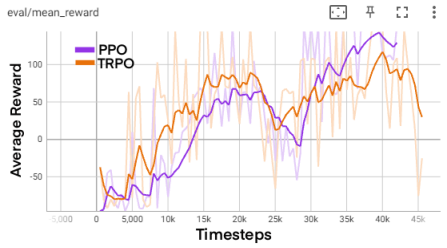


Figure 9: Mean Reward vs Time steps for PPO & TRPO in a Static Environment

After these models were fully trained, we then proceeded to evaluate their performance by running them through the randomized environment a hundred times. Then, the averages were computed by how many times the agents could surpass the presented obstacles and reach the goal while maintaining generalizability. The environment used in this study was fully randomized, resulting in an unseen environment for each run. The obstacles were divided into four levels, corresponding to the four levels of obstacles encountered by the agent. If the agent successfully passed the final obstacle, it was considered to have reached the goal.

For the discrete algorithms, the results revealed that the DDQN algorithm achieved the highest completion score of 93%, indicating the best performance in obstacle avoidance within a generalized static environment. In comparison, the DQN algorithm achieved an average score of 86%, which was 7% lower than the DDQN algorithm. For a detailed breakdown of the results, please refer to Table 1.

Table 1: Testing discrete algorithms' avoidance rate on static environments

Average Obstacle Avoidance Rate (Static Environment)		
Obstacle Sequence	DDQN	DQN
Level: 1	97%	95%
Level: 2	94%	93%
Level: 3	93%	91%
Goal Reached	93%	86%

In contrast, the continuous algorithms showed different results. The PPO algorithm achieved a completion score of 79%, while the TRPO algorithm scored an average completion of 71%, 8% lower than PPO. These results are summarized in Table 2. Notably, there is a 14% drop in completion score between the best-performing discrete algorithm and the best-performing continuous algorithms. The agents operating in continuous space faced difficulties in navigating tighter areas. They tended to oscillate between the y-axis and z-axis instead of moving forward steadily. This behavior led to occasional contact with obstacles, preventing them from reaching the goal. The average passing rates support this observation. The PPO agent successfully passed the first level 97% of the time but only reached the goal in the final level 79% of the time. In contrast, the discrete algorithms, with their fixed action space, performed better in maneuvering through narrower gaps. For example, the DDQN agent only experienced a 4% decrease in success rate between passing the first level and reaching the goal in the final level.

Table 2: Testing continuous algorithms' avoidance rate on static environments

Average Obstacle Avoidance Rate (Static Environment)		
Obstacle Sequence	TRPO	PPO
Level: 1	97%	91%
Level: 2	92%	86%
Level: 3	89%	77%
Goal Reached	79%	71%

For our final experiment, we trained the two best-performing algorithms, which are the DQN and DDQN algorithms, and trained them in a Dynamic Environment. We yet again ran these algorithms for 50k time steps, with early stopping at 40k time steps. Both algorithms didn't trigger early stopping and trained for a full 50k timesteps both taking 11 hours to complete training. In terms of episode length, both algorithms took roughly the same amount of time to reach the goal. In terms of reward, DDQN again exhibited a slight improvement over DQN, with a higher average reward over time. However, it is again worth noting that DQN achieved a higher maximum reward of 188.5 compared to DDQN's 156.2. This is shown in Figure 10.

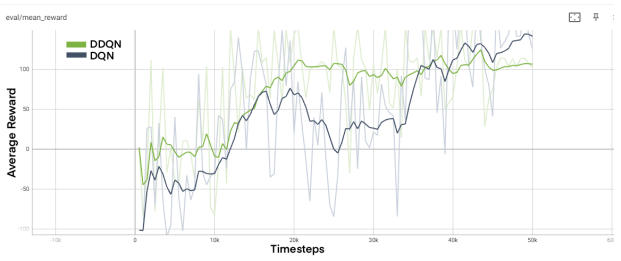


Figure 10: Mean Reward vs Time steps for DQN and DDQN in a dynamic environment

After the models were fully trained, we measured their average completion rate by running the agent's policies through the same environment used during training. Next, we tested the models on a shuffled version of the environment, where the order of obstacles was randomized. This was done in order to verify the agent's ability to generalize in an environment it hasn't seen before. Similar to the previous experiment, we performed 100 run-throughs for each environment in this experiment. The obstacles were again divided into four levels, with the successful passage of the final obstacle considered as reaching the goal. We computed the average number of times each obstacle was overcome, which is shown in Table 3.

Table 3: Testing discrete algorithms' avoidance rate on dynamic environments

Average Obstacle Avoidance Rate (Dynamic Environments)				
	DDQN		DQN	
Obstacle Sequence	Training Env	Testing Env	Training Env	Testing Env
Level: 1	99%	94%	95%	93%
Level: 2	93%	89%	91%	89%
Level: 3	87%	86%	88%	84%
Goal Reached	85%	84%	85%	81%
Average Goal Reached	84.5%		83%	

In Table 1 and Table 3, the best-performing algorithm in a static environment achieved a completion rate of 93%, while the best-performing algorithm in a dynamic environment achieved 84.5%, a drop of 8.5%. Comparing the two algorithms in the dynamic environment, DDQN achieved 84.5% completion rate, slightly better than DQN's 83%. In a shuffled environment with randomized obstacles, DQN's completion rate dropped by 4%, while DDQN's dropped by only 1%, demonstrating their ability to generalize well to new environments.

7 Conclusion

The aim of this thesis was to develop a reinforcement learning-based solution for unmanned aerial vehicles (UAVs) that enable drones to safely navigate through unmapped cluttered environments, including obstacles that are either static or moving. To reach this aim. the following objectives were set and achieved.

The first objective, **O1**, involved constructing a Reinforcement Learning Framework using Air-Sim, Unreal Engine, and Python (Gym and Stable-Baseline3) for drone control. Simulated environments with static and dynamic components, allowing evaluation of generalization capabilities. For our second objective, **O2** we determined depth sensor as the best option for obstacle detection. Converted the problem to a classification task, training Convolutional Neural Networks (CNNs) with labeled imagery from the depth sensor. Chose the smallest CNN architecture for computational efficiency and achieved high recall rates (98.2% and 97.2%) for static and dynamic environments. In our third objective, **O3**, we deployed four agents with different algorithms (Double Deep Q-Network, Deep Q-Network, Proximal Policy Optimization, Trust Region Policy Optimization) in an environment with static obstacles. Evaluated agents' performance by measuring the average number of obstacle clearances and goal achievements. DDQN performed the best (93% completion rate), followed by DQN (86%), PPO (79%), and TRPO (71%). Finally, for our fourth objective, **O4**, we utilized the top-performing algorithms (DQN and DDQN) in an environment with dynamic obstacles. Adapted observations to stacked image frames for capturing obstacle movement. Measured average completion rates in both the training and shuffled testing environments. DDQN achieved the highest performance (84.5% completion rate), with DQN closely following (83% completion rate).

7.1 Future Work

In this project, the main focus is on Air-Sim's higher level of control for drone navigation in complex environments. The agent has both discrete and continuous action spaces in the x, y, and z-axis. To further enhance the complexity, we can incorporate the drone's Yaw, Roll, Pitch, and Throttle, which require considering negative to positive ranges for Yaw, Roll, and Pitch (in radians) and a throttle range of 0.0 to 1.0. This introduces a new dimension of control for rotation, enabling more complex environments and curved trajectories. Accompanying this new level of control are rewards and observations related to the agent's orientation and body frame relative to the goal, as discussed in [24]. Our policy evaluation in this project is limited to the AirSim Simulation. However, future work can extend it to real-life scenarios replicated and tested within the simulation. For example, [25] proposed a high-level controller that tracks and pursues another UAV in a real-life pursuit-evasion scenario. We can leverage this approach to assess the effectiveness of our policies in a more realistic and challenging setting.

References

[1] A. T. Azar, A. Koubaa, N. Ali Mohamed, H. A. Ibrahim, Z. F. Ibrahim, M. Kazim, A. Ammar, B. Benjdira, A. M. Khamis, I. A. Hameed, and others, "Drone deep reinforcement learning: A review," *Electronics*, vol. 10, no. 9, p. 999, 2021.

[2] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement Learning for UAV Attitude Control," *ACM Trans. Cyber-Phys. Syst.*, vol. 3, 2 2019.

[3] S.-Y. Shin, Y.-W. Kang, and Y.-G. Kim, "Obstacle Avoidance Drone by Deep Reinforcement Learning and Its Racing with Human Pilot," *Applied Sciences*, vol. 9, no. 24, 2019.

[4] G. Muñoz, C. Barrado, E. Çetin, and E. Salami, "Deep in Reinforcement Learning for Drone Delivery," *Drones*, vol. 3, no. 3, 2019.

[5] W. Cao, X. Huang, and F. Shu, "Unmanned Rescue Vehicle Navigation with Fused DQN Algorithm," in *Proceedings of the 2019 International Conference on Robotics, Intelligent Control and Artificial Intelligence*, RICAI 2019, (New York, NY, USA), pp. 556–561, Association for Computing Machinery, 2019.

[6] C. Zammit and E.-J. Van Kampen, "Comparison between A* and RRT Algorithms for UAV Path Planning," Proceedings of the 2018 AIAA Guidance, Navigation, and Control Conference, 11 2018.

[7] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," *CoRR*, vol. abs/1105.1186, 2011.

[8] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and others, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[11] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[12] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Neural Information Processing Systems*, vol. 25, 11 2012.

[13] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[14] E. Aldao, L. M. González-de Santos, and H. González-Jorge, "LiDAR Based Detect and Avoid System for UAV Navigation in UAM Corridors," *Drones*, vol. 6, no. 8, p. 185, 2022.

[15] G. Ariante, S. Ponte, U. Papa, A. Greco, and G. Del Core, "Ground Control System for UAS Safe Landing Area Determination (SLAD) in Urban Air Mobility Operations," *Sensors*, vol. 22, no. 9, 2022.

[16] C. Blanc, R. Aufrère, L. Malaterre, J. Gallice, and J. Alizon, "Obstacle detection and tracking by millimeter wave RADAR," *IFAC Proceedings Volumes*, vol. 37, no. 8, pp. 322–327, 2004.

[17] M. P. Owen, S. M. Duffy, and M. W. M. Edwards, "Unmanned aircraft sense and avoid radar: Surrogate flight testing performance evaluation," in *2014 IEEE Radar Conference*, pp. 548–551, 2014.

[18] E. Çetin, C. Barrado, G. Muñoz, M. Macias, and E. Pastor, "Drone Navigation and Avoidance of Obstacles Through Deep Reinforcement Learning," in *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, pp. 1–7, 2019.

[19] H. v. Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 2094–2100, AAAI Press, 2016.

[20] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 1995–2003, JMLR.org, 2016.

[21] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, 2015.

[22] C. Yan, X. Xiaojia, and C. Wang, "Towards Real-Time Path Planning through Deep Reinforcement Learning for a UAV in Dynamic Environments," *Journal of Intelligent & Robotic Systems*, vol. 98, 3 2020.

[23] M. J. Hausknecht and P. Stone, "Deep Recurrent Q-Learning for Partially Observable MDPs," in *AAAI Fall Symposia*, 2015.

[24] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous Drone Racing with Deep Reinforcement Learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1205–1212, 2021.

[25] M. A. Akhloufi, S. Arola, and A. Bonnet, "Drones Chasing Drones: Reinforcement Learning and Deep Search Area Proposal," *Drones*, vol. 3, no. 3, 2019.