**A Thesis**
**For the Degree of Master of Science**

# Autonomous UAV Navigation in Unknown Terrain/Environment using Reinforcement Learning

**Mudassar Liaq**

**Department of Computer Engineering**

**GRADUATE SCHOOL**
**JEJU NATIONAL UNIVERSITY**

**June 2019**

# Autonomous UAV Navigation in Unknown Terrain/Environment using Reinforcement Learning

**Mudassar Liaq**

**(Supervised by Professor Yung-Cheol Byun)**

Submitted to the Department of Computer Engineering and the Faculity of Graduate School of Jeju National University in partial fulfillment of the requirements for the degree of Masters of Computer Engineering

2019.06

The thesis has been examined and approved.

Yung cheol Byun

Thesis Supervisor, Yung-Cheol Byun, Professor, Jeju National University

Wang Cheol Song

Wang-Cheol Song, Professor, Jeju National University

Sang Yong Byun

Sang-Yong Byun, Professor, Jeju National University

Department of Computer Engineering

GRADUATE SCHOOL
JEJU NATIONAL UNIVERSITY



ii

*Dedicated to "Salma Yasmin"*

*"Mom, I am privileged to be your Son".*

# ACKNOWLEDGMENTS

In the name of Allah, the most beneficent, the merciful. Without his grace, I am nothing, and to whom I am bound to return.

First and foremost, Professor Yung-Cheol Byun, for continuously guiding me and helping me in research here at JNU.

Some very humble seniors from Dr. Afaq to Dr. Israr from whom I was able to learn very different, contrast yet successful ways of life.

Some very close friends who have drifted away in winds of time (The Molvis, Ustads, Murshids, Silents, and Sattis)

To Ms. Wafa Shafqat. You are something else entirely.

Lastly and most importantly my nephew Aayad, my brother-in-law Waqar Ahmed, my sisters Shahwana Waqar and Ridda Liaq, my father Liaq M. Khan, and of course "Salma Yasmin," my mother to whom I owe everything.

***Mudassar Liaq***

June 2019

# Autonomous UAV Navigation in Unknown Terrain/Environment using Reinforcement Learning

Mudassar Liaq

*Supervisor: Prof. Yung-Cheol Byun*

**ABSTRACT**

Over the last few years, UAV applications have grown immensely from delivery services to military use. Major goal of UAV applications is to be able to operate and implement various tasks without any human aid. To best of our knowledge, in the existing works for autonomous navigation for UAV's, ideal environments (e.g., 2D) are considered instead of realistic or special hardware are used (e.g., nine range sensors) to navigate through an ideal environment. Therefore, in this thesis, we aim to overcome the limitations of the existing works by proposing a model for navigating a drone in an unknown environment without any human help or aid. The goal of this research is to navigate from location A to location B in unknown terrain without having any prior knowledge about the terrain using default drone sensors only. We present a model which is compatible with almost every off-the-shelf drone available in the market. Our methodology utilizes only standard drone sensors which are attached to almost every drone. These include a camera, GPS, IMU, magnetometer, and barometer. Our methodology uses 3D, POMDP, and continuous environment. We have experimented with three different types of simulation environments in this work; *Blocks, Landscape, Neighborhood.*

In our approach, we use a DNN for predicting UAV's next movement. First few layers are convolutional layers with propose of generating deep vector representation of camera image. This

deep vector representation is combined with other sensor data in fully connected layers. The network outputs the next movement for our UAV. Our neural network architecture is sort of CNN but the key difference is instead of classification, it generates probability distribution for the next possible movement of UAV.

We achieved completely autonomous (unaided) flight and navigation using Deep Q-learning, which is a subfield of RL. We implemented two different versions of the algorithm, i.e. policy-based DQN and value-based DQN. We were able to achieve 97.24% success rate for policy-based DQN and 96.74% success rate for value base DQN. We were able to demonstrate that our proposed approach was able to navigate the unknown environment successfully when it was trained for over 1000 iterations. The results showed that the rewards for the first 500 iterations were low. This was because the DQN was exploring different strategies and finding ones which work. Post 500 iterations the reward started to go up, and the performance started to improve. After 1000 iterations the DQN was successfully able to navigate drone in an unknown environment with ease.

Our main contributions are: 1) We are using realistic environment model including factors like rain and wind. 2) We are only using onboard computing resources to run our model instead of some external server. 3) We were able to achieve improved results in terms of success (97.24%), failure (1.09%), and stray rate (1.66%). Another factor that distinguishes this work from other works is its potential for mass adaptability. In this work we are only using standard sensors without any special hardware requirements. This make our work widely adaptable for any off-the-shelf drone in the market.

# List of Acronyms

| | |
|---|---|
| UAV | Unmanned Aerial Vehicles |
| UAS | Unmanned Aircraft System |
| NN | Neural Network |
| DNN | Deep Neural Network |
| CNN | Convolutional Neural Networks |
| RL | Reinforcement Learning |
| DQN | Deep Q Network |
| POMDP | Partially Observable Makarovian Decision Process |
| API | Application Programming Interfaces |
| ICCA | intelligent Cooperative Control Architecture |
| PPO | Proximal Policy Optimization |
| DDPG | Deep Deterministic Gradient Policy |
| ROS | Robot Operating System |
| VLOT | Vertical Take-off and Landing |
| RTT | Round Trip Time |
| MDP | Makarov Decision Process |
| YOLO | You Only Look Once |
| IMU | Inertial Measurement Unit |
| PID | Proportional Integral Derivative Unit |
| GPS | Global Positioning System |
| TRPO | Trust Region Policy |
| MAV | Micro-Air Vehicle |
| SLAM | Simultaneous Localization and Mapping |
| CGLA | Cooperative and Geometric Learning Algorithm |

# CONTENTS

# List of Tables

# Chapter 1

# Introduction

With the rapid advancements in the technology, the fields of robotics and mechatronics have drastically upgraded in terms of their role in the modern business and different industries. UAV also known as a drone is an aircraft that flies without any human pilot and are controlled remotely.

## 1.1. Drones and UAVs

These UAVs will potentially affect the way traditional businesses run and how different companies operate. Besides the challenges and risks the commercial UAVs industry is facing, a range of diverse products are being presented. There are organizations and companies focusing on to generate productive ways of UAVs usage in daily tasks.

A considerable number of investors are investing in the idea of new vehicles generation that are capable of multiple tasks, e.g., flying, swimming, crawling up a wall or even walk. According to a report [1], drone companies managed to take around $3,163 billion of investments since 2008.

The excellent growth rate of the UAV industry makes it nearly impossible to list down or explain all the applications that have been used regularly nowadays; also, its difficult to mention all the businesses concerned in developing drone platforms or drone parts suppliers e.g., sensors as cameras, thermal, video, and infrared, etc.

## 1.2. Applications of UAVs

The UAV applications have attained a high reputation as they benefit the aerial vehicles e.g., quadcopters with an ease that can help them execute different tasks such as avoiding obstacles without collisions, autonomous maneuvering, and VLOT. For a quadcopter to perform in an indoor environment, it requires to operate with agility and efficient feedback without losing control. Stabilization is needed for an outdoor environment for a drone to operate against external forces. In short, for both situations, a stable flight is immensely important for a drone to function effectively. This can help a quadcopter to survive in extreme external factors e.g., heavy rain or wind as it will be able to prevent crashing. In order to use these applications for navigation and stabilization, UAVs and drones need to be supplied with a series of sensors e.g., depth cameras, accelerometer, gyroscope, magnetometer, etc.

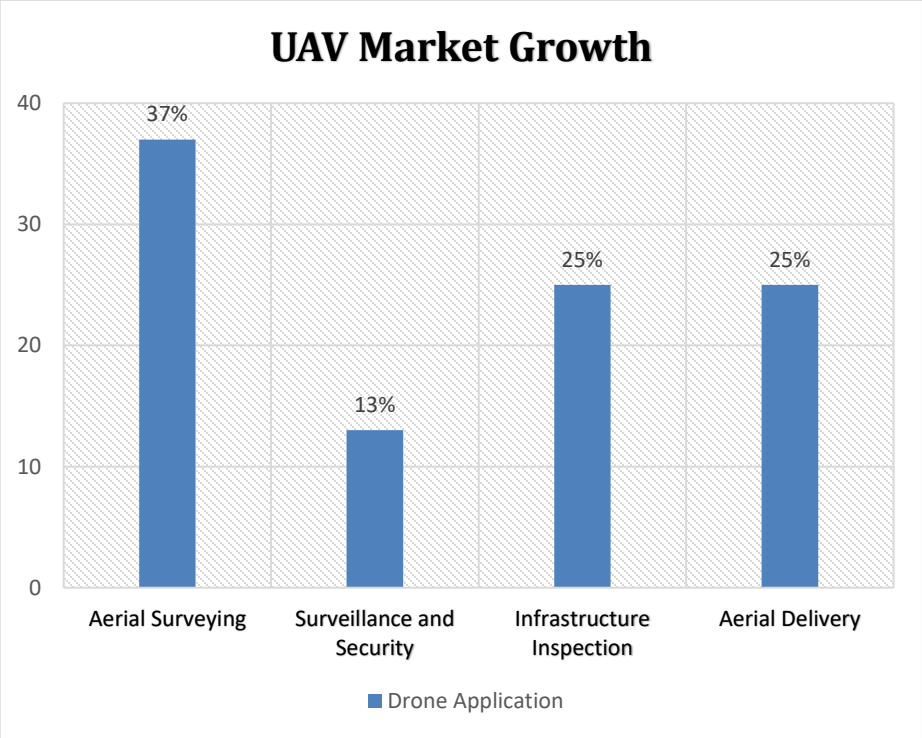


**Figure 1. UAV Applications (Market Growth)**

Apart from this, applications of drones can be observed in many fields such as accident reporting, infrastructure inspection, crop monitoring, etc. Figure 1 is from a survey which presents the growth rate of different UAV applications, it can be seen that aerial surveying and mapping is growing at a higher rate [2].

## 1.3. Quadcopters: Brief Overview

Quadcopters are a more specific form of an aerial vehicle known as multicopter having an arbitrary number of rotors. Quadcopters are driven by four rotors. In the 1920s, when quadcopters were first introduced, they were unable to gain popularity due to challenges like having comparatively large size and weight, mechanical complexity, control management, and etc. [1, 2].

In recent years, as a result of advances in electronics, efficient microcontrollers, better sensors, and durable batteries; quadcopters and UAVs managed to attract many researchers and developers to put some research efforts towards the potent utilization, design, and implementation of the quadcopter. Today, many applications of quadcopters can be observed in areas like crop monitoring, video surveillance, military operations, and rescue or search operations and many others. This accelerated growth of UAVs suggests that in the next decade there will be a high demand for the experts who can design and implement UAVs.

## 1.4. Quadcopters and AI

Data in large volumes are often spawned by drones and UAVs. These UAV applications will only be useful for the consumer if this data is managed efficiently and effectively without requiring extra efforts. Therefore, AI (Artificial Intelligence) seems to cover these challenges as nowadays every other company or industry is using AI techniques, machine learning or deep

learning to process immense amount of data. There are many AI based tasks that deal with image recognition, UAVs are also required to perceive and understand the environment, detect and avoid collisions in order to achieve a smooth flight and motion planning is also another task which requires AI and machine learning to play an active part as shown in Figure 2.

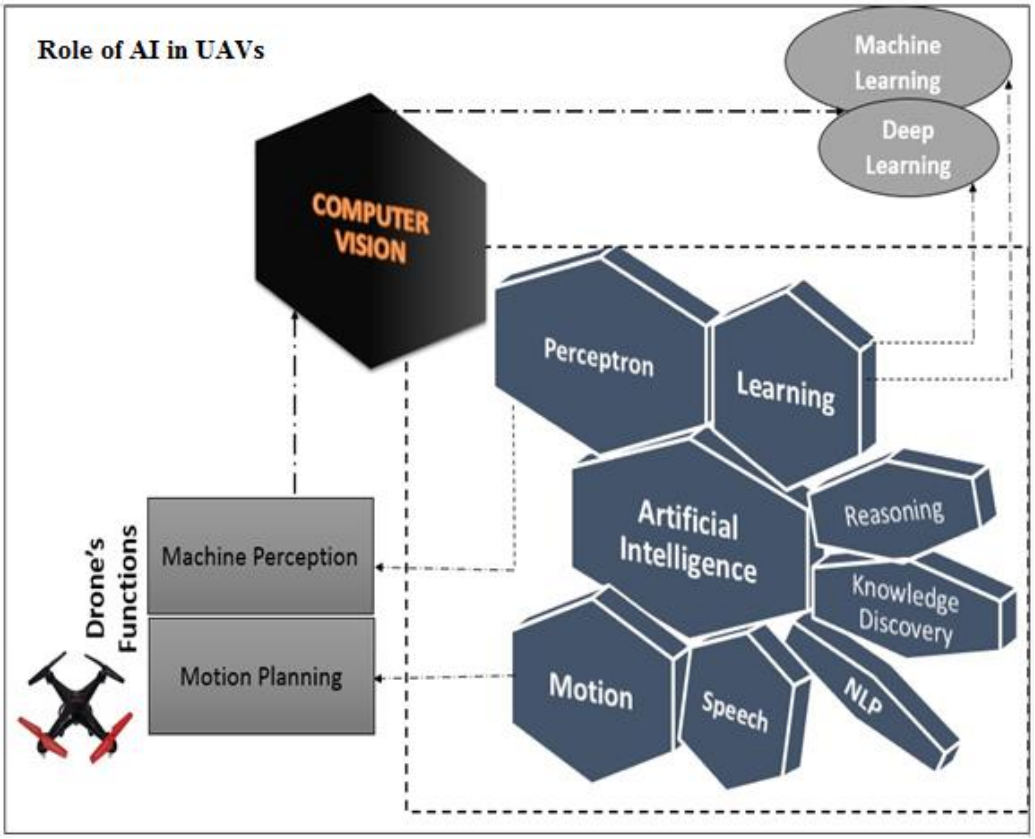


**Figure 2. Role of AI in UAVs**

## 1.5. RL in Drones and Potential Issues

A subfield of machine learning known as RL is gaining success in solving many difficult research problems. In RL an optimal behavior is learned by an agent via interacting and getting a response from the environment. In contrast with supervised learning where the knowledge is already provided to an agent in order to execute a certain task, RL uses a different way of training

and get through rewards and punishments [3]. This method is beneficial when the environment can't be explored completely or easily i.e., it's large and dynamic [4].

RL has many advantages over optimization techniques and supervised learning [5] such as, i) there is no requirement of a predefined controller structure which helps reduce the human efforts and ii) enhances the performance as well. In different researches, it has been proved that for many difficult jobs, a well-trained model will be able to perform better and effective as compared with human experts [6, 7]. In continuous spaces, traditional methods do not perform extraordinarily but RL based methods show promising results [8].

Though for decades robotics are using RL, there are still limitations to the areas where it can be applied. It can help control a UAV, not just controlling its trajectory but also its complete movement with a NN based trained model. In this work, we determine that a UAV can be fully operated and controlled using a NN trained in a RL based simulation environment. We use a DQN which is policy-based as to overcome the limitations of the previously done researches.

## 1.6. Research Problems and Objectives

This work aims to address the limitations of existing research works and hence contributing by taking into account different parameters, listed in Table 1. We propose a model for navigating a drone in an unknown environment without any human help or aid. The goal is to navigate from location A to location B in unknown terrain without having any prior knowledge about the terrain using default drone sensors only. As per the flight is concerned, we want to achieve a fully autonomous flight with a small set of data. We aim to achieve a successful navigation in different large-scale complex environments

**Table 1. Research Objectives**

| Performance metrics/Objectives | Domain | Goal |
|---|---|---|
| **Environment** | 3D/2D | 3D |
| **Observable** | Fully/Partially observable Markov decision process or POMDP. | Partially observable Markov decision process or POMDP. |
| **Prior Knowledge** | Partial/ No Environmental Knowledge | No Environmental Knowledge |
| **Drone flight Autonomy** | Partial/complete | Complete |
| **Dataset** | Large/Small | Small |

## 1.7. Thesis Orientation

The main chapters of this thesis are structured and organized in the following way: Chapter 2 is a literature review. It considers the contributions of existing works in the UAV navigation field using deep learning techniques. Chapter 3 elaborates the proposed system and architecture and explain the layered architecture in detail. Chapter 4 explains the experimental results and performance evaluation of the overall system compared with existing works. Finally, Chapter 5 concludes this thesis.

# Chapter 2

# Related Works

Work in this research is related to autonomous UAV or UAS navigation using RL. Therefore, it is important to discuss some major concepts of machine learning, drones, and to elaborate on the related works done in this area. This chapter deals with the limitations of the existing models and benefits of the proposed mechanism.

## 2.1. UAV and Unknown Terrain

The usage rate of UAV is increasing on a daily basis. It is not only utilized for security surveillance but also for wildfire monitoring, civilian tasks, target tracking, rescue operations, and among many others, military operations. Covering a widespread known or unknown area automatically is its forte and hence having an unprecedented growth rate. The most challenging task is to research on unknown paths or depicting the unknown environment based on prior knowledge of the environment. The UAV is defined as a "powered, aerial vehicle that does not carry a human operator, uses aerodynamic forces to provide vehicle lift, can fly autonomously or be piloted remotely, can be expendable or recoverable, and can carry a lethal or nonlethal payload" [9]. In this case, RL helps us in remotely piloting the drone to the required destination whose path is unknown to us, or we have little information about the environment. UAVs are classified into six functional categories [9]:

a. Target and decoy – yielding ground and aerial force a prey that reproduce an enemy aircraft or missile

b. Reconnaissance – produce intelligence related to battlefield

c. Combat – for missions with high-risk, it provides attacking capabilities

d. Logistics – cargo delivery

e. Research and development – improvement of UAV technologies

f. Civil and commercial UAVs –data collection and photography on the aerial force

## 2.2. Machine Learning Methodologies

Machine Learning can be categorized into supervised learning, unsupervised learning, and RL. Regression and classification are labeled as supervised learning, whereas clustering is classified as unsupervised learning. RL is useful in making real-time decisions, AI games, tasks learning, skills acquisition, and robot navigation.

In this research, we focus on RL and how it is helpful in UAV navigation in an unknown environment. In RL, the input or output is not labeled as supervised learning. It is helpful even if the prior knowledge of the environment is less known or not known at all. As the model works, it learns by itself and thus helpful in our research area. Previously UAV applications have been developed using RL algorithm. RL algorithm depends on feedback and uses trial and error method to maximize the correctness of the output. It is based on MDP which has set of states *s*, actions *a*, rewards *r* and probability of transition *T*. $T(s', a, s) = P(s' \mid s, a)$ describes the effect on the state. It requires the next state'*s'* and the reward depends on the previous state *s* and action *a* [9].

## 2.3. Existing Frameworks

Relatively much work has been going on with respect to automatic UAV navigation with machine learning algorithms but less or no work has been done on unknown path travel by UAV. Su Yeon Choi and Dowan Cha in their paper "Unmanned aerial vehicles using machine learning

for autonomous flight; state-of-the-art" has divided their research areas into three categories parameter tuning, adaptive control and real-time path planning [10].

Hwangbo, Jemin, et al. "Control of a quadrotor with reinforcement learning" deals with policy network that is mapped to the state of rotor thrust instead of high-level trajectory decisions. Recommends controlling and dynamic stabilization from upside down throw. Also encourages new deterministic on-policy method to learn separate policy and value network [11]. Drawbacks of this research is high tracking error and unsatisfactory simulation model accuracy.

Pham, Huy X., et al. "Autonomous UAV navigation using reinforcement learning" proposes a framework to apply RL algorithm to enable UAV to operate in an unknown environment. It uses the Q-learning algorithm using MATLAB. Drawbacks are the deterministic environment. It uses a 2D environment instead of 3D, and another drawback is it is infeasible for the real world [12]. Koch, William, et al. "Reinforcement learning for UAV attitude control" uses flight control system train with DDGP, TRPO and PPO. Compares performance with PID using their GYM-FC environment. Drawbacks are ideal environment considered and decreased convergence [13]. Lambert, Nathan O., et al. "Low Level Control of a Quadrotor with Deep Model-Based Reinforcement learning" proposes a model based RL to rapidly generate low level controllers w/o specific domain knowledge. Disadvantages are only using experience data and model divergence for longer timestamp [14].

Smolyanskiy, et al. "Toward low-flying autonomous MAV trail navigation using DNN for environmental awareness" proposes a micro air vehicle system to follow trails in an unstructured outdoor environment. Drawbacks are NVidia custom hardware chips (TX1+J120) are used and not fully autonomous (Human intervention required) [15]. For non-colliding paths for multi-agent

UAV's cooperative planners have been developed. It has inaccuracies. Geramifard et al. studies on the ICCA for combining cooperative planners and RL techniques as a framework [16 -20]. Zhang et al. [21] propose a CGLA, which is made for path planning on multiple UAV cooperation [22]. In CGLA, to balance between the economy of paths and collision avoidance, the parameters are trained. The individual weight matrix and cost matrix are proposed for an efficient path planning of both single and multiple UAVs [10]. Based on the geometric distance and risk information shared among UAVs, the individual weight matrix is calculated and adaptively updated. The optimal path from a starting point to a target point is calculated. The simulation results validate the effectiveness and feasibility of CGLA for safe navigation of multiple UAVs. However, it is required to utilize the coarse and fine grid to design a fast path planning algorithm in the proposed framework and apply the algorithm to an actual UAV [10].

There are results to avoid a collision for UAV using laser range finders or Kinect cameras. Vandapel et al. [23] avoid a collision in 3D space using lidars, and Bachrach et al [24], and Bry et al. [25] depict obstacle avoidance in an unknown room using scanning lidars for SLAM [23, 24, 25, 26]. Bachrach et al. use Microsoft Kinetic camera in an unknown room [26]. However, these research works are required to improve on problems of power supply, payload, UAVs cost, and the reliability of localization. As a result, Achtelik et al. [27], Wendel et al. [28], Fraundorfer et al. [29] created maps by a single camera, or stereo cameras [26, 27, 29]. Achtelik et al. [30] created sparse maps of the environment using a single, cheap camera [26]. Wendel et al. [20] created dense maps of the environment using a camera [19]. Fraundorfer et al. [29] demonstrated accurate depth estimation and localization using stereo cameras. However, even though the algorithms are reasonably fast to avoid a collision, they were still too computationally expensive for UAVs flight. Recently, many researchers have studied path planning and formation light techniques [32, 13, 32].

## 2.4. Performance Comparisons with Existing Works

In this section, we have compared the recent works in the same field. The performance is evaluated on the following parameters. Table 2 presents a summary of performance comparisons with existing works.

### I. Environmental Model

The environmental model defines the space in which drone navigates. In [12], the 2D environmental model is utilized. In [11], a simplified model in a 3D environment is used where the impact of drag forces on the drone is ignored. In [12], the 3D model is utilized for modeling. In our work, we have considered a complete 1D environment with consideration to all the environmental factors, which include winds, rain as well as drag effect created by these elements.

### II. Observation Space

Observational Space defines how the environment behaves when some action is performed. In the deterministic environment, if given a state, action pair next state can be guaranteed while in a stochastic environment, it's not guaranteed that state, action pair will generate the same output every time. In [11, 12], observational space is set to deterministic while [14, 15, 32] have used stochastic model. We have also utilized the stochastic observational model for our environment as its closer depiction of real-world scenarios.

### III. Observability

Observability refers to how much environmental state is available to UAV at any given point in time. In [12, 15], completely observable environment is utilized. In [11, 15, 32], assumed the partially observable environment is assumed where some part of the state is available at any

given time instead of whole (which is in case of fully observable). We have taken environment observability to be partial.

## IV. NN Architecture

In [14], Model based RL along with random shooter model is deployed. In [12, 32], model free RL is used while in [11, 15], standard DNNs are exploited. In our work, we are utilizing model free RL.

## V. Separate Image Processing Module

In [14], a separate module for image processing is used, but detailed explanations are not specified in the paper. In [11, 12, 32], no camera module is utilized at all. Therefore, there is no image processing module. In [15], a separate module for video processing is utilized. For this purpose, the YOLO module is utilized. In our work, we are not utilizing any separate module for video processing. The data is instead fed into the NN, which utilizes it for generating action probabilities.

## VI. Outside Helper Modules

Ideally, a UAV should not rely on any other external help for its decision making. In [14], however, an external ROS server is used as a helping node. In [12], the extra onboard PID module is used. In [11, 15, 32], any helper modules are not required. In our work, we are not utilizing any external helper modules except the NN.

## VII. Drone Type

Drone type is essential as it plays a huge part in deciding the adaptability of research in real-world scenarios. Off the shelf, drones are the desired scenario as it increases the chances of

adaptability to very high. In [11, 12, 14], off the shelf drones are used. In [15, 32], customized drones are used for the experiment. In our work, we are using off the shelf drone.

## VIII. External Hardware

In [14], an extra ROS server is used, which is deployed on a high-end machine. In [11, 12], no off-board resources are deployed in their work. In [15], TX-1 and AuVedia chips are placed on a drone for processing. In [32], an extra nine range sensors are deployed for the experiment. In our work, we have not utilized any off-board computing resource and have used only off the shelf drone.

## IX. Pre-training

When a model is pushed into drone memory which is already trained on some existing dataset, then that model is called pre-trained model and process is called pre-training. In [12, 14, 32], they have not used any pre-training. In [11], 512 initial and 1024 branching trajectories are utilized for pre-training the model. In [15] IDSIA dataset is being utilized for pre-training. In our work, we have not utilized any pre-training.

## X. Primary Data Source

In [15], state action pairs from the previous 15 iterations are used as an input for the NN. In [12], the 3D environmental state is deployed as its primary data source. In [11], orientation, position, and velocity are exploited as input. In [15], cameras are used as input. In [32],

gyroscope, GPS, and range sensors are deployed as input. In our work, we have utilized the camera, IMU, GPS, barometer, and magnetometer as input.

**Table 2. Comparisons with State-of-the-art Works (Summary)**

| Performance Metrics | Low Level control of quadcopter with Deep model based RL [14] | Autonomous UAV navigation using RL [12] | Control of quadcopter with RL [11] | Towards low flying autonomous MAV Trail navigation [15] | Autonomous navigation in UAV in Large scale environments [16] | Our Approach |
|---|---|---|---|---|---|---|
| **Environment** | 3D | 2D | 3D | 3D | 3D | 3D |
| **Deterministic/ Stochastic** | Stochastic | Deterministic | Deterministic | Stochastic (Limited) | Stochastic | Stochastic |
| **Observability** | Fully | Fully | ( just hovering so not valid) | Partial | Partial | Partial |
| **NN Architecture** | Model based Reinforcement Learning | Deep Reinforcement Learning | Deep Neural Network | Deep Neural Network | Model Free Reinforcement Learning | Model Free Reinforcement Learning |
| **Separate Module for Image Processing** | Yes (Not Specified in paper) | Not Utilized (More GPS based the Camera) | Not Utilized (Only use orientation, Position and velocity) | Yes ( Yolo) | No | No |
| **Extra helper Modules** | Yes (random shooter Model) + ROS server | Yes (PID used ) | No (But they are only hovering) | No | No | No |
| **Drone** | Off the shelf | Off the shelf | Off the shelf | Customized | Customized | Off the shelf |
| **Extra hardware** | ROS Server | No | No | Jetson TX1 , Auvedia 120 Camera *3 | 9 Range Sensors | No |
| **Pre-Training** | No | No | Yes (1.0 Mil Samples) | Yes ( on IDSIA Dataset) | No | No |
| **Primary Data Source** | Previous Action/State Pairs | 2D environment State (X,Y) | Orientation, Position and velocity | 3 Cameras Input | Range Sensors, Gyroscope, GPS | Camera , IMU , GPS, Magnetometer, Barometer |

# Chapter 3

# Proposed Model and Architecture

In this section, we propose a model for navigating a drone in an unknown environment without any human help or aid. The goal is to navigate from location A to location B in unknown terrain without having any prior knowledge about the terrain using default drone sensors only. This work aims to address the limitations of existing research works by taking into account the following parameters as primary contributions of this work:

- A continuous domain is being considered rather than a discrete one
- The simulation environment is 3D
- 3D navigation is enabled, i.e., the quadcopter can navigate across all three-axis x, y, and z.
- High model accuracy (High success rate, low stray and failure rate)
- The model is completely autonomous
- Performance evaluation on comparatively larger datasets.

The rest of this section covers the details of the proposed design and architecture. First, the proposed system design is presented followed by a three-layered system architecture which aims to divide the system into different tiers and explains what is happening at each level.

## 3.1. System Overview

The baseline idea is to autonomously navigate a drone in an unknown environment, which means in whatever environment a drone is, it must be able to navigate effectively and efficiently. This is achieved by using policy-based DQN, a deep learning technique.

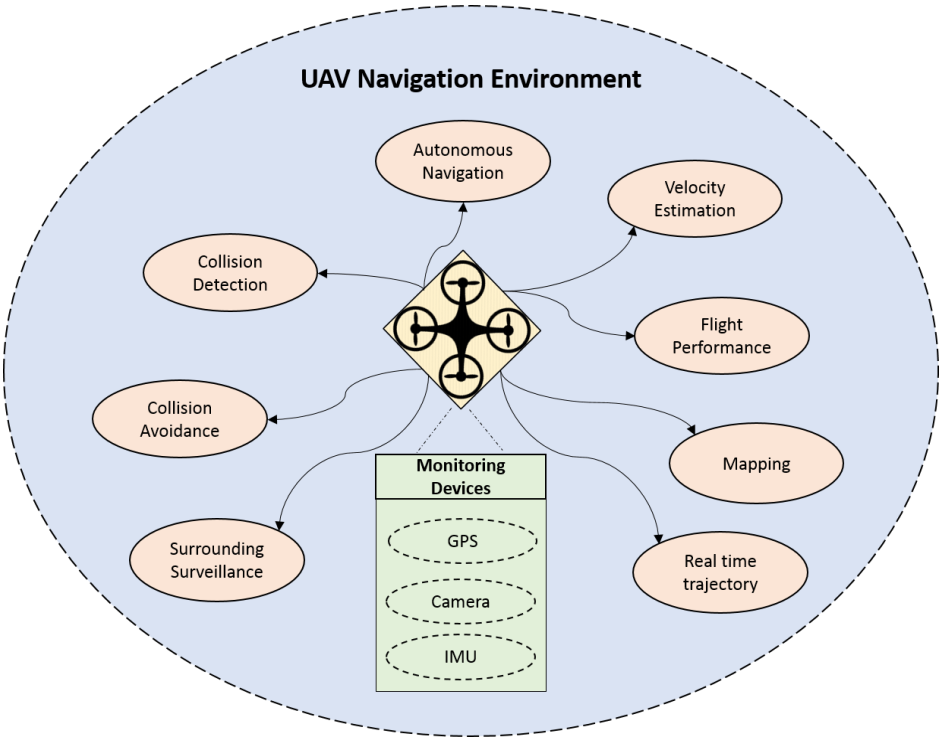


**Figure 3. Conceptual View of Proposed System**

Figure 3 presents the conceptual view of our system, environment, and the functionalities this UAV navigation will enable us with. The quadcopter drone is enabled with monitoring devices, i.e., GPS, camera, and IMU sensors. It navigates autonomously, learns the trajectory in real time, performs environment monitoring, detects obstacles, and avoids collisions.

## 3.2. Proposed Model and System Design

Here, the proposed system design and model is explained in detail. There are five major modules in our system named as *Input*, *Output*, *System*, *Environment*, and *Reward*, as shown in Figure 4. Input deals with the sensor generated values. The environment here represents the terrain where our quadcopter is flying or navigating. Environment generates many parameters like sensor's readings and some states on which the reward function is based.

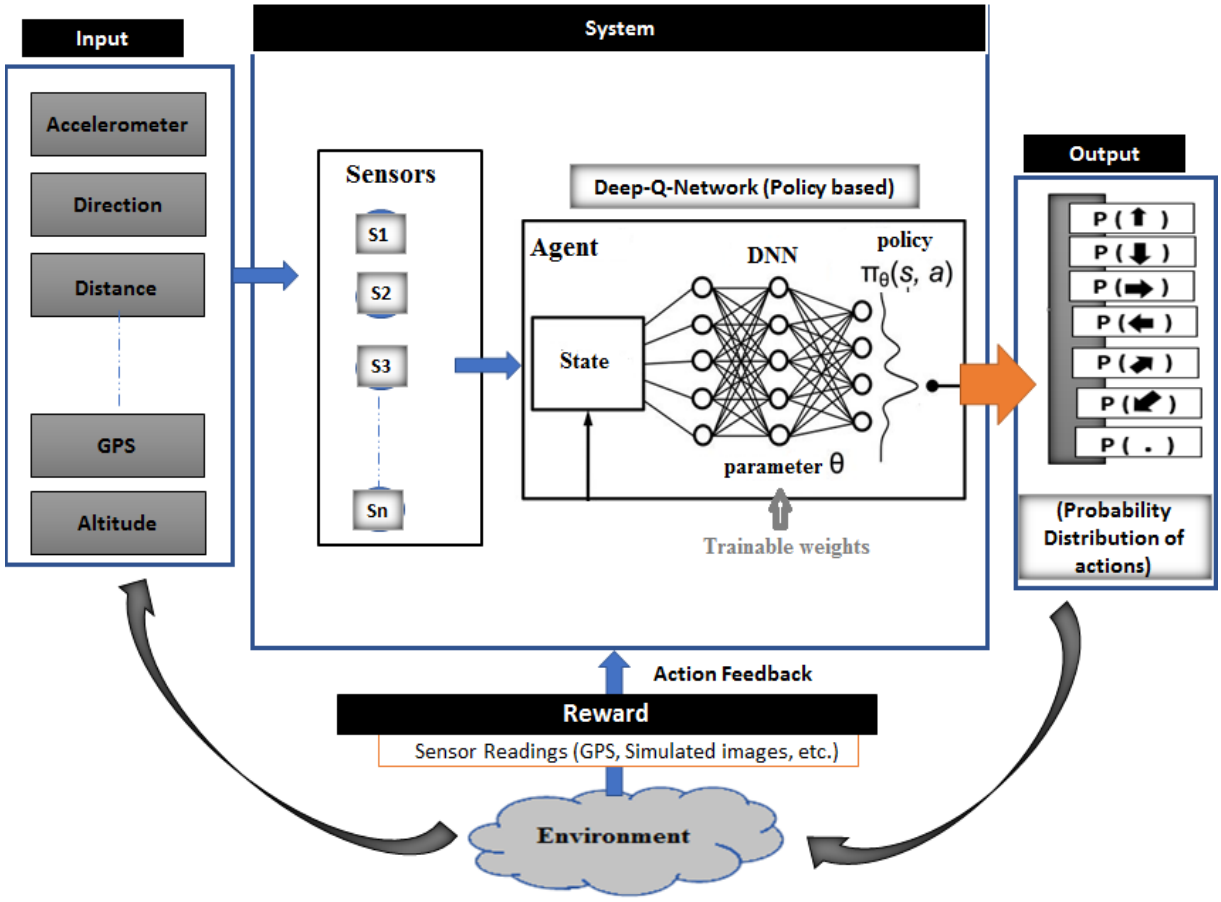


**Figure 4. Proposed System Design**

The reward is calculated as an indicator of an action's feedback. The more the action is performed accurately, the higher is the reward. Next, this reward is forwarded to the system, where our policy-based DQN model (described in next section) processes all the inputs and generates

output in the form of the probability distribution of actions as shown in Table 3. The most favorable action is performed by the quadcopter; action feedback is recorded, passed to our DNN, and the cycle continues.

### 3.2.1. Policy-based DQN vs Value-based DQN

DQN is the product of RL combined with deep learning. DQNs use Q-learning to learn to take an optimal in a given state (the Q-value) and use CNN as an approximating function for the Q-learning function. It is represented and denoted as $Q(s, a; \theta)$, where $\theta$ represents the trainable weights of the network.

**Table 3. Description of Actions**

| Actions | Action's Probability Representation | Description |
|---|---|---|
| **Move up** | **P ( ⬆ )** | P (Up): Probability of quadcopter to move in an upward direction |
| **Move down** | **P ( ⬇ )** | P (down): Probability of quadcopter to move in a downward direction |
| **Move right** | **P ( ➡ )** | P (Right): Probability of quadcopter to move in the right direction |
| **Move Left** | **P (⬅ )** | P (Left): Probability of quadcopter to move in the left direction |
| **Move Forward** | **P ( ↗)** | P (Forward): Probability of quadcopter to move in the forward direction |
| **Move Backward** | **P ( ↙ )** | P (Backward): Probability of quadcopter to move in the backward direction |
| **Hover at place** | **P ( . )** | P (Hover): Probability of quadcopter to a place or do nothing |

In this work, we are using DQN to predict the policy directly instead of Q-value. For comparison we are also implementing a value-based DQN. Both the algorithms utilize same NN

architecture. Value-based DQN treats the output of neural network in a different way as compared to policy-based DQN. In case of value-based DQN, output action is the one which has the maximum value in the output layer. In case of policy-based DQN the output from NN is treated as probability distribution. Actual action is then sampled from this probability distribution. The advantage of using policy-based approach is it works better for continuous domains.

Figure 4 is the complete system design covering all the inputs, methods, and outputs. The elements of DQN are explained in Table 4. An *agent* is an entity which decides and takes certain actions. *State* refers to the internal state of our DNN, e.g., learning weights. An *action* is a decision made by the agent to make quadcopter do some movement, e.g., moving up or down. The *reward* is a measure of how good an agent is performing.

**Table 4. Key RL Elements**

| Key RL Terms | Explanation | Examples |
|---|---|---|
| **Agent** | Decision-making entity that takes action | DQN |
| **State** | NN internal weights | Camera, GPS |
| **Action** | A decision made by the agent to take any of the actions mentioned above | Quadcopter moving left or right |
| **Reward** | How good agent is performing | Distance from goal |

## 3.3. Layered View of Architecture

The overall system architecture is presented in this section. The system consists of a layered architecture, as described in Figure 5. Layer I, is the physical layer. It consists of our simulation environment, sensors, and actuators. Data and control layer (Layer II) deals with data from sensors,

formatting this data to be interpretable by the processing layer. The processing layer (Layer III) is based on a DNN trained on the data collected through sensors. The functions it performs include actions to be taken by the drone. These actions might include hovering, drone movement in forward or backward direction.

Once the actions are decided, the AI module forwards these actions to actuators, i.e., electronic speed controllers (ESCs), actuators perform these actions. This process happens in a loop after the actions are taken, new data is being collected through sensors, and new actions are

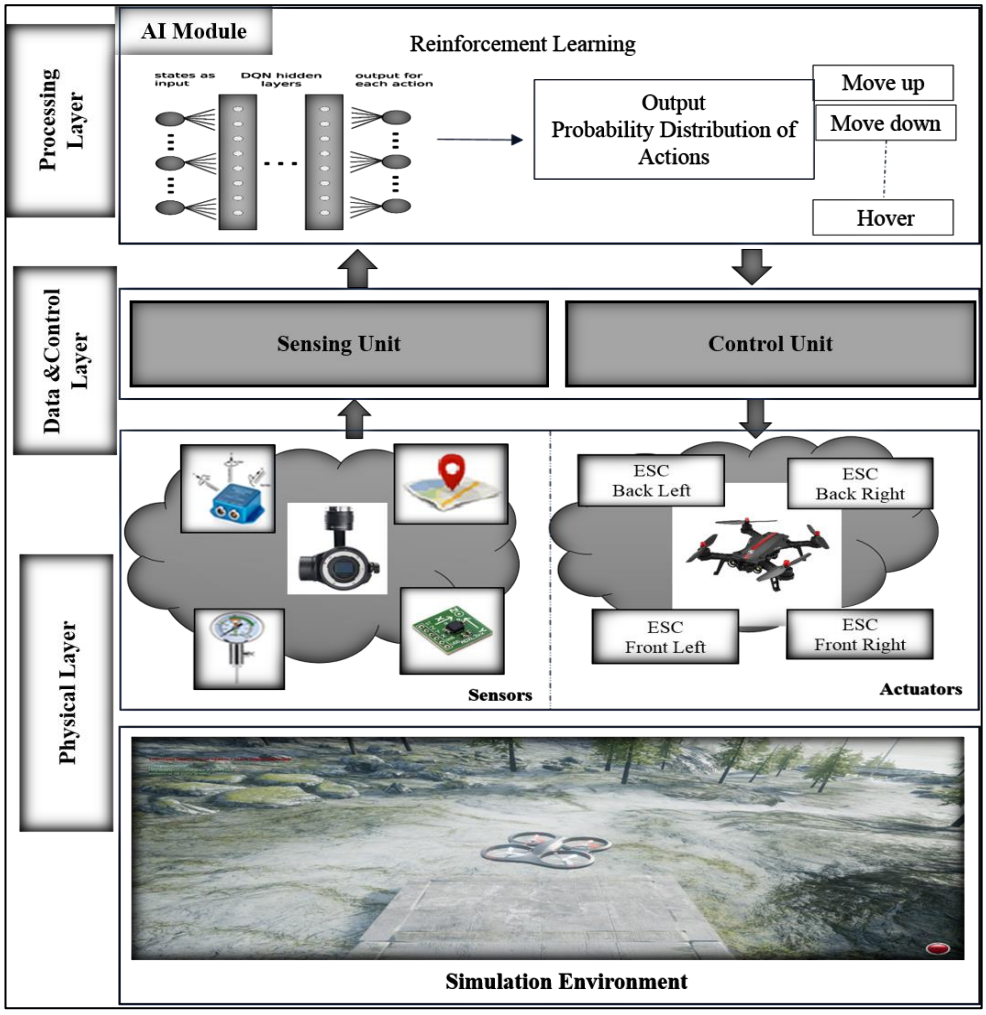


**Figure 5. Proposed Architecture- Layered View**

decided by the AI module, and hence, this cycle continues to repeat. Each layer is explained in detail below.

### 3.3.1. Physical Layer

This is layer I, which comprises of our simulation environment, sensors, and actuators.

#### A. Simulation Environment

To experiment with deep learning, RL, and computer vision algorithms for UAVs, we have used AirSim as a simulation tool. Figure 6 is a screenshot of our simulation environment. AirSim enables us to use different APIs to retrieve images, control the vehicle, etc. Table 5 presents the components and their details for the simulator. AirSim is used as a platform for simulations. It provides different APIs to enable communication with vehicles in the simulation environment. Some example APIs are given in table below.

**Table 5. Details of the Simulation Environment**

| Component | Name | Details |
|---|---|---|
| **Simulation tool/Platform** | AirSim | Open-source simulator for UAVs, cars and more |
| **AirSim APIs** | simGetImage,<br>reset,<br>confirm connection,<br>simGetObjectPose,<br>simGetCollisionInfo | To retrieve images, get state, control the vehicle and for many other tasks |
| **Vendor** | Microsoft | |
| **Sensors** | Camera, GPS<br>IMU,<br>magnetometer, barometer | AirSim currently supports these sensors data |

These APIs can be deployed on a computer on the vehicle with which you aim to interact, in our case, its quadcopter as these APIs are also accessible as an independent cross-platform library. Code written in the simulator can later be executed on the real UAVs.


**Figure 6. Screenshot of Our Simulator**

In this thesis, we have experimented drone's navigation in three different environments named as *Blocks, Landscape,* and *Neighborhood*. All these environments are supported by AirSim. These environments are open source and are readily available on unity's unreal engine store [33] for free. Following is an explanation about the environments along with screenshots.

### i. Blocks

*Blocks* is the default environment that comes with AirSim simulator. It's a simple environment as it spans by few buildings and obstacles of arbitrary height over an area of half kilometers. The major obstacles in this environment are building blocks along with a spherical object and a small cone as shown in Figure 7.

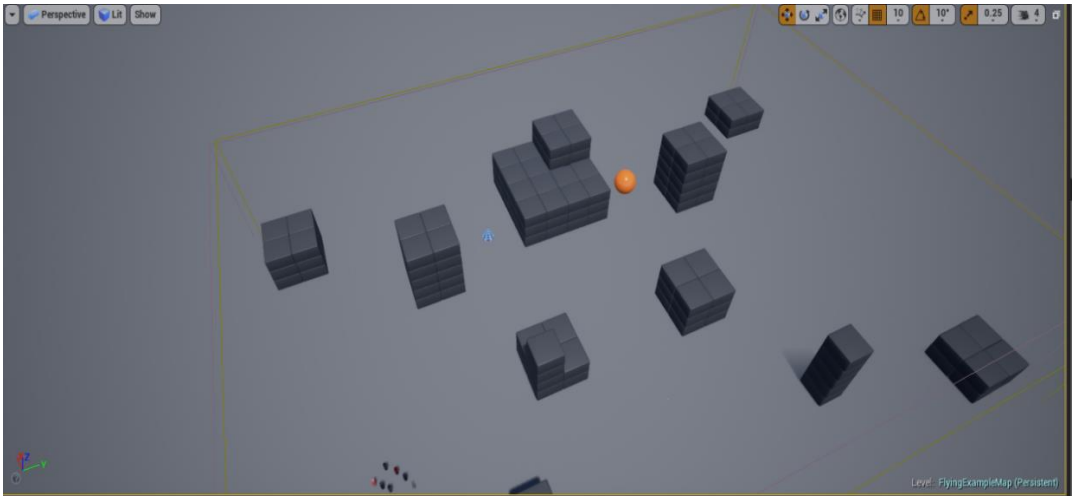


**Figure 7. Screenshot of *Blocks* Environment**

### ii. Landscape

*Landscape is* a well-known simulation environment. It's a large size environment like a jungle. In this environment the major objects are trees, buildings and power transmission lines. It also contains a road on the right side of the map as shown in Figure 8.


**Figure 8. Screenshot of Landscape Environment**

### iii. Neighborhood

*Neighborhood* is a medium size environment which is based on a busy urban neighborhood. There are many obstacles in this environment such as buildings of random heights, cars, trucks, trees, houses, and traffic signals as shown in Figure 9.

**Figure 9. Screenshot of Neighborhood Environment**

Table 6 summarizes the different aspects of these environments.

**Table 6. Simulation Environment Details**

| Environment | Dimensions | Number of obstacles | Major obstacles | Size (MBs) | Open source | Computability OS |
|---|---|---|---|---|---|---|
| Blocks | 400 * 600 m | 10 | Buildings | 650 MB | Yes, Default Airsim environment | Windows / Linux |
| Landscape | 2 * 2 Km | 1000+ | Buildings, Trees, Power lines | 4.3 GB | Yes, Available on unity assets store for free | Windows / Linux |
| Neighborhood | 1.2 *1.2 Km | 1000+ | Cars, Trees, Homes, Traffic Signals | 4.5 GB | Yes, Available on unity assets store for free | Windows / Linux |

## B. Drone Sensors and Actuators

We have used GPS, camera, magnetometer, IMU, and barometer, which are currently supported by AirSim. There are four ESCs in total, working as actuators. Figure 10 presents the default configuration of our sensors.

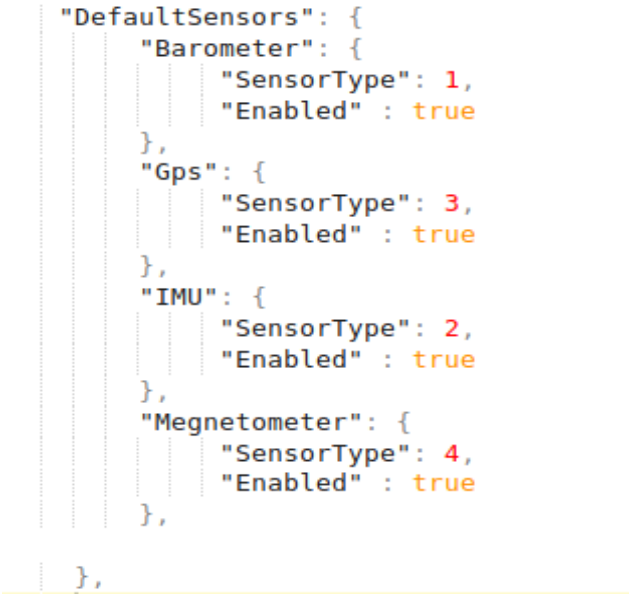


**Figure 10. Sample Drone Sensors Configuration**

There are in total four ESCs; *Front Right*, *Front Left*, *Back Right*, *Back Left* as shown in Figure 5. Following are the details of the sensors we used.

### i. GPS Sensor

It is used to calculate the drone's position accurately, also to collect the coordinates of the destination. Figure 11 shows the GPS data in our scenario; latitude and longitude describe the exact location of the drone at a specific timestamp. Velocity is another important parameter which describes the velocity of the drone with respect to x, y, and z-axis at the time of measuring the GPS coordinates.

```
gps_data: <GpsData> {   'gnss': <GnssReport> {   'eph': 0.3000044822692871,
    'epv': 0.40000447630882263,
    'fix_type': 3,
    'geo_point': <GeoPoint> {   'altitude': 123.34488677978516,
    'latitude': 47.64197914136834,
    'longitude': -122.14017833284416},
    'time_utc': 1560199207884190,
    'velocity': <Vector3r> {   'x_val': 0.0,
    'y_val': 0.0,
    'z_val': 0.0}},
    'is_valid': True,
    'time_stamp': 1560199207884190720}
```


**Figure 11. Screenshot of our GPS Sensor Data**

### ii. Magnetometer Sensor

It is used to handle the wind effect. It calculates the value of magnetic north based on which it can measure the angle of deviation or the amount of drift caused by wind or some other factors, e.g., rain. Figure 12 shows the magnetometer data in our scenario.


```
agnetometer_data: <MagnetometerData> {   'magnetic_field_body': <Vector3r> {   'x_val': 0.24983103573322296,
   'y_val': 0.03394465148448944,
   'z_val': 0.36363229155540466},
   'magnetic_field_covariance': [   ],
   'time_stamp': 1560199208079194880}
```


**Figure 12. Screenshot of our Magnetometer Sensor data**

### iii. IMU Sensor

IMU repeatedly compute the existing position of a drone. It first incorporates the sensed acceleration, with gravity estimate, to compute the current velocity of the drone. Then this velocity is being used to calculate the current position. Figure 13 shows the IMU data in our scenario.


```
imu_data: <ImuData> {   'angular_velocity': <Vector3r> {   'x_val': 0.0018179158214479685,
    'y_val': 0.0021809835452586412,
    'z_val': -0.0007190685137175024},
    'linear_acceleration': <Vector3r> {   'x_val': -0.0013736896216869354,
    'y_val': -0.01396923791617155,
    'z_val': -9.800806999206543},
    'orientation': <Quaternionr> {   'w_val': 1.0,
    'x_val': 0.0,
    'y_val': 0.0,
    'z_val': 0.0},
    'time_stamp': 1560199208094195200}
```


**Figure 13. Screenshot of our IMU Sensor Data**

### iv. Barometer Sensor

It is used to measure air pressure. Figure 14 shows the barometer sensed data in our scenario.

```
barometer_data: <BarometerData> {   'altitude': 123.31890106201172,
    'pressure': 99851.7109375,
    'qnh': 1013.25,
    'time_stamp': 1560199208079194880}
```

**Figure 14. Screenshot of our Barometer Sensor Data**

### v. Camera Sensor

We used depth vision as a camera for collision avoidance, detection, or looking ahead. Figure 15 shows the comparison between a normal camera view and depth vision camera view. \

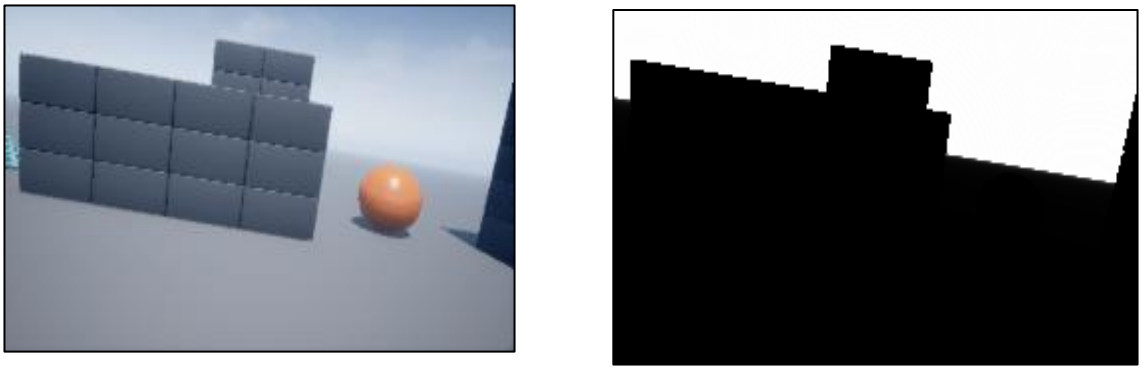

**Figure 15. Screenshot of Image taken by Normal camera vs Depth Camera**

On the left side, a normal camera view is shown. On the right side, the same view is shown but from a depth vision camera. The way to measure depth is using the intensity of black pixels, i.e., the more black a pixel is, the nearer the object is.

### 3.3.2. Data and Control Layer

This layer consists of sensors unit and actuators unit. This layer acts as a bridge between the physical layer and the processing layer. The data from the physical layer collected through sensors is sent to the sensing unit where it's stored in a specific format and is forwarded to the processing layer. After processing the output in the form of actions is passed to the control unit. Control unit interprets it and forwards it to actuators to perform a specific action.

### 3.3.3. Processing Layer

This layer comprises of our DNN model, i.e. policy-based DQN. RL is a machine learning technique in which an autonomous agent acquires to discover an optimum performance over trial and error exchanges with its environment [5]. In our proposed network model, there are six layers in total, i.e., one input layer, two convolutional layers, two fully connected layers, and one output layer.

The input data is collected through different sensors. First, images are taken as input; different convolutions are performed on it to extract features, as shown in Figure 16. These features are flattened, and convolutions are performed again to extract low-level features, which result in dense vector representation. Adam optimizer is used for optimization.

Other sensors data, e.g., data from IMU, barometer, and GPS, are combined with the dense vector representation, and all these features are passed to two fully connected layers. Output layer presents the probability distribution of all the possible actions as shown in Table 3.

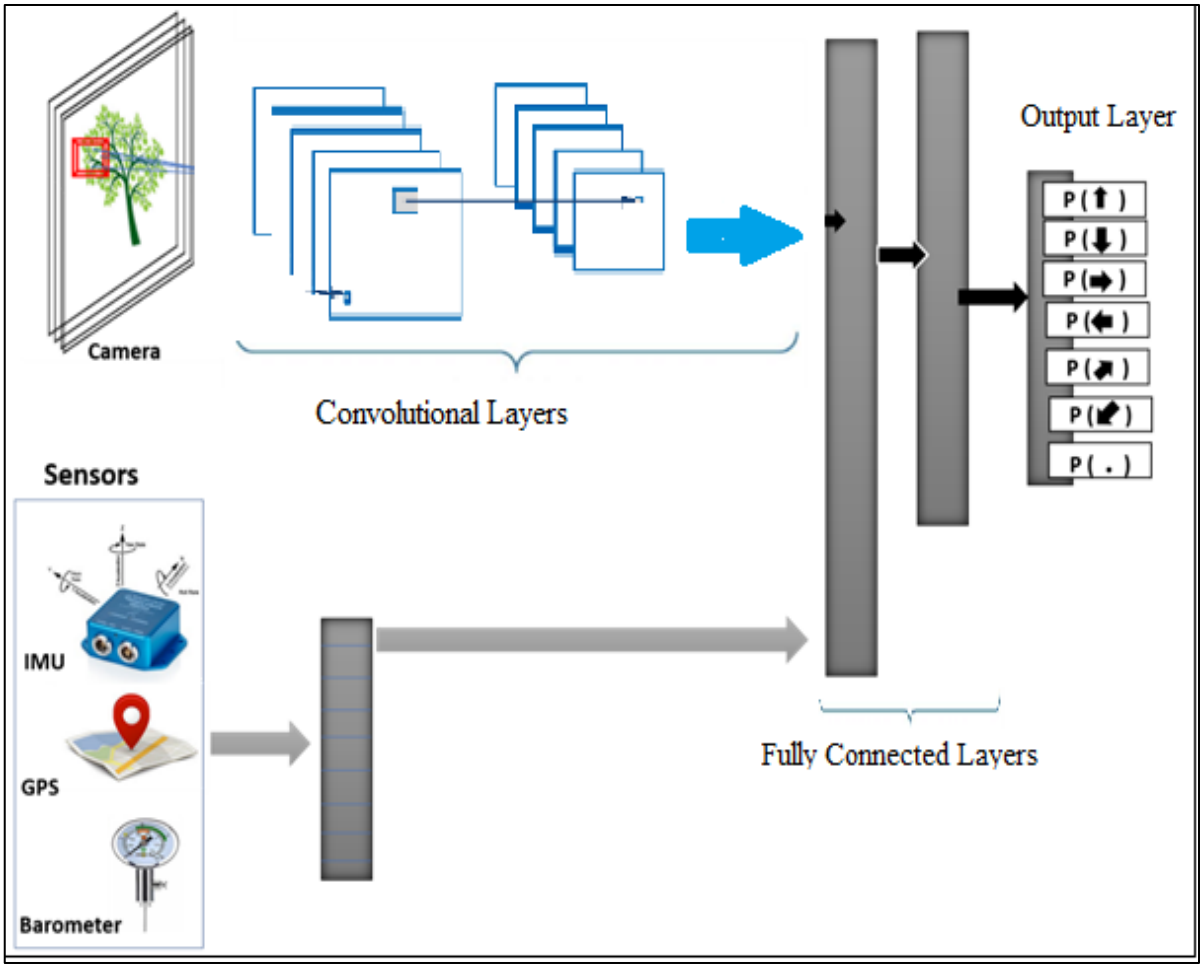


**Figure 16. NN architecture**

Table 7 lists the neural network specifications used to create the model and to achieve effective performance results.

Our NN takes camera pixels as an input. The input is [192 * 256 * 3] where 192 is the image width, 256 is the image length and 3 are RGB frames. This input is passed through three different convolutional layers which have 16, 32, 32 filters respectively. This generates a dense vector representation of the image.

**Table 7. Implementation Details**

| Components | Techniques |
|---|---|
| AI module | Value-based DQN |
| Network Training | Backpropagation |
| Optimizer | Adam Optimizer |
| Loss Function and Reward | DQN |
| Number of Hidden layers | 5 |
| Camera Input dimensions | [192*256*3] = 147,456 |
| Sensors Input dimensions | [10 * 1] + [3 *1] + [10*1] + [3*1] = 26 |
| Camera Hidden Layer1 | Filters = 16 * [8*8], stride= 4 |
| Camera Hidden Layer2 | Filters =32 * [4*4], stride= 2 |
| Camera Hidden Layer3 | Filters= 32 * [3*3], stride= 1 |
| Dense Layer 1 dimensions | 256 * 1 |
| Dense Layer 2 dimensions | 128 * 1 |
| Output Layer dimensions | 7 * 1 |

On the other side, all the sensors data is accumulated and then normalized. Normalization is done to map different sensor input values into same range [0-1]. These normalized input values are then passed to fully connected layer 1 along with dense vector representation. This layer has 256 neurons. Its followed by another layer with 128 neurons followed by output layer. Output layer has 6 neurons each of which is the probability of action which neuron can take.

## 3.4. An Example of Simulation Scenario

Here, an example of the simulation environment is presented. We have taken a block environment and run few iterations. A starting point is selected for the drone to start navigation towards the target or goal called as finishing location. Each obstacle in the environment has a specific height. There are three types of blocks or obstacles based on heights, the small size blocks are of height range up to 50m, medium size blocks are in height range of 51-100m and obstacles above 100m are categorized as large blocks.

**Table 8. Simulation Parameters**

| Parameter | Value |
|---|---|
| Environment | Simulation |
| Drone's Max. altitude | 200m |
| Block types<br>(Based on heights) | Small, Medium, Large |
| Block sizes<br>Height ranges | Small: [0-50m]<br>Medium: [51-100m]<br>Large: [100m+] |
| Simulation Input | Drone Movements (Up, Down, Left, Right, Forward, Backward) |
| Simulation Output | Sensor Readings (GPS, Camera, Magnetometer, Barometer) |
| Neural network Input | Sensor Readings (GPS, Camera, Magnetometer, Barometer) |
| Neural network Output | Drone's Next Movement (Up, Down, Left, Right, Forward, Backward) |
| Reward Function | 1. (Euclidian distance between current location and final destination) + collision penalty |

The details of the simulation environment used to illustrate how a drone navigate from one point towards its goal are shown in Table 8.

### 3.4.1. Environment Details

The complete environment can be shown in Figure 17. The drone can move in all six directions in this environment. When the environment starts the neural network takes sensor readings from the simulator. It decides to make a move based on these readings. NN then outputs next movement for the drone, simulator implements that movement and returns the next sensor readings.

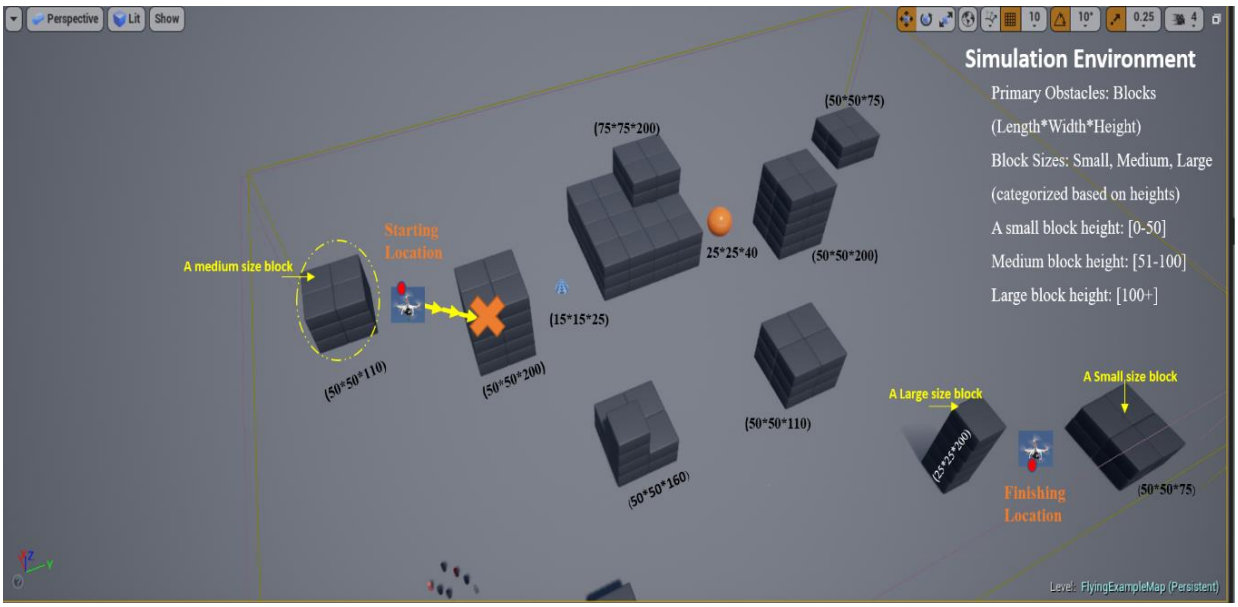


**Figure 17. Simulation Environment Primarily Consisting of Blocks of Different Heights as Obstacles**

After this process, a reward is calculated. This reward is a simple fusion of Euclidian distance and collision penalty. To be precise, it's a sum of the negative distance from the drone's current location to its final destination and collision penalty. Collision penalty is essentially a penalty which occurs when a drone collides with some obstacle. Whenever a collision occurs, the drone is reset to its initial position and the iteration starts again.

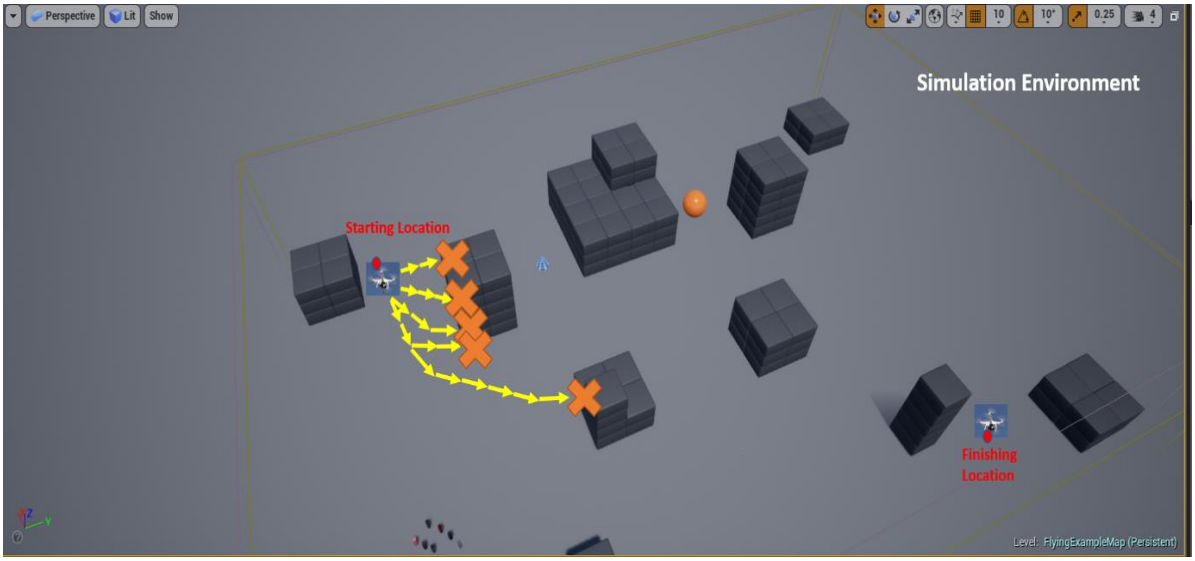


**Figure 18. Drone's Navigation Path After Some Iterations**

When the simulation starts, this cycle of inputs to the neural network from the simulator and corresponding movement from the neural network to simulator begins. The neural network is trying to maximize the reward all the way possible. The drone follows the path shown in yellow as shown in Figure 18. It collides with the obstacle Infront of it. This is because it didn't anticipate that it would collide with obstacle if it only considered reducing the distance (maximize reward). This collision will incur a high penalty and very low reward. This will discourage the drone from following this set of moves again.

Over multiple iterations, the neural network will try a different set of moves to maximize the reward. After a certain amount of iterations, it will learn that getting too close to an obstacle can result in a collision. Due to this, it will try to make moves which increase the reward but at the same time also fly not too close to an obstacle. Once it's able to learn rules like these, it will start to navigate in the environment successfully. After a neural network is fully trained, successful

navigation path from the starting location towards the goal destination is shown in Figure 19. More information about the environments is provided in the Appendix A.

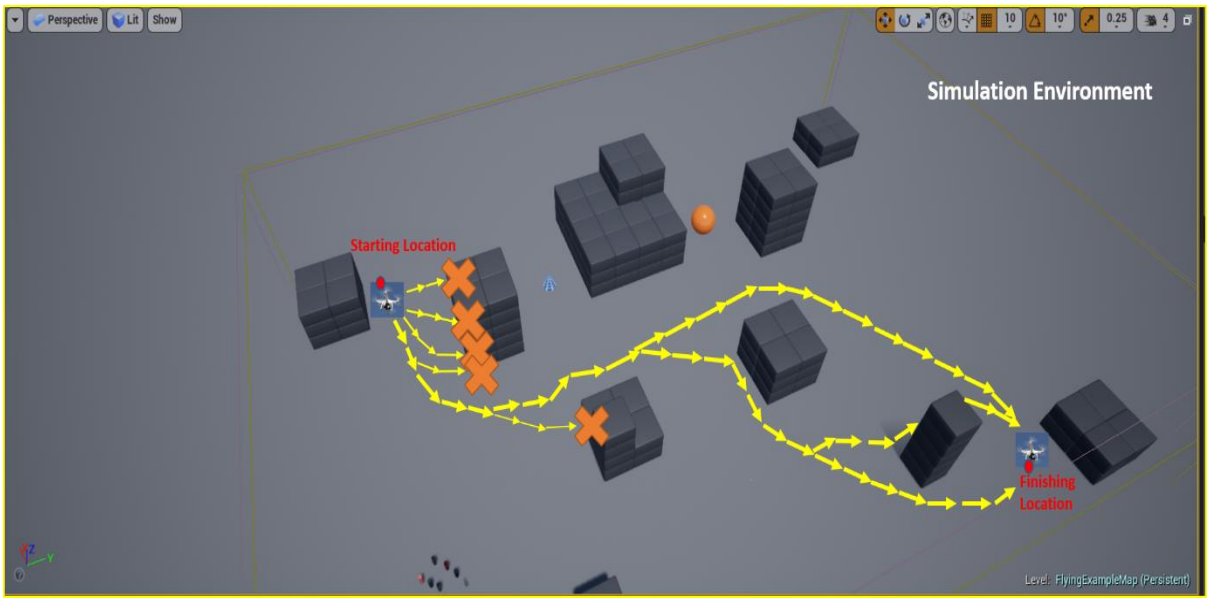


**Figure 19. Drone's Successful Navigation Path from starting point towards its Goal**

## 3.5. System Specifications

Table 9 presents the system specifications. We used python 3.6 as a programming language, Tesorflow version 1.13, Cuda version 10.0, cudann version 7.4.2.

**Table 9. System Specifications**

| Components | Versions |
|---|---|
| Operation System | Linux (x64) |
| Python | 3.6 |
| Tesorflow | 1.13 |
| Cuda | 10.0 |
| CudaNN | 7.4.2 |

# Chapter 4

# Experimental Results

This section explains the simulation results. The simulator described in the previous section is used to execute the experiments.

## 4.1. Experimental Setting

The drone is initially spawned in a random location on the map, and the final destination is set to a predefined location (usually midpoint in the map). The goal of the drone is to fly from its random initial position to the final destination without any collisions. If the drone collides with any obstacle, then the episode ends, and the drone is restarted from the initial position. For a drone, we selected a standard off the shelf drone. We are utilizing gyroscope, magnetometer, GPS, and camera as sensors in a simulation environment. The configuration file for these sensors is shown in Figure 7.

We used three types of large-scale environments, each of a different kind. The first environment is called *Blocks* as it spans with multiple buildings and obstacles of arbitrary height over an area of half kilometers. The second environment is called *Landscape*, a well-known simulation environment, here, drone navigates in a jungle. The third environment is called *Neighborhood;* it is based on an urban city location where the drone is supposed to navigate in an environment crowded with buildings of random height. During implementation, the drone is trained in each environment separately, where on every collision, it finishes one iteration.

Whenever an agent initiates an interaction with the environment, several objects and obstacles are generated based on the environment it chooses for navigation.

The speed and altitude of the UAV range between 0 to10m/s and 5 to 200m respectively. The speed and altitude can change depending upon the moves or actions UAV takes. The episode length (i.e., time step) can be up to 200. The network parameters are learned by using Adam Optimizer with a learning rate of $10^{-3}$. The discount factor $\gamma$ for the experiments is varied in the range of 0.91 to 0.99, as shown in Table 10.

**Table 10. Values and Range of Different Simulation Factors**

| Simulation components | Value/Range |
|---|---|
| Simulation Environments | Three [Blocks, Landscape, Neighborhood] |
| Sensors | Gyroscope, Magnetometer, GPS, Camera |
| UAV speed | 0 to 10 m/s |
| UAV altitude | 5 to 200 m |
| Episode length | 200 |
| Learning rate | $10^{-3}$ |
| Discount factor | 0.91 to 0.99 |

## 4.2. Reward Function

The reward is the incentive that the algorithm gets when it makes the desired move. In our scenario, the reward is the combination of the distance between target and collision. A simple example would be distance plus collision. If there is no collision, then reward is +10 and distance covered in the shortest path. In case of collision, the reward would be -10 and the learning episode will end. In our case, the reward value ranges from 0 to 200. The rewards get to 200 when the

UAV reaches the destination, and the iteration ends. The environment is continuous; therefore, the state is the reading gathered from sensors.

We used Bellman's equation for the optimization of our DQN.

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

Here *r(s, a)* represents the immediate reward that we get from that action. *Q(s', a)* is the reward that is achieved when we get into state *s'* after taking action *a*. Gemma is the discount factor that we use to give preference to current reward and discourage future reward. In our scenario, the possible actions would be moving forward, backward, up, down, left, right, and hover. The state would be the sensor readings. The DQN network will predict Q-Value, and we will optimize the network using the following cost function.

$$Cost = \left[ Q(s,a;\theta) - \left( r(s,a) + \gamma \max_{a} Q(s',a;\theta) \right) \right]^2$$

In this equation, we calculate the cost of the predicted value vs. the original cost. Here the first part of the equation will be the original value, and the second part would be predicted by the network. In training iterations, we continuously optimize this loss.

## 4.3. NN Training

The following graph shown in Figure 20 presents a trend of reward function values when we run the program for 500 epochs. It can be observed that the network was learning very slowly. This is a known behavior as the network is experimenting with different outputs to check which

strategy works. After a few hundred iterations, it starts to learn about some successful strategies, and reward gradually starts to build up.

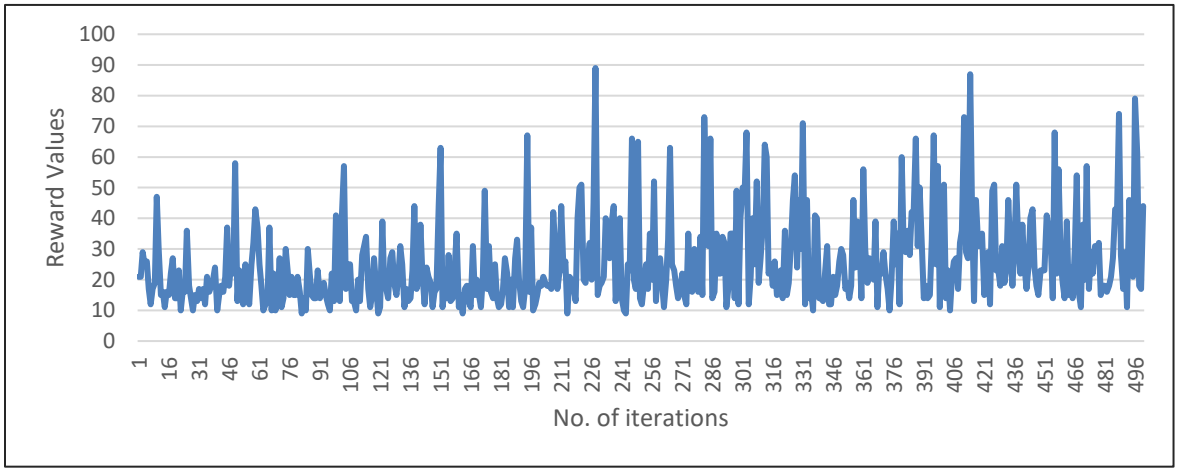


**Figure 20. Reward Function (500 Iterations)**

The only measurable factor that can quantify that the DQN algorithm is working properly is a reward. In our scenario, the reward function comprises of the combination of hubber loss and distance from the destination.

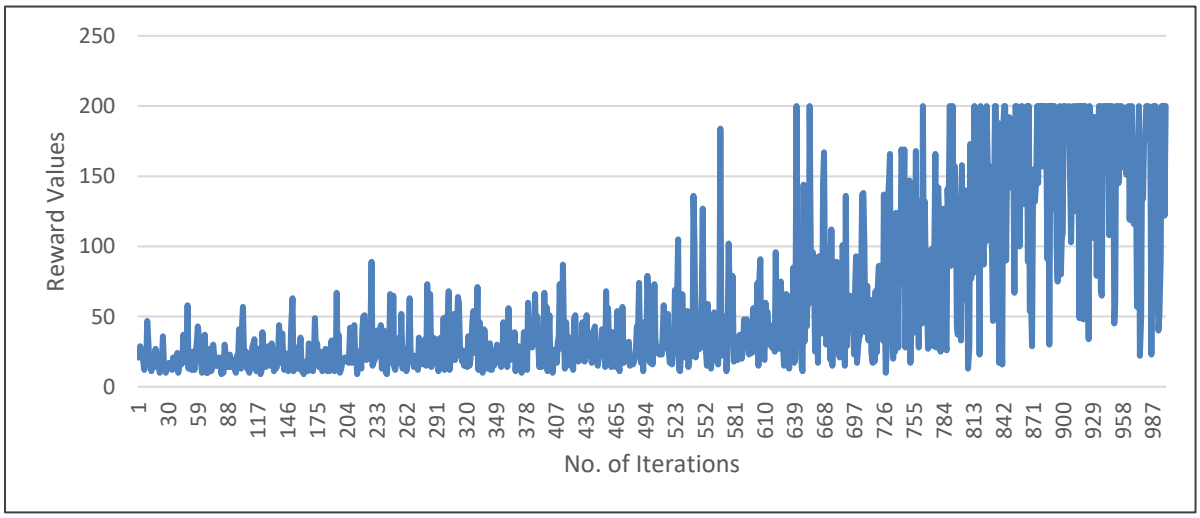


**Figure 21. Reward Function (1000 Iterations)**

The graph shown in Figure 21 shows a trend of reward function values when we run the program for 1000 epochs. After a few small tweaks and an increasing number of iterations to 1000, we can observe that the reward is maxing out. The reward was capped at 200. As the iterations increase, the reward values show a rapid increase, respectively.

## 4.4. Performance Evaluation

In this section, we have evaluated the performance of our proposed model both on qualitative and quantitative measures using both policy-based DQN and value-based DQN, as shown in Table 11.

**Table 11. Types of Performance Metrics**

| Performance Metrics | Description |
|---|---|
| Quantitative measures | System Round Trip Time, Avg. the execution time of each iteration, Effect of gamma, Different learning rates, varying iterations |
| Qualitative measures | Model free RL, Learning based on no prior knowledge (No large datasets required for training), efficient, Scalable |

### 4.4.1 Quantitative Evaluation

In this section, we have evaluated the performance of multiple implementations (value-based vs. policy-based DQN) based on different metrics. The first parameter which affects the performance of UAV is the RTT or Round-Trip Time, as shown in Figure 22. RTT pattern for both value-based DQN and policy-based DQN.

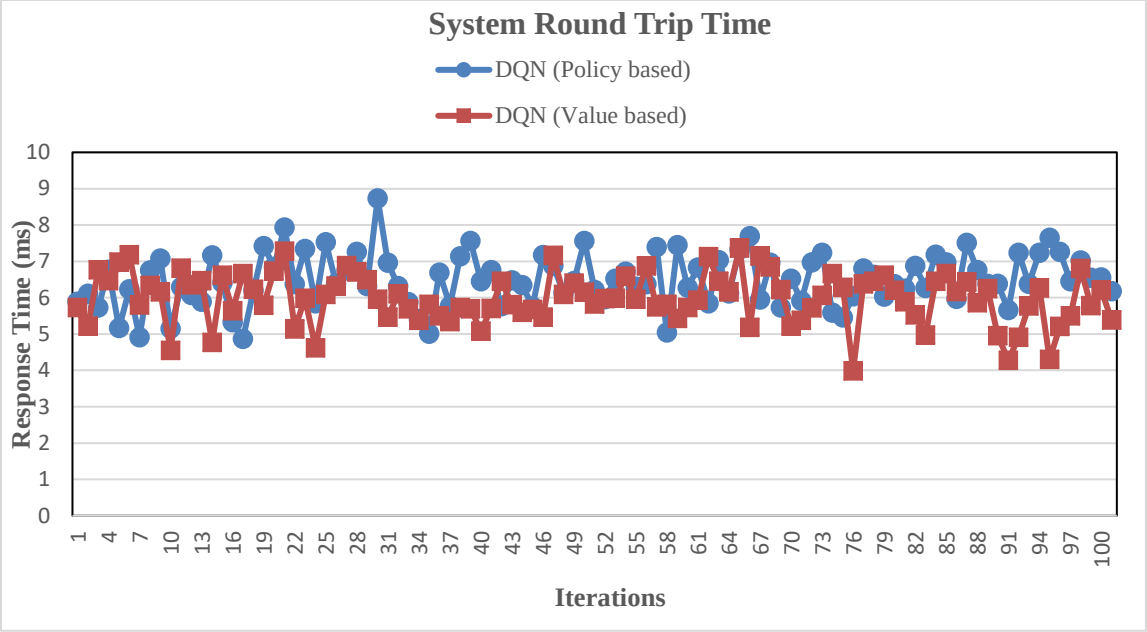


**Figure 22. System Round Trip Time vs. Iterations**

In our scenario, it comprises of taking sensor readings from simulation, feeding it into the DQN, getting the predicted action, forwarding that action to the simulator and finally getting new sensor readings. This cycle is an important metric as it allows us to calculate the real-time performance of our algorithm. In value-based DQN the RTT is slightly lower than policy-based DQN. The reason for this increase in RTT is the extra probability sampling that is done in policy-based DQN but is not present in value-based DQN. For policy-based DQN, the mean value for RTT is 6.5 milliseconds while for value-based DQN this value is 6.0 milliseconds.

Average execution time for each iteration is an important parameter for measuring the performance of the DQN. Figure 20 shows the pattern of execution time for each iteration for the first 180 iterations. Execution time shows the time in seconds that UAV navigated in the environment before colliding with an obstacle.

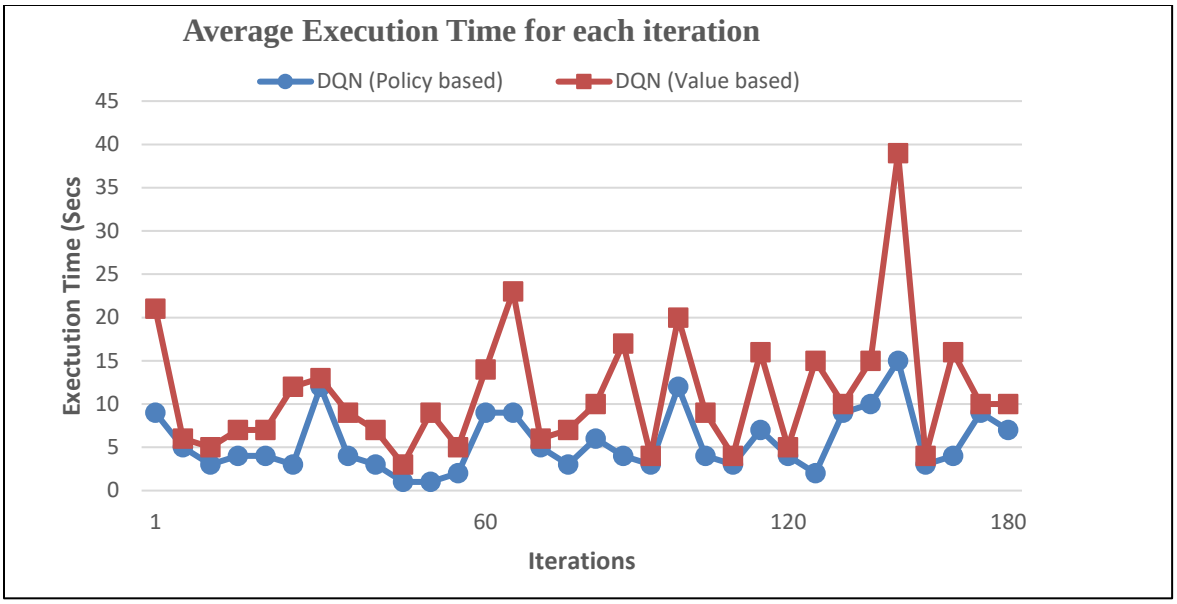


**Figure 23. Average Execution Time for each Iteration**

For initial iterations, DQN usually tries to figure out the policy that works better. In this process, the execution time is relatively unstable and is varying. In Figure 23, both policy-based DQN, as well as value-based DQN, are showing a slight increase in execution time as the iteration count increases.

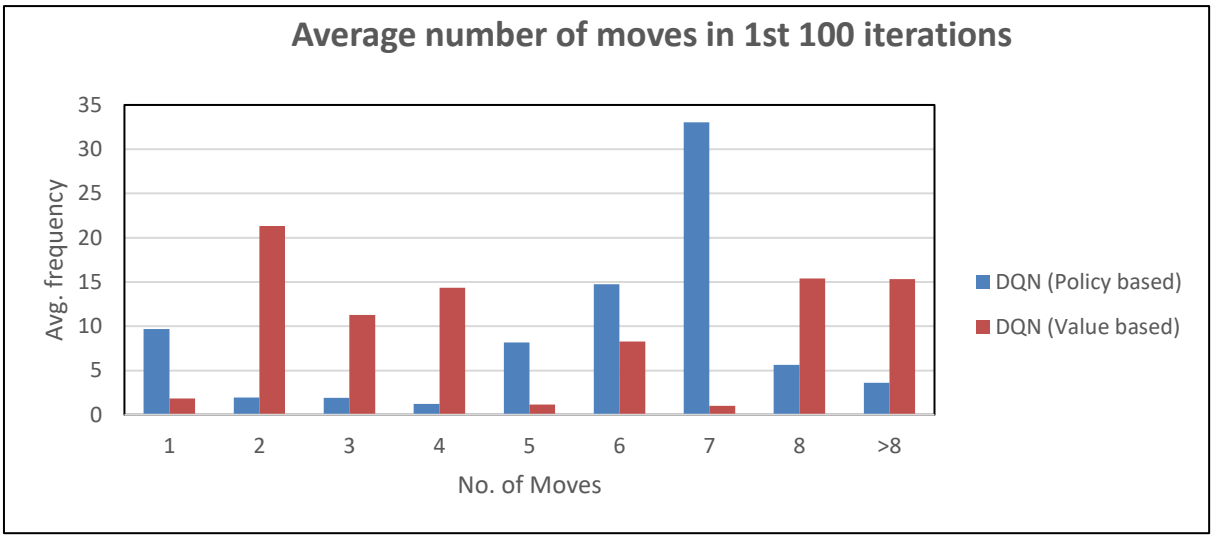


**Figure 24. Average Number of Moves in 100 Iterations**

Policy-based DQN is showing a bit more execution time than value-based DQN. This behavior is expected as the policy-based gradient is using probability distribution for actions as compared to value-based, which is forwarding action with maximum value. The average number of moves in the first 100 iterations also shows important results in comparing the performance of two DQN variants.

The graph in Figure 25 presents the collisions count as the time progresses. The collisions gradually decrease as the training time increases. This is because as the network starts to learn more about the environment, the collision count decreases. For both the value-based DQN as well as policy-based DQN the count decreases; however, the policy-based DQN performs slightly better in the long run with collision count reducing as the time passes.

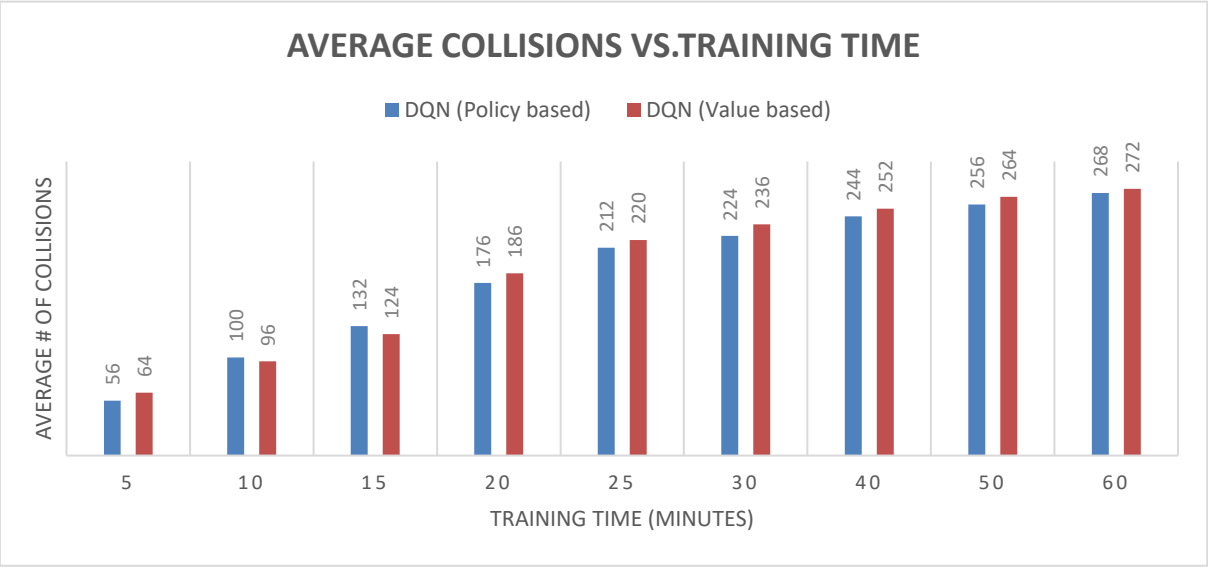


**Figure 25. Average Number of Collisions with respect to Time**

Initially, value-based DQN performs better in the first 10 mins with low collision count, but as we approach the hour mark, policy-based DQN takes the lead. The difference between the two algorithms is very small, though.

**Table 6. Statistical Quantities of Value-Based DQN and Policy-Based DQN in Different Environments**

| Environment \| Results | Success Rate | | Crash Rate | | Stray Rate | |
|---|---|---|---|---|---|---|
| | DQN (Policy-based) | DQN (Value-based) | DQN (Policy-based) | DQN (Value-based) | DQN (Policy-based) | DQN (Value-based) |
| **Blocks** | 98.02% | 97.25% | 1.50% | 1.70% | 0.48% | 1.05% |
| **Landscape** | 96.45% | 96.17% | 2.00% | 2.12% | 1.55% | 1.71% |
| **Neighborhood** | 97.25% | 96.80% | 1.50% | 1.50% | 1.25% | 1.70% |
| **Average** | **97.24%** | **96.74%** | **1.66%** | **1.77%** | **1.09%** | **1.48%** |

Table 12 summarizes the performance of both value-based DQN as well as proposed policy-based DQN. We define the success rate as the percentage of iterations in which the UAV was able to successfully navigate from source to destination using a fully trained model.

The crash rate is defined as the percentage of iterations the drone had collusion and stray rate as the percentage of epochs when the drone was unable to reach a destination or have a collision. We compare these three rates in three different simulation environments.

In all three environments, policy-based DQN outperforms the value-based DQN by a small margin. The performance on the success rate for both the networks is very good. The highest success rate is in *Blocks* due to its comparatively small size as compared to *Landscape* or *Neighborhood*. The average success rate for all three environments for policy-based DQN is 97.24 % while 96.74 % for value-based DQN.

**Table 7. Effect of Discount Factor**

| Discount Factor \| Results | Success Rate | | Crash Rate | | Stray Rate | |
|---|---|---|---|---|---|---|
| | DQN (Policy-based) | DQN (Value-based) | DQN (Policy-based) | DQN (Value-based) | DQN (Policy-based) | DQN (Value-based) |
| **0.99** | 97.24% | 96.74% | 1.66% | 1.77% | 1.09% | 1.48% |
| **0.97** | 96.31% | 96.10% | 1.68% | 1.78% | 2% | 2.12% |
| **0.95** | 94.15% | 93.84% | 1.72% | 1.79% | 4.13% | 4.37% |
| **0.93** | 93.07% | 92.96% | 1.83% | 1.85% | 5.10% | 5.19% |

The crash rate is also very low, varying between 1.5 to 2.0 percent for all three environments for both the algorithms. Similarly, the stray rate is also relatively small, ranging from 0.48 to 1.48. Table 13 summarizes the output based on different discount factors used. The discount factor is used to determine the importance of future reward as compared to the current reward. Higher discount factor means high importance to future reward and vice versa. Above table summarizes an important pattern in this regard. As we decrease the discount factor, the success rate starts to decrease (by 4.17 %), and the stray rate starts to increase (by 3.71%).

Interestingly the failure rate increases but very minutely (by .07%). This indicates that by increasing the discount factor, we can look ahead in the future and achieve better results at high values as compared to lower ones. Another important point is that although by decreasing the discount factors, the success rate also decreases but policy-based DQN still performs better than value-based DQN. Table 14 summarizes the success rate variation based on different learning rates over 1000 iterations.

**Table 8. Effect of Learning Rate**

| Learning Rate \| Results | Success Rate | | Crash Rate | | Stray Rate | |
|---|---|---|---|---|---|---|
| | DQN (Policy-based) | DQN (Value-based) | DQN (Policy-based) | DQN (Value-based) | DQN (Policy-based) | DQN (Value-based) |
| **0.1** | 75.08 | 74.54 | 3.74 | 3.90 | 21.18 | 21.56 |
| **0.01** | 92.15% | 91.84% | 2.32% | 2.39% | 5.46% | 5.77% |
| **0.001** | 97.24% | 96.74% | 1.66% | 1.77% | 1.09% | 1.48% |
| **0.005** | 91.07% | 90.96% | 2.83% | 2.85% | 6.10% | 6.19% |
| **0.0001** | 84.19% | 82.55% | 6.57% | 6.84% | 9.24% | 10.61% |

The results show that the learning rate for 0.001 gives the best performance with 97.24% success rate. If we decrease the learning rate, then the network remains undertrained. For values greater than 0.001, learning rate shows oscillations and fails to converge to an optimal state.

### 4.4.2 Qualitative Evaluation

In this section, we will evaluate our approach in comparisons with state-of-the-art work based on a few important parameters. Table 15 summarizes the performance of our approach in comparison with existing works.

#### I. Realistic Environment

In any navigation, the most important part is a realistic and complete environmental model. It's a baseline for interaction as a strong environmental model would result in strong results. In [11, 12], complete environments are used. In [12], a 2D environment is used, which in itself reduces the complexity model. It also removes a lot of complication from the problem statement. In [11], the 3D model is used, but a very simplified model is assumed. All the drag forces that act

on the drone and make navigation a difficult task are ignored. In [14, 15, 32], completely realistic environment is utilized. In our work, we are also utilizing a completely realistic environment taking into account all the factors that make the environment more stochastic.

### II. Model Free RL

Model free RL is not the golden standard for navigation tasks. Its performance is best among all the available methods. In [14], model based RL is utilized. In [15], simple DNNs are deployed. In [11, 12, 32], Deep model free RL is used.

### III. Use of Commodity Hardware

It's very important that what kind of hardware is utilized for the navigation as it has a direct impact on its adaptability if someone is getting very good results but is utilizing very specific hardware, then its less useful as compared to one which has relatively less impressive results but is using commodity hardware. [11, 12, 14] are using commodity hardware, so their work is extensible. In [15], very specific Nvidia TX1, AuVedia 120, and three cameras are utilized. In [32], an extra nine range sensors were attached to their UAV. In our work, we are using off the shelf drone with all the standard sensors. This makes our work more relevant for implementation by industry.

### IV. Onboard Processing

Onboard processing is an important benchmark in drone navigation. Nowadays, it's necessary that the drone should have all the processing onboard to guarantee complete autonomy.

In [14], an extra ROS server is deployed. In [11, 12, 32], only using onboard processing power is exploited. In [15], extra chips beside the traditional hardware are utilized. TX-1 and Auvedia-120 chips are being used. In our work, we are also not utilizing any off-board processing.

**Table 9. Performance Evaluation Summary**

| | Evaluation Parameters | | | | | | |
|---|---|---|---|---|---|---|---|
| **Related Work** | **Realistic Environment** | **Model Free Reinforcement Learning** | **Commodity Hardware** | **Only On Board Processing** | **Camera Utilization** | **Learning On the Fly** | **Navigation for long time spans (> 5 Mins)** |
| Low Level control of quadcopter with Deep model based RL [14] | ✓ | X | ✓ | X | ✓ | ✓ | X |
| Autonomous UAV navigation using RL [12] | X | ✓ | ✓ | ✓ | X | ✓ | X |
| Control of quadcopter with RL [11] | X | ✓ | ✓ | ✓ | X | ✓ | X |
| Towards low flying autonomous MAV Trail navigation [15] | ✓ | X | X | X | ✓ | X | ✓ |
| Autonomous navigation in UAV in Large scale environments [16] | ✓ | ✓ | X | ✓ | X | ✓ | ✓ |
| **Our Approach** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

## V. Camera Utilization

The camera is an essential part of drones nowadays. Its effective utilization is an important factor for any navigation system. In [14], they utilize standard drone camera as part of the proposed research. In [15], drone cameras are utilized, but instead of using the only available camera, 12 extra cameras are utilized that were installed separately. In [11, 12], no extra drone camera is deployed in proposed works. In [32], no drone camera is utilized but to detect the environment; nine range sensors are used as a replacement. In our work, we have utilized the commodity hardware that usually comes installed with most drones.

### VI. Learning on the Fly

This refers to the ability to learn from its mistakes as its flies instead of training model explicitly. In [11, 12, 14, 32], deep RL is utilized; therefore, the ability to have a model that learns on the fly exists. In [15], DNN is used; therefore, they lack this ability. We are also utilizing learning on the fly ability as we are also using model free RL.

### VII. Navigation for Long Durations

This parameter defines whether the authors were able to fly drone autonomously for more than 5 mins. In [15], they were able to achieve 6 seconds flight from 3 minutes of data. If they tweaked with the training data, their results deteriorated. In [12], results were presented for the 2D environment. Therefore, the work was only good enough for the testbed. For an outdoor, real-world 3D environment where drag forces effect, the model couldn't perform well. In [11], drone stabilization was achieved but not the navigation. Their work shows that the drone can hover for more than 5 minutes but cannot navigate through the environment. Another issue in this work [11] was a simplified model assumption which makes even hovering unstable in a real-world environment.

In the above section, we have summarized the critical factors that are important in drone navigation scenario. We have concluded from the above considerations that most of the works only cover a few of the above considerations. This makes our work stand out from rest as no one has covered all these factors in their research (till date) along with our state-of-the-art results to the best of our knowledge. The work that was closest to our implementation is presented in [32], but it also misses a few key aspects (e.g., customized hardware, extra sensors, etc.). We were also

able to achieve state-of-the-art success rate in our work. In our research work, we have taken into account all these factors and have completed autonomous navigation.

# Chapter 5

# Conclusion

Autonomous navigation in unknown terrain is an exponentially hard task due to unavailability of pre-constructed maps or path planning. In this thesis, a deep RL based framework is developed for UAVs navigation in unknown large-scale complex environments.

Specifically, first, an efficient policy-based DQN is designed, which comprises of convolutional and fully connected layers to extract important features from images taken through depth vision camera. These features are then combined with inputs taken directly from other sensors, e.g., IMU or magnetometer, etc. and are passed to a fully connected layer. This is followed by a policy-based Q-learning approach to apprehend UAV navigation.

We implement two different versions of this algorithm, i.e. policy-based DQN and value-based DQN. We can achieve 97.24% success rate for policy-based DQN and 96.74 % success rate for value-based DQN. Our results show that both these algorithms perform good but policy-based DQN outperforms value-based DQN by a small margin. We also achieve failure rates to around 1.6% for policy-based DQN and 1.7% for value-based DQN. As far as our knowledge, these are the best results presented to date. We are also able to autonomously navigate from start to finish line using only standard sensors. This factor combined with high accuracy in results is important in enabling this work to be adaptable to any off-the-shelf drone available in the market.

In regard to the contributions of this work, we were able to achieve a completely autonomous drone flight in a 3D environment through partially observable Markov decision

process or POMDP without having any prior knowledge of the environment and feature engineering. We were able to achieve 97.24% success rate and able to design a model that can run on any off-the-shelf drone available. Besides, focusing on to cover the shortcomings of the previous works, many aspects of the dynamic environment in which drone flies are also being considered.

Our research has proved that DRL is an effective tool to navigate a UAV from one location to another in a 3D environment. It will be useful for efficient resource management and dynamic tracking. Nonetheless, there is still room for future research. For instance, one next step for this research would be to experiment with this model in a real environment and enabling drone to fly.

# Bibliography

[1] https://www.droneii.com/drone-investment-trends-update.

[2] https://www.gpsworld.com/uav-report-growth-trends-opportunities-for-2019/

[3] J. Hwangbo, C. Gehring, D. Bellicoso, P. Fankhauser, R. Siegwart, and M. Hutter, "Direct state-to-action mapping for high dof robots using elm," in Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. IEEE, 2015, pp. 2842–2847.

[4] W.D. Smart and L.P. Kaelbling. Reinforcement learning for robot control. Mobile Robots XVI (Proc. SPIE 4573), 2001.

[5] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. Arxiv preprint cs/9605103, 1996

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al., "Mastering the game of go with deep neural networks and tree search," Nature, vol. 529, no. 7587, pp. 484–489, 2016.

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

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” arXiv preprint arXiv:1509.02971, 2015.

[9] https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle

[10] Su Yeon Choi & Dowan Cha (2019): Unmanned aerial vehicles using machine learning for autonomous flight; state-of-the-art, Advanced Robotics, DOI: 10.1080/01691864.2019.1586760

[11] Hwangbo, Jemin, et al. "Control of a quadrotor with reinforcement learning." IEEE Robotics and Automation Letters 2.4 (2017): 2096-2103.

[12] Pham, Huy X., et al. "Autonomous uav navigation using reinforcement learning." arXiv preprint arXiv:1801.05086

[13] Koch, William, et al. "Reinforcement learning for UAV attitude control." ACM Transactions on Cyber-Physical Systems 3.2 (2019): 22

[14] Lambert, Nathan O., et al. "Low Level Control of a Quadrotor with Deep Model-Based Reinforcement learning." arXiv preprint arXiv:1901.03737 (2019).

[15] Smolyanskiy, et al. "Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness." (IROS). IEEE 2017.

[16] ReddingJ, Geramifard A, UndurtiA, et al. An intelligent cooperative control architecture. In American Control Conference (ACC), 2010; IEEE; 2010. p.57–62.

[17] Redding J, Geramifard A, How JP. Actor-critic policy learning in cooperative planning. In AAAI Spring Symposium: Embedded Reasoning,2010.

[18] Geramifard A, Redding J, Roy N, et al. UAV cooperative control with stochastic risk models. In American Control Conference (ACC),2011; IEEE;2011. p.3393–3398.

[19] Geramifard A, Redding J, How JP. Intelligent cooperative control architecture: a framework for performance improvement using safe learning. J Intel Robot Syst. 2013;72(1):83–103.

[20] Zhang B, Liu W, Mao Z, et al. Cooperative and geometric learning algorithm (CGLA) for path planning of UAVs with limited information. Automatica. 2014;50(3):809–820.

[21] Xu H, Carrillo LRG. Fast reinforcement learning based distributed optimal flocking control and network codesign for uncertain networked multi-UAV system. In Unmanned Systems Technology XIX; International Society for Optics and Photonics; 2017. vol. 10195, p. 1019511.

[22] Vandapel N, Kuffner J, Amidi O. Planning 3-d path networks in unstructured environments. In (Proceedings of) the 2005 IEEE International Conference on Robotics and Automation (ICRA); IEEE;2005. p.4624–4629.

[23] Bachrach A, He R, Roy N. Autonomous flight in unknown indoor environments. Int J Micro Air Vehicle. 2009;1(4):217–228.

[24] Bry A, Bachrach A, Roy N. State estimation for aggressive flight in GPS-denied environments using onboard sensing. In 2012 IEEE International Conference on Robotics and Automation (ICRA); IEEE;2012, p.1–8.

[25] Bachrach A, Prentice S, He R, et al. Estimation, planning, and mapping for autonomous flight using an RGBD camera in GPS-denied environments. Int J Rob Res. 2012;31(11):1320–1343.

[26] Achtelik, M., Achtelik, M., Weiss, S., & Siegwart, R. (2011, May). Onboard IMU and monocular vision based control for MAVs in unknown in-and outdoor environments. In 2011 IEEE International Conference on Robotics and Automation (pp. 3056-3063). IEEE.

[27] Wendel, A., Maurer, M., Graber G., et al. Dense reconstruction on-the-fly. In 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); IEEE; 2012.p. 1450–1457.

[28] Fraundorfer, F., Heng, L., Honegger, D., et al. Vision-based autonomous mapping and exploration using a quadrotormav. In2012IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); IEEE, 2012. p. 4557–4564.

[29] Luo, Q., Duan, H. Distributed uav flocking control based on homing pigeon hierarchical strategies. Aerosp Sci Technol. 2017;70:257–264.

[30] Lee D, Kim S, Suk J. Formation flight of unmanned aerial vehicles using track guidance. Aerosp Sci Technol. 2018;76:412–420.

[31] Ambroziak, L., Gosiewski, Z. Two stage switching control for autonomous formation flight of unmanned aerial vehicles. AerospSciTechnol.2015;46:221–226.

[32] Wang, Chao, et al. "Autonomous Navigation of UAVs in Large-scale Complex Environments: A Deep Reinforcement Learning Approach." IEEE Transactions on Vehicular Technology

[33] https://www.unrealengine.com/marketplace/en-US/store