University of Southern Queensland

Faculty of Health, Engineering and Sciences

# Comparison of SLAM Algorithms and Neural Networks for Autonomous Navigation in Simulated Environments

A dissertation submitted by

Brayden McConnell

in fulfilment of the requirements of

ENG4111 and ENG4112 Research Project

towards the degree of

Bachelors of Engineering (Honours) (Mechatronics)

Submitted October, 2020

# Abstract

When looking at the research and industrial landscape, the tendency to favour SLAM based algorithms is ever present and for a good reason. SLAM algorithms are the most mature algorithms that we have available today. However, competitors to this are starting to arise to fill in the gaps that SLAM has in its capabilities at the moment. One of these competitors is the utilisation of Deep Neural Networks or DNN's - a term loosely applied to Deep Learning algorithms.

The proposed study focussed on establishing the characteristic differences between SLAM based algorithms and CNN based algorithms within simulated environments.

The Deep Neural Network being tested was developed by the author of the project and trained using data collected within the chosen simulation's engine. The simulation engine utilised was Unreal Engine 4 (version 4.24) using the latest version of the AirSim plugin (as of the March 2020 update v1.3.1). The SLAM algorithms being used for the comparison were MATLAB variants of ORB-SLAM and CEKF-SLAM.

The findings of the project are largely inconclusive, with the project conclusion defining that additional work needs to be complete in order to provide conclusive evidence for the project. However, the supporting findings of the project validate its position as having useful research findings. A clear consensus has been established on the methods required to operate within these environments and understanding the limitations that apply under these software conditions. This is information that is currently poorly defined in the Literature and as such this project serves as a strong base for future projects to build on.

Anecdotally, it has been found that the utilisation of CNNs is far more accessible in simulated environments such as AirSim. However, it is a direct result of immaturity for the software utilised. Additionally, the strong Literature Review provides evidence in the direction of CNN utilisation over SLAM based algorithms however these findings were not validated.

The project is aimed to be conducted across the formal 2020 academic year spread across the two courses ENG4111 and ENG4112 as part of the dissertation process for a graduating BENGH Engineer.

University of Southern Queensland

Faculty of Health, Engineering and Sciences

# ENG4111 & ENG4112 Research Project

## Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering and Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering and Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitles “Research Project” is to contribute to the overall education within the student’s chosen degree program. This document, the associated hardware, software, drawings, and any other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

# Certification

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Brayden McConnell

Student Number: [redacted]

# Acknowledgments

To my supervisor, Dr. Tobias Low, thank you for allowing me to do this project and for your feedback throughout the year. This has been a great learning experience for me.

To my Father, thank you. I am the man I am today because of everything you have taught me. I cannot put into words my respect and admiration for you.

To my Mother, thank you for caring for me and always showing me how to love and how to be loved. You have always put us first and I am thankful for that.

To Mitchell, Reece and Amelia, thanks for all of your support and being amazing siblings.

To Brie, thank you for all of your love and support throughout this project.

BRAYDEN MCCONNELL

# Contents

# Nomenclature

| | |
|---|---|
| CNN | Convolutional Neural Network |
| CPU | Central Processing Unit |
| DNN | Deep Neural Network |
| GPU | Graphics Processing Unit |
| PiE | Player in Editor |
| RAM | Random Access Memory |
| RPM | Revolutions Per Minute |
| SLAM | Simultaneous Localisation and Mapping |

# Chapter 1 - Introduction

## 1.1 Preamble

Complete automation has been a major point of interest for industry researchers and influencers as of current. It is beginning to become a requirement for underground mining globally and is a topic of interest and debate currently within the production automotive industry. With Tesla pushing the industry to more heavily focus and invest in automation there is beginning to be a dispute with respect to the methods that should be utilised to achieve the end goal of complete automation.

For mobile robots, the general standard for building automation is based on the utilisation of mapping software that can produce an understanding of the environment. Generally, this is lumped together with a localisation method in order for the mobile robot to make sense of its position within the map it has produced. This is known as a SLAM algorithm - Simultaneous Localisation and Mapping.

The utilisation of Neural Networks can be considered the bread and butter of modern statistical based inference algorithms. Neural Networks is a broad defining topic and often describes a wide genre of algorithms - but hold commonalities in their basic structure. It is within this field with which this project will reside. Deep Learning is the field of Machine Learning and Neural Network development that is essentially ‘large’ with respect to today's compute capability. Often, Convolutional Neural Networks (CNN) and Generative Adversarial Networks (GAN) are lumped into this broad term.

This paper will look at the utilisation of a Deep Learning algorithm along with the industry preferred SLAM algorithms to investigate the current state of the industry within a simulated environment and utilise evidence from simulations to determine whether or not the evidence found in this paper supports the literature or establishes discourse in the field..

## 1.2 Project Description

### 1.2.1 Aim

This project aims to investigate the use of Deep Neural Networks compared to Simultaneous Localisation and Mapping based algorithms for autonomous control of drones within simulated environments and to validate findings within the simulated environment. This focuses on the aspects and differences of

developing and deploying a Deep Neural Network and a SLAM algorithm for use in a simulated environment and aims to establish a clear understanding of the variances between each algorithm type.

### 1.2.2 Scope

The scope of the project intends to cover the research, development and training of an appropriate Deep Neural Network and the comparison of it to well established SLAM based algorithms.

The performance comparison will be interpreted through two main avenues; the first being the resource utilisation and compute performance required for the system to operate sufficiently, the second being the performance of the system with respect to known SLAM research difficulties. The difficulties have been outlined in Section 2.1 however the difficulties can be briefly outlined as; dynamic problems, illumination variance and the 'kidnap' problem.

The project will attempt to limit its focus, especially with respect to the development of the Deep Neural Network, to that outlined in the above scope along with aiming to establish considerable understanding on how to operate these algorithms in a simulated environment. If the scope was only defined as a Neural Network approach, then the extent of research into the design of varying neural structures would expand the project beyond its intended and beneficial decree. To define it simply, the project is limited to the investigation of and comparison of a Deep Neural Network based autonomy algorithm to that of well researched and well-defined Simultaneous Localisation and Mapping based autonomy algorithms as provided by the research organisation OpenSLAM. The two SLAM algorithms to be adapted are the ORB-SLAM and CEKF-SLAM algorithms. The application of the autonomy algorithms will purely be limited to that of a monocular visual input quadcopter as defined in the AirSim documentation and the sensors provided within the package.

### 1.2.3 Objectives

The completion of this project involves the following objectives:

1. Design and develop the appropriate simulated environments
2. Convert and implement the known comparison SLAM algorithms from OpenSLAM
3. Design, train and implement a DNN capable of drone flight
4. Design and conduct experiments
5. Determine suitability of DNN
6. Compare performance metrics of SLAM and DNN

7. Present a conclusion and future work necessary

### 1.2.4 Motivation and Problem Statement

There are a variety of influences that have motivated the work completed in this project, the simplest being a personal desire to learn about and clarify disagreements within the industry of automation. Throughout my years of study leading up to this project, I have been exposed to the works of individuals such as Lex Friedman, Ilya Sutskever and Sertac Karaman in which my beliefs of the capabilities of neural networks and their applications expanding beyond categorical identifications and labelling have brewed.

My main goal with the learnings, findings and my own personal gain throughout this project is to eventually begin to make general and specific autonomy better understood. This can be achieved by attempting to establish a consensus on the characteristics of each algorithm type and the general benefits or disadvantages that are inherit to them within a simulations environment. Consider mining vehicles, which must be able to not only determine paths, but understand friction differentials with the material with which they are driving upon - it is extremely common for underground paths to be either extremely dusty or extremely slippery due to dust suppression or burst water mains. The ability to infer differences in grip along with driving characteristics which best suit this can be learned and inferred within a complex Deep Learning model given it has the ability to abstract to this degree.

Now, the reasoning for applying this to drones is the fact that there are many more degrees of freedom associated with the flight control of a drone compared to a land-based vehicle and that the research tool utilised for the project specifically focussed on the utilisation of quadrotor drones. From a mostly personal standpoint, I see this technology being heavily investigated within the research community, but its application is far behind the studies being conducted. Top performing companies are attempting to move into this area, but the current unreliability of neural networks is forcing the consumption of SLAM based algorithms - which is not exactly a detriment - as a means to resolve their current dilemma.

Developing a coherent understanding between well-established SLAM algorithms compared to Deep Neural Networks may begin to help the industry to better understand the benefits of each method and ways in which both may be beneficial when combined to resolve a task. Conducting these comparisons in a simulated environment is beneficial for a variety of reasons. Not only for the safety factors, but the fact that parallel and continuous experimentation and data gathering can be conducted on a scale that is impossible to match in the real world.

### 1.2.5 Hypothesised Outcome

I hypothesise that the solution based on the deep neural network architecture will present evidence in favour of its utilisation as opposed to simultaneous localisation and mapping algorithms. This is due to the end to end capability of deep learning algorithms and the ability to approximate multiple actions at once in a single network. This would theoretically be more computationally efficient after being trained than a SLAM based algorithm and provides the basis for more complex control given the network is trained effectively. I do believe that if SLAM algorithms provide more significant evidence in favour of their utilisation compared to DNN algorithms that it will be due to ineffective implementation of the DNN and not necessarily due to the performance benefits of a SLAM based approach. However, I will not misrepresent the findings in any fashion and will attempt to utilise any results to bolster improvement approaches for both algorithms.

## 1.3. Background Information

The landscape of drone automation - and the majority of vehicle-based automation in general - has long been built on the pillars of SLAM based automation. This is for a good reason, SLAM has long since established itself as a worthy algorithm to mould the industry around. However, as compute power becomes more viable for mobile robots, the application of AI is beginning to hold prominence. Specifically, Deep Learning is beginning to become more prominent for navigational and autonomous purposes.

The project in some degree is beginning to set the scene for a SLAM and DNN showdown to find the optimal solution - but considering that both algorithms are largely unfinished (being that there is no “one” solution for all scenarios) it will be difficult to draw a clear conclusion based on this project alone. It is ultimately likely that the ultimate solution will not be entirely binary, but a blend that accentuates the superior characteristics of each method.

The push for training within simulated environments is becoming stronger for a multitude of reasons. Regulations in the real world are beginning to limit the capability to test systems in that it may realistically no longer be viable to conduct research in the real world at the same scale. Looking at the regulations implemented throughout the early stages of 2020 due to COVID-19 it is understandable that simulated test and training environments are likely to become more preferable in the future. Ultimately the push for simulated environments has an added benefit. Training in a simulated environment allows for

the training of networks to be accelerated and begins to open the capabilities up for genetic networks, parallel operation and simultaneous training.

The following section is focussed on defining the relevant background information for the project to begin establishing a consensus on each algorithms capability and method of operation.

### 1.3.1 Drone Dynamics & Understanding Drone Flight

In order to be able to properly approach the proposed project, a basic understanding of the dynamics of the system involved is required. An understanding of drone kinematics in the real world can be transferred to the simulated environment - but the accuracy is not identical. Looking at the white paper for AirSim, the necessary dynamics are outlined.

To simply explain how a drone is able to fly, it is via the displacement of air such that the downward force of air results in a climbing force being produced by the rotors. Knowing that the rotors (or propellers) are directly responsible for the force production then it can be claimed that by altering the force produced from the rotors a change in the flight path will occur. Thus, the velocity and directions they spin directly correlate to the flight path and the rotations necessary to develop them. The displacement of air follows the rotational path of the blade and utilising this we can also begin to introduce rotations to the system.

The present forces for the simulated quad rotor drone can be best outlined as directly noted in the white paper for AirSim. The model for it can be seen as follows in Figure 1:

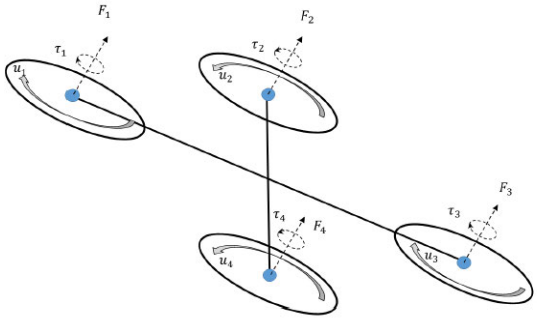



Figure 1. Force Model for Quadrotor Drone in AirSim (Shah et. al. 2017, pg. 5)

As can be seen, there are three apparent general forces in the model. The inputs from the ‘motors’ are input as a unitless scaler control input {u1, u2, u3, u4} and can be considered the acceleration component of the motors. The model for the vehicle presented here is based on how it is represented in the AirSim

simulation engine. The input Ui develops a force normal to the rotational direction - this produces a calculable torque for each input. Therefore, the forces can be outlined as follows:

$$u_i = \{u_1, u_2, u_3, u_4\}$$
$$F_i = \{F_1, F_2, F_3, F_4\}$$
$$\tau_i = \{\tau_1, \tau_2, \tau_3, \tau_4\}$$

The torque ($\tau_i$) and force values ($F_i$) can be defined as follows:

$$F_i = C_T \rho \omega_{max}^2 D^4 u_i$$
$$\tau_i = \frac{1}{2\pi} C_{pow} \rho \omega_{max}^2 D^5 u_i$$

(Shah et. al. 2017, pg. 5)

Where $C_T$ and $C_{pow}$ are the coefficients of thrust and power and are based on the propeller, $\rho$ is the air density of the environment, D is the propeller diameter and $\omega_{max}$ is the maximum angular velocity of the rotor RPM (Shah et. al. 2017). The $u_i$ input values can be considered as an acceleration input to the model in order to develop the necessary torque at the rotors. However, the simulation model does not exactly see the input $u_i$ as an acceleration. It doesn't directly accrue the acceleration value of the vehicle, but it is the excitation percentage that indicates the ability to accelerate. Essentially, treat it as a throttle input value.

When looking at the sensors that are generally used to facilitate stable flight the first that comes to mind is the IMU - Inertial Measurement Unit. This is a unit that records data from two sources - a gyroscope and an accelerometer. This provides feedback to the unit in the form of measured change in its movement in the form of rotational and linear accelerations. For the six degrees of freedom within which the unit can move, the IMU provides real-time measured data - however this will generally need to be filtered.

For a quadrotor drone moving in a realistic environment it will not be exposed to an entirely homogeneous airspace as it is in the simulation environment. In a realistic environment the airspace will change throughout its movement path - meaning minor alterations in pressure, temperature, wind speed etc. such that minor control changes will need to be made in order to maintain accuracy. These conditions cannot entirely be accounted for within the simulations environment as it was not a part of the AirSim plugin at the time of conducting the project. Therefore, any findings for this project will need to be considered applicable only to simulations environments and transfer learning or accounting for these minor alterations will be necessary for a production drone. T

There are a surprising number of methods to fly a quadrotor drone. One would assume that you require all of the rotors to be active in order to appropriately fly the drone, but this does not appear to be the case. Under certain circumstances, the drone can fly with only 2 active rotors under the condition that they can provide enough lift to maintain altitude. These must be directly opposite one another, however under these conditions the drone loses its capability to directly control all of its degrees of freedom. Under these conditions the drone loses its ability to control rotation about the z-axis - it will spin seemingly uncontrollably however the drone can maintain control over all other degrees of freedom (TED | Raffaello D'Andrea, 2013). Developing the capabilities to do this into the drone model for this project is not necessary however understanding these capabilities exist may assist in future training of the models if it is considered necessary to implement additional modes of recovery for failed motors.

One of the major research benefits of utilising the AirSim plugin is that the majority of the kinematic and control models for the quadcopter are already embedded in the software. It provides an API for the user to control the drone's; pitch, yaw, roll, and x, y, z linear accelerations (Shah et. al. 2017). This simplifies any additional structure that would need to be built into the system - this structure being a model to balance acceleration distribution between the four motors when inputting a desired action such as a forward request. For a further explanation of the API control methods used for this project please refer to Chapter 5.

### 1.3.2 SLAM Algorithm Design & Implementation

A term that will arise quite often when researching the underpinning mechanics of mobile robots is that of SLAM. In essence, SLAM (Simultaneous Localisation and Mapping) is an algorithm for autonomous mobile robots that allows them to construct a map based on the apparent environment and then subsequently localise itself within the map. A more formal definition can be defined as,"…the process of concurrently building a feature-based map of the environment and using this map to obtain estimates on the location of the [object]." (Williams & Dissanayake & Durrant-Whyte 2002). This allows a mobile robot to have an understanding of its position in a real-world environment and to understand the environment around it as well - allowing it to be able to make decisions depending on the navigational software it is using. To break a generic SLAM algorithm into its most basic components, it consists of; landmark extraction, data association, state estimation, state update and landmark update (Riisgaard & Blas 2004, pg. 6).

SLAM algorithms found their rough modern-day origins with the publication of the paper, 'Estimating Uncertain Spatial Relationships in Robotics' by Smith R. et. al. which investigated the use of spatial

information for a robot to produce a stochastic map that contains relationships between the origination of the data (consider it the mobile robot) and objects within the map it has produced. From here, it progressed with the publication of 'Simultaneous Map Building and Localization for an Autonomous Mobile Robot' by Leonard & Durrant-Whyte which aimed to resolve the issue of initialisation for SLAM algorithms. The authors presented the problem as being, "to move precisely, a mobile robot must have an accurate environment map; however, to build an accurate map, the mobile robot's sensing locations must be known precisely." (Leonard & Durrant-Whyte 1991) to which they likened to the age-old question pondered about the chicken and the egg - which comes first?

A general outline of a SLAM algorithm in flowchart form can be seen below in Figure 2:

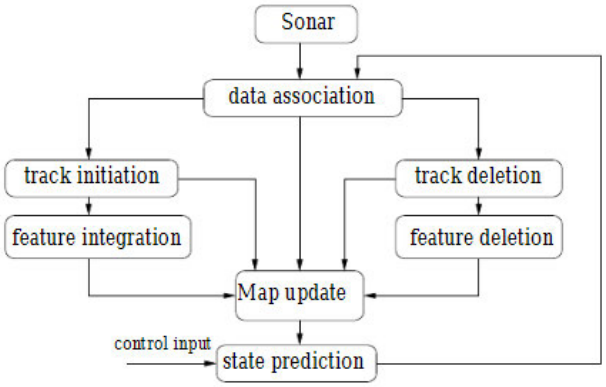



Figure 2. Generic SLAM Algorithm Modified from source (Zunino 2002, pp 39)

This is a flowchart utilised for a Sonar-based application shown in the paper titled, *'Simultaneous Localization and Mapping for Navigation in Realistic Environments'*, by Zunino, G. that utilises a Sonar input as the main source of data for the algorithm. The input to a SLAM algorithm can be a variety of sources however, including; RGB camera input, Ultrasonic sensors, Radar, thermal heat maps, etc. so long as it provides an ability for the algorithm to produce a map in some form of its local environment.

Looking at the flowchart in Figure 2. and having the understanding of the basic components of SLAM an appropriate explanation of a generic SLAM algorithm can be given. Using an input from an RGB camera as opposed to a Sonar input the algorithm can be generalised to give an understanding of how it is applied in this project.

The algorithm will produce a map (explained further in the section) and this map will be stored as the environment state at step *n*. In this environment state, if it is the first piece of incoming data then the algorithm will go through an initialisation process. This means that there is no prior data to create

associations with. However, let's assume that instead of looking at the environment at step *n* we are looking at the environment at step *n + 1* being that initialisation has already successfully occurred. When assessing step *n + 1* in relation to step *n*, data association occurs. This portion of the algorithm assesses the data for features in which it can produce an orientation and understanding of change from. For instance, if a prominent feature, such as a yellow strip, moves horizontally between steps and the incoming frame rate or time between keyframes is understood then inference about the change in the mobile robot's position and thus the perceived map can be made. Treat this initial portion, the data association, simply as an identification step. From here, the algorithm will split into two simultaneous paths. The track initiation portion will store prominent feature points in the map and attempt to find them within each incoming image. If a new prominent and easily distinguishable data point is found, then it will be integrated into what is known as the feature map. Looking down the right-hand side of the flowchart, there is the process of track deletion and feature deletion. They are quite self-explanatory in reference to the actions on the left-hand side of the feature map; however, the initial portion maintains an understanding of changes that have occurred to the incoming image. If there are prominent features that are no longer desirable, then the algorithm will periodically remove the features from the map. With an understanding of the change in input data, and an understanding generally of the movement that has occurred, the map will be updated to reflect the new state of the system. The algorithm will then utilise this current map along with the incoming control input (this is generally given by the navigational component of the mobile robot) to make a new state prediction. This state prediction can be better understood by treating it as the pose estimation of the mobile robot which encompasses the location and rotation data for the mobile robot. From here, the state from step *n+1* is fed into the system and the algorithm repeats the process.

The prior paragraph provides a rudimentary understanding of how a generic SLAM algorithm works and will help when looking at future implementations of the algorithm type. However, it is a relatively poor technical breakdown of the algorithm. It must be understood that for the majority of applications, a SLAM algorithm alone will not suffice for a mobile robot to conduct autonomous operations. It will need a navigational and path planning algorithm that interprets the SLAM output data. There are a variety of path planning techniques such as A*, D*, Dijkstra and RRT (Correll, N 2007). However, it is quite often for a navigational algorithm to utilise what is known as an occupancy grid or an occupancy map.

Mapping can be done due to optical flow and pixel movement velocities estimating depth data however it is significantly more difficult to conduct visual SLAM when utilising an individual input camera (monocular) compared to having two input cameras (stereo) or using RGB-D (depth and ToF data)

cameras (MathWorks n.a.). Under monocular SLAM there is generally a requirement to combine the image data with some form of accurate movement data (velocity, rotation and orientation). This can be given by an IMU sensor or by an external camera tracking the alterations in the movement of the mobile robot - but this isn't always a feasible option (University Freiburg n.a.).

Localisation is done utilising the incoming data along with the map projection. Localisation is conducted in varying fashions depending on the algorithm type being utilised. ORB-SLAM takes a current keyframe and creates a set of 3D map points based on features within the previous keyframe and the current pose of the mobile robot (MathWorks n.a.). It will then apply a process known as bundle adjustment to minimise reprojection errors by making adjustments to the 3D points and the camera pose. From here the loop closure is applied and the system returns to the mapping and tracking aspect of the algorithm.

However, the implementation of SLAM alone is not sufficient for autonomous control. There needs to be action taken on the data that a SLAM algorithm produces. This can be done through the use of an occupancy grid as mentioned prior. An occupancy grid is used to represent the environment or workspace of a mobile robot in a discrete grid (MathWorks n.a.). In the case of a monocular input, the occupancy grid is based on a probabilistic likelihood that that grid is navigationally satisfactory and as such suffices to move within. It is a free space estimate based on the images contents and creates a costmap for the movement path to be determined (MathWorks n.a.).

The example images provided by MathWork serve best to describe its operation. This appears as follows:

Figure 3. Estimate of Free Space Indicated by Green Overlay (MathWorks n.a.)

It can be seen that the pretrained algorithm in this case has determined portions of the road that are considered to have free space (in this instance, it is a road-based vehicle). This free space estimate is then overlayed with a green overlay for visual understanding for the user (MathWorks n.a.). From here, a confidence score on the free space estimate is determined. Taking the image shown in Figure 3. its confidence scores would appear as follows in Figure 4:

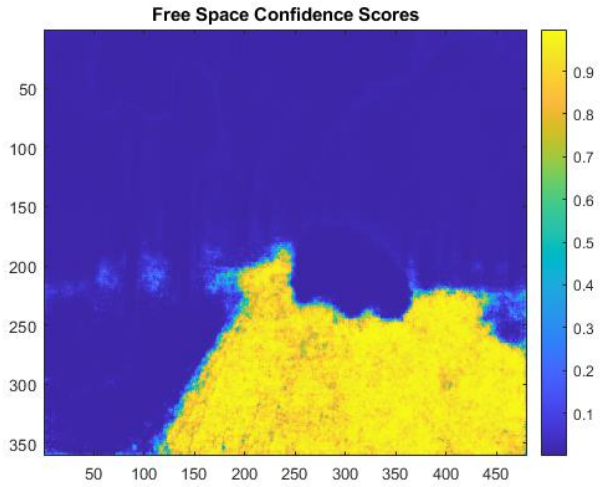
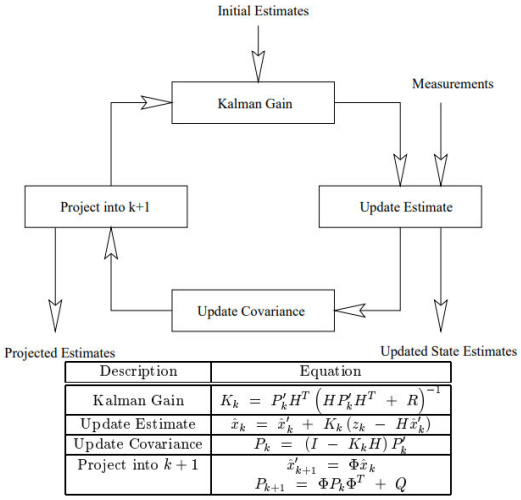



Figure 4. Free Space Confidence Score Plot (MathWorks n.a.)

From here, precise control algorithms can be applied to the new understanding of the environment that the mobile robot has developed. The control navigational execution can be done by a multitude of complex controllers, however the 'simplest' and often used case is a PID controller which adjusts speed and angle based on error of the centre pixels in an image such that it remains within its general path vicinity and moving in the direction of free space if that is its only known mission requirement.

Explaining a Kalman filter may help the reader to understand filters and SLAM algorithms from a mathematic viewpoint. The Kalman filter shown below in Figure 5 has clearly defined objectives. The system begins by taking the initial estimation and parsing it through to the Kalman Gain Matrix, from which its output is compared with incoming measurements and an update is applied to the current estimate. Following on, it will update the state estimation and update the covariance matrix – this is a measure of the scatter of data and its variance. Utilising the new covariance values, the system will project the update into step $k+1$ of the system and produce an updated projected estimate matrix as a result. Here, this is compared again to the Kalman Gain of the initial estimate in order to repeat the process. This very closely emulates a generic SLAM algorithm as it is applied over the top of a SLAM algorithm.




| Description | Equation |
|---|---|
| Kalman Gain | $K_k = P'_k H^T \left(H P'_k H^T + R\right)^{-1}$ |
| Update Estimate | $\hat{x}_k = \hat{x}'_k + K_k(z_k - H\hat{x}'_k)$ |
| Update Covariance | $P_k = (I - K_k H) P'_k$ |
| Project into $k+1$ | $\hat{x}'_{k+1} = \Phi \hat{x}_k$<br>$P_{k+1} = \Phi P_k \Phi^T + Q$ |

Figure 5. Generic Kalman Filter Flow Chart with Equations (Lacey n.a., pp 138)

. Taking a look at the equations attached, we can see there is an equation listed for; Kalman Gain, Update Estimate, Update Covariance and Project into *k+1*. The following equations and analogy are excerpts from the document 'Tutorial: The Kalman Filter' by Tony Lacey.

Assessing the use of the Kalman Gain equation we see the following:

$$K_k = P'_k H^T (H P'_k H^T + R)^{-1}$$

Where; $K_k$ is the Kalman Gain of the system in matrix form; $P'_k$ is the prior estimate of $P_k$, which is the error covariance matrix at time k with size *n*x*n*; $H^T$ is the transpose of the vector that represents a noiseless connection between the state vector and the measurement vector, this is defined and remains static; R is the covariance matrix of the measurement error and also remains static.

Assessing the Update Estimate equation, we see the following:

$$\hat{x}_k = \hat{x}'_k + K_k(z_k - H\hat{x}'_k)$$

Where; $x_k$ is the estimate matrix; $x'_k$ is the prior estimate matrix; $K_k$ is the Kalman Gain of the system; $z_k$ is the actual measurement of the $x_k$ matrix; together, $K_k(z_k - Hx'_k)$ is known as the measurement residual.

Assessing the Update Covariance equation, we see that:

$$P_k = (I - K_k H) P'_k$$

Where; $P_k$ is the covariance matrix; I is an identity matrix of size *n*x*n*; $K_k$ is the Kalman Gain matrix; H is the noiseless vector that represents the connection between the state and measurement vector as seen in; $P'_k$ is the previous covariance matrix values.

Finally, assessing the systems projection into k + 1 we have 2 equations present:

$$\hat{x}'_{k+1} = \Phi \hat{x}'_k$$

Where; $x'_{k+1}$ is the state projection estimate; $\Phi$ is the state transition matrix; $x_k$ is the current estimate matrix.

Where; $P_{k+1}$ is the projected error covariance matrix; $\Phi$ is the state transition matrix; $P_k$ is the current error covariance matrix; $\Phi^T$ is the transpose of the state transition matrix; Q is the covariance of the noise matrix associated with white noise.

The understandings developed here will assist in the utilisation and development of the SLAM algorithms for this project such that they can be used as a validation component in fulfilling the proposed project aim.

### 1.3.3 Artificial Neural Networks & Deep Learning

Artificial Neural Networks is a broad defining term. In essence, Artificial Neural Networks, or simply Neural Networks, are a computer emulated attempt at replicating the architecture that has made organic brains so efficient and effective at accomplishing a wide variety of tasks. But, what exactly is Deep Learning? In the book, 'Programming PyTorch for Deep Learning: Creating and Deploying Deep Learning Applications' by Ian Pointer lies a very succinct description of the term Deep Learning. In the Preface the author states, 'A way of defining it is to say that deep learning is a machine learning technique that uses multiple and numerous layers of nonlinear transforms to progressively extract features from raw input." (Ian Pointer 2019). That should be clear, right? Well, hardly. To describe Deep Learning in an informal manner in my own words; it is essentially the application of an algorithm, generally a neural network, in which an output state can be objectively learned and estimated as a result of either supervised

or unsupervised training. But how does relating the prior sentence to the claim that they are replications of the brain's function correlate?

Developing a complete understanding of the brain is not necessary to understand the basic underlying mechanics of neural networks. In the absolute simplest of forms, a neural network can be presented as a Perceptron. This is a 3-node network in which there is an input node(s), an activation function, and an output node (DeepAI n.a.). The input node(s) are generally a vector or an array of vector inputs. Following on from here is the activation function. This looks at a sum of inputs. Say for instance, that there are 3 input nodes going into one summing node. The activation function at this node will look at the incoming values from all three inputs, and if their sum value exceeds the value determined by the activation function, then the summing node will become active and fire a forward response. The input values are generally biased by a weight which controls the amount of influence that is given to each input node. The simplest activation function is considered to be the binary step function. It operates on the basis of if the input condition is met, then it will activate and output a value of 1. If the condition is not met, then the output condition is 0. For the output node, it will also maintain its own activation function. If the incoming values exceed the output nodes activation function, then it will activate. For a standard condition the output node is generally tied to a class which indicates a feature within the data - take for example an image classifier, the output class may be a cat that has been detected in the image data, or it may simply be an input into another hidden layer.

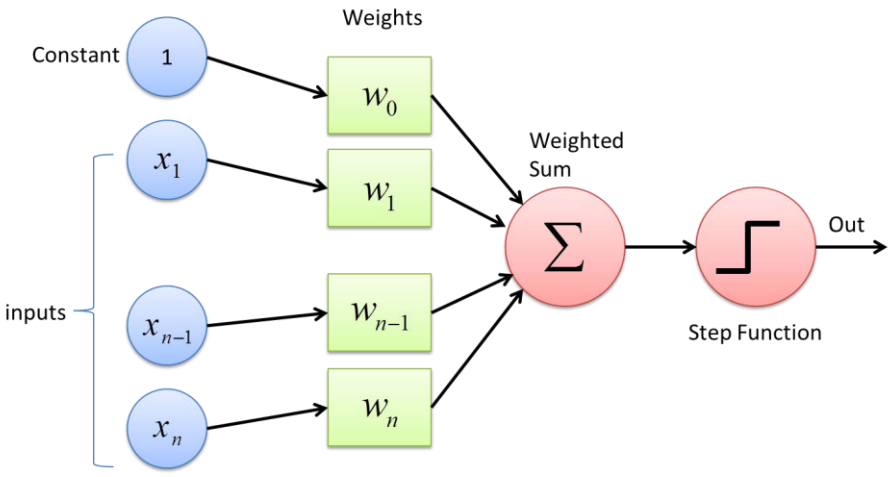



Figure 6. Example of a Multi Input Single Output Perceptron with a Binary Output Activation Function (DeepAI n.a.)

Artificial Neural Networks are seemingly cutting-edge technology, but in reality their modern day adoption only appears to revolve around the increase in compute power and compute efficiency as it has

been strongly dismissed in the past. In 1969, M. Minsky and S. Papert published a text on the limitations of Artificial Neural Networks titled Perceptrons which is arguably recognised to have sowed doubt into the research community about investigating Artificial Neural Networks further and subsequently a large portion of funding was cut for the sector (Indiana University South Bend n.a.). Its identifications and explanations surrounding the limitations of the simplistic functions being used at the time was the main objective taken from the text. Stepping back even further in time, the first inclinations towards designing Artificial Neural Networks began over 70 years ago, with the first neuron model being developed in 1943 by W. McCollough and W. Pitts (Indiana University South Bend n.a.). However, the most notable discovery which is often viewed as the starting point for modern neural architectures is that of the Perceptron. The Perceptron was developed by Frank Rosenblatt in 1958 and is actually the simple 3 node neural network described in the prior paragraph. When asked about the project, Rosenblatt is credited with saying, "... we are about to witness the birth of such a machine – a machine capable of perceiving, recognizing and identifying its surroundings without any human training or control." and he held strong remarks for its capabilities, making the claim that the perceptron would be, "the first machine which is capable of having an original idea." (Lefkowitz, 2019). Professor Rosenblatt's enthusiasm surrounding the capabilities of Artificial Neural Networks may have been dismissed as irrational at the time, but their capabilities are quickly beginning to be realised and have become the source of fear for many. With claims from individuals such as Elon Musk that, "I think the danger of AI is much greater than the danger of nuclear warheads by a lot…" (Clifford, 2018) during a questions and answers segment at his presentation at SXSW in 2018 showing this distrust for the technology. There are many opposing philosophies that consider the danger of AI to be negligible due to their current immature status. Regardless, investigating the future of AI and ANN hardly falls in line with the project scope. Neither is making judgements on its suitability for the future stability of the society.

To get a better understanding of the operation of a Perceptron, let's take a look at a neuron in the brain and its features as follows in Figure 7:

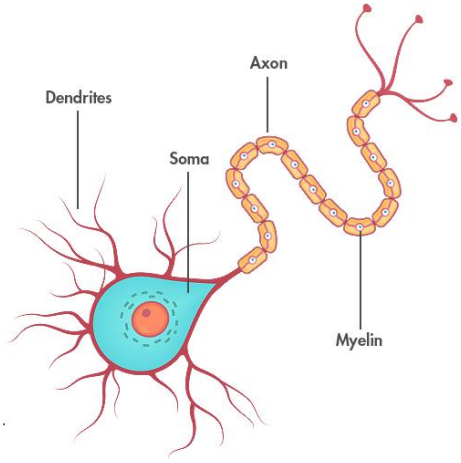



Figure 7. Artistic Rendering of a Brain Neuron (Centre for Neuro Skills n.a.)

The soma can be thought of in a similar fashion to that of the node in an Artificial Neural Network. Its main function is to determine the suitability of incoming information, and whether or not the information meets the necessary requirements to create an activation of the cell and subsequent electrical firing (Centre for Neuro Skills n.a.). Identically, the activation function tied to a node conducts this action by determining whether the significance of the incoming signal values meets or exceeds the artificial value determined by the weights and the activation function. Looking at the dendrites, these are a tree-like structure wherein the roots connect out into other cells - similar to neuron connections between layers within a structure (Centre for Neuro Skills n.a.). Their main role is to collect information from surrounding relevant neurons and convey this information to the soma for processing. The dendrites and axons can be considered similar when discussing the neuron structure with respect to the artificial neuron, however identifying the importance of the axon is necessary. Its main task is to directly connect neurons between one another, and its significance is reinforced by the fact that it is protected by a fatty layer called the Myelin sheath - this ensures signal integrity between neurons and maintains directionality in the signals (Centre for Neuro Skills n.a.). This can be conveyed through to an Artificial Neural Network by thinking of the connections with poor weightings as being dendrites - their main tasks are to collect some form of information and redistribute it to the surrounding neurons (Centre for Neuro Skills n.a.). Whereas the axons can be thought of as the connections with high priority and hold significance in their signal outputs and would therefore be weighted more heavily.

The significance of the architecture of the brain continues through to discussing the Artificial Neural Network of choice for this project - Convolution Neural Networks. It is noted by the paper titled, 'Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future' by Lindsey, G. PhD. that the cognitive model of a CNN very closely resembles the visual cortex structure as found in the brain (Lindsey 2020). The author notes that researchers initially determined that within a cat's visual cortex there appeared to be two distinct types of cells - a simple and a complex cell. The simple cells were identified to respond to bars of light depending on their specific spatial location and orientation. Each cell had an association with a location and orientation and bar which activated the cell the most strongly. The complex cells maintain a weaker response to exact spatial data but appeared to indicate a response to the object itself. Therefore, leading the researchers to believe that the simple cells were feeding the complex cells information about the object and its spatial data (Lindsey 2020). Research that was built on top of these findings in 1980 identified a functional model of the visual cortex. The model was referred to as the Necognitron and serves as the basis of modern Convolutional Neural Network architectures (Lindsey 2020). To understand the significance of these findings with that of the structure of a Convolutional Neural Network, the following image can be used:

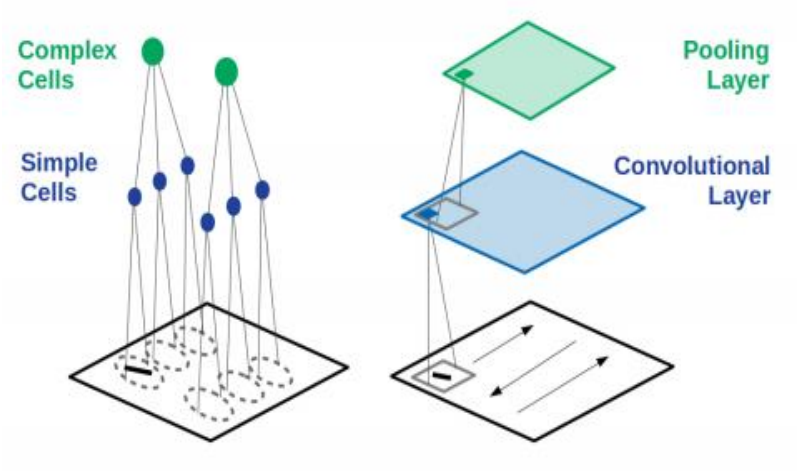



Figure 8. Organic Visual Cortex Relation to Convolutional Neural Network Structure (Lindsey 2020, pp. 3)

This serves as a good point to begin delving into the design architecture of Convolutional Neural Networks. Understanding that simple cells were providing spatial information about an incoming image and that the complex cells were collating information to determine the objective of an image allowed researchers to develop two layers that mimic their operations. These are known as the convolutional layer and the pooling layer - these serve as the basic underlying structure of Convolutional Neural Networks.

To get an understanding of Convolutional Neural Networks, two books serve as a good foundation. Looking at 'Foundations of Deep Reinforcement Learning: Theory and Practice in Python' by Laura Graesser and Wah Loon Keng and 'Programming PyTorch for Deep Learning: Creating and Deploying Deep Learning Applications' by Ian Pointer, the basic operations and mechanism that enable CNN's to operate are well established. The two books resolve the understandings in varying ways - the initial book looks to explain a historic rendition and reasoning for CNN whilst the latter explains the underlying mechanics utilising code as an example.

Throughout his text, Ian Pointer identifies that a standard fully connected neural network can approximate any objective function, however the issue originates in the fact that the time to train this function is largely undefined and is reliant on the end goal and the compute power of the system (Pointer 2019). As a result of this, network structures such as CNN's have arisen to improve efficiency in specific use cases.

The best method to gain an understanding of CNN's is to follow that path that data would follow within the network. To provide a quick overview, for the condition of visual inputs a 2D image is provided to the network. It is normalised prior to being input into the input layer. Then, a convolutional layer will drag a convolutional kernel over the 2D plane (Pointer 2019). This convolutional kernel can be either explicitly defined to detect edges, straight lines, curves etc. or the user can allow the training process to define the kernel structure. This convolutional kernel will move with respect to its size and the stride value that it is provided. The stride value can be either a vector or a tuple. If the convolutional kernel size and the stride value provided are not a factor of the dimensions of the input image, then there will be an overlap effect that occurs - meaning that a kernel could read outside of the defined image value. This is where a padding can be applied, but it is not always necessary. If the designer intends, then the final kernel can skip the overlapping scan effectively shrinking the input image dimensions. The opposing method is to pad the dimensions with a value (generally 0) such that the kernel can make a full scan of the image without skipping any data (Pointer 2019). The number of varying kernels that are used depend on the number of desired output channels defined by the user. From here, the convolutional layers - the output channels from the convolutional kernels - are pooled into a collection of channels. This generally matches the size of the output channels from the convolutional layer and serves to compress the data which helps to speed up training and makes the network more efficient. This pooling utilises a kernel and stride similar to the convolutional kernel, however rather than scanning and replicating the data, it will shrink the matrix according to a ruleset. There are two main methods applied, the Average Pool and the Max Pool. The average pool will drag the pooling kernel over the matrix and take the average value of the values within the kernel matrix whereas the max pool will drag the kernel over the matrix taking the maximum value

from each kernel scan. These are then collated into another layer with which the convolutional and pooling layers are generally repeated multiple times. A final major aspect to understand in a CNN is the use of a dropout layer. This is a layer, generally in the final fully connected layers, that essentially zeros out a random percentage of the data points to ensure that overfitting does not occur (Pointer 2019). It is not always essential to utilise this, but it is generally recommended to implement the process during training and not during the deployment of the network (Pointer 2019). The final main point to discuss in the architecture of a CNN is the utilisation of batch normalisation. This is the recentring of values to a known mean with a minor standard deviation - generally the maximum and minimum values are locked to between 0 and 1 or -1 and 1 depending on the developer.

Starting with the input, let's take an image of size A pixels high and B pixels wide. This image can be either grayscale or RGB. Depending on the normalisation applied to the input (see Chapter 5) there will be some value for each pixel that indicates either a single channel value (0 - 255 indicating the range between white and black for a grayscale image) or in a colour condition a triple channel input will be present with a value of 0 - 255 indicating the intensity of the Red, Green and Blue channels for the image where the image input is in an 8-bit colour range (8-bit meaning 2^8 = 256 values). Using an example similar to the source text will be the most appropriate. Full acknowledgement must be given to Ian Pointer and the book ‘Programming PyTorch for Deep Learning: Creating and Deploying Deep Learning Applications’ for the following description adaptation.

If we use the following matrix as a sample portion of the above image in grayscale, we might have something similar to the following:

$$\begin{bmatrix} 15 & 64 & 12 & 53 \\ 35 & 64 & 51 & 18 \\ 1 & 41 & 1 & 21 \\ 5 & 46 & 5 & 18 \end{bmatrix}$$

With our understanding that a convolutional kernel is scanned across the image. This kernel is generally learned by the network, however as an example the following kernel can be applied:

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

Therefore, scanning this from left to right the following matrix will be produced. Since a 2x2 kernel is being applied, if we wish to minimise overlap in the kernel scans then a stride of 2 will be applied:

$$\begin{bmatrix}15 & 64\\35 & 64\end{bmatrix}\begin{bmatrix}1 & 0\\1 & 0\end{bmatrix}$$
$$\begin{bmatrix}12 & 53\\51 & 18\end{bmatrix}\begin{bmatrix}1 & 0\\1 & 0\end{bmatrix}$$
…etc.

Which will have the following product:

$$\begin{bmatrix}50 & 63\\6 & 6\end{bmatrix}$$

Therefore, the kernel will shrink the dimension of the image depending on the kernel size, stride and the initial dimensions of the image (i.e. if there is pooling that needs to be applied).

This completes the convolutional portion; however we understand that the pooling layer follows on from here. Taking the convolutional matrix output, the pooling layer will apply a pooling kernel over the matrix in a similar fashion with respect to the stride and dimension of the kernel.

Now, using the principal of Max Pooling, which takes the maximum value for each kernel matrix, the following is output from the original matrix using a kernel size of 2, stride of 2 and no padding:

$$\begin{bmatrix}15 & 64 & 12 & 53\\35 & 64 & 51 & 18\\1 & 41 & 1 & 21\\5 & 46 & 5 & 18\end{bmatrix} \rightarrow \begin{bmatrix}64 & 53\\46 & 21\end{bmatrix}$$

Therefore, a rudimentary understanding of the convolutional and pooling layers has been established. From here, fully connected layers or regressor layers are applied to compress the output path into a series of classes (Pointer 2019). This forces the network to identify features such that one of the varying number of output classes is satisfied. It is common to have more than one class hold an equal weighting in an image and it is up to the designer to implement more efficient training methods and data in order to assist the network in better identifying individual class outputs.

A detriment towards the use of CNN's as identified by Graesser & Keng is that the use of convolutions is that they are local. This means that each filter or convolution only assesses a small portion of the image and therefore ignores the general global structure of the incoming image (Graesser & Keng 2020). They

do note however, that the method used to mitigate this is to increase the size of the kernels in the initial layers to increase the size of each convolutional assessment. They do note also that utilising an additional Multilayer Perceptron on top of the CNN to run a full assessment on the image improves output however assessing a case such as this falls outside of the scope of the project. But, having an understanding that a CNN only serves to emulate the visual cortex and not the entire brain, there is certainly room to estimate that supporting networks will be necessary in more complex applications to ensure that a full assessment of data occurs. It is specifically stated by the authors that consideration for networks structures used in control scenarios that, "if the state provided by the environment is an image, include some convolutional layers in the network." (Graesser & Keng 2020).

The authors clearly outline network structures in the text and their best use case which presents a moderate case against using a CNN for a control-oriented scenario such as the one present in this project. However, the literature surrounding the topic is in support of the use of CNN's for control. Therefore, further projecting from the completion of this dissertation would be an interest in investigating the scenario in which a CNN and an RNN-CNN are utilised for a similar task. It makes sense to utilise a Recursive Neural Network (RNN) in line with a CNN from control-oriented scenarios where the data is sequential (Graesser & Keng 2020). This allows the network to have an understanding of how it is moving in a scenario rather than simply reacting to an input state. This could be looked at as a logical progression from a standard CNN for control-oriented algorithms and is worthy of investigation. If the scope of the project had permitted investigating multiple Artificial Neural Network structures, then it would have been included on the basis of strong evidence for its investigation.

There are a number of libraries that are available to the public for the development of Deep Learning algorithms. To list a few there are:

- Theano
- Keras
- TensorFlow
- MxNet
- CNTK
- PyTorch

Each library holds its distinct advantages and disadvantages, but it is clearly outlined in the book, 'Programming PyTorch for Deep Learning: Creating and Deploying Deep Learning Applications' by Ian

Pointer that PyTorch holds distinct advantages when being applied to research scenarios. The author notes that PyTorch's largest competitor is TensorFlow, however TensorFlow utilises statics graphs. These are graphs that have a well-defined size and graph representation upon which operations are executed as opposed to PyTorch which utilises dynamic graphs which allows for faster changes to the architecture of the network during development and during training (Pointer 2019). It is specifically identified by the author that PyTorch is gaining popularity within the research community due to this reason, mentioning that a 200% increase in PyTorch use in published papers for the International Conference on Learning Representations which is a significant increase. A minor point in favour of PyTorch over TensorFlow is its pythonic nature. TensorFlow is not directly pythonic oriented (in its standard form) but it must be recognised that this issue mostly comes down to personal gripe of the developer or a favouring for the programming design that python is well known for.

A reason for selecting PyTorch over TensorFlow was that throughout the Literature Review, it was found that a method of stabilisation used during the training process of a Convolutional Neural Network significantly improves its ability to be used as a control algorithm. This required the use of Spectral Normalisation - which is a training method normally applied to the training of GAN Networks (PyTorch na.). However, it was identified that for a CNN it greatly improves flight stability as will be seen in the Literature Review. After researching the utilisation of Spectral Normalisation for both PyTorch and TensorFlow it was found that it is natively integrated into PyTorch, whilst it was not natively implemented in TensorFlow at the commencement of the project.

### 1.3.4 CASA Regulations and Simulated Environments

Drone operations within Australia are restricted by the regulations set by CASA (Civil Aviation Safety Authority). CASA regulations state that for a drone above 250g, the pilot must hold the valid drone pilot accreditation certificate and register the drone (CASA 2019). Additionally, BVLOS (Beyond Visual Line Of Sight) flight is not permitted. However, a company called Airbotics received the first approval for BVLOS flight in order to map an underground mine in Australia in 2019 (Hosie, E 2019). This opens the avenue for investigating this project in a real world scenario. It must be known however, that failure to abide by regulations with issues that must be taken to court can be fined up to $10,500AUD (CASA 2019). Due to this, it is worthwhile conducting simulated tests and experimentations.

Additionally, another limitation to conducting the project is the high upfront cost of the necessary materials and equipment. The inability to be able to personally fund the project will result in the

simulation aspect of the automation becoming very important. High quality drones are not a cheap investment nor are the mobile compute units which are capable of running real time complex SLAM algorithms or Deep Learning algorithms. Additionally, we want a level of abstraction to be available such that complete measurement and control over the process can be ensured. In order to be able to guarantee this, the best method to focus on ultimately resolves to that of a purely simulated environment.

On top of this desire for complete control over the experimentation process, the largest limitation with respect to trialling the findings throughout the project is the limitation of access to the appropriate environments. Without the ability to access an appropriate environment there will be no capability to validate stages of the project. Not only is there the arrangement of an area for flight that needs to be conducted, but CASA approval and lodgement of the flight within regulations is necessary for the completion of the project. The flight will need to be within a human accessible area to ensure that recollection of the drone is possible if failure occurs.

Based on the provided evidence in this section, there is strong evidence towards simulations environments being favourable for research that aligns with the project type. Therefore, this will be a simulation focussed project.

# Chapter 2 - Literature Review

This literature review focuses on the analysis of the methods of drone automation and entertains the idea of utilising Deep Neural Networks for machine vision-based automation. When inspecting the literature, it is easy to see that the preferred industry standard is the utilisation of SLAM algorithms. There is an evident lack of papers focussing on CNN based automation (in comparison to SLAM) and an even lesser amount focussing on the utilisation of a simulations engine to both develop and compare the two algorithm types. This literature review will limit its scope to purely the main relevant topics for the project. The topics to be reviewed are as follows; Simultaneous Localisation and Mapping for Drone (Section 2.1), Neural Network Based Automation (Section 2.2) and Training in Simulated Environments (Section 2.3). Following from here the Knowledge Gap (Section 2.4) will be identified and clearly stated and the possibilities resulting from the project can be found in Section 2.5.

## 2.1 Simultaneous Localisation and Mapping for Drones

As noted in Section 1.3.2 SLAM algorithms are heavily utilised for autonomous mobile robots under conditions where appropriate input data can be utilised - however what Section 1.3.2 did not describe is the design and characteristics of individual SLAM algorithms. This section will analyse research proven SLAM algorithms and surrounding supporting research both for and against their utilisation in production scenarios. It will also assess two algorithms that are freely provided by OpenSLAM for use in academic research, these are; ORBSLAM and CEKF-SLAM.

Monocular vision-based SLAM is very common however the positional and 3D mapping accuracy of these algorithms can generally fail to be viable for professional and commercial use. This is due to the inability to conduct triangulation on the scene where depth, orientation etc. can more accurately be determined by either stereo cameras or a LiDAR sensor. The paper written by Huang. R et al identifies that their method of visual SLAM maintained roughly a 10cm accuracy for the position of the drone at all times (Huang, R et al., 2015). This would generally be acceptable for generic control. However, given the necessity for professional equipment to maintain a high level of accuracy a discrepancy of 10cm could be considered unacceptable if the end goal was to provide a professional quality map.

Some applications of visual SLAM, such as the one discussed in the paper written by Sergio Garcia, M et al. utilised the control technique of SLAM plus a PID controller for error-based movement within image

frames (Garcia, S et al., 2016). However, in this method they also utilised a secondary external camera for visual frame reference – which did improve the systems 3D positional understanding (Garcia, S et al., 2016). It is not viable to be able to assume the ability to maintain a secondary visual image and as such this method will not be pursued for the project. However, it is worthy to note the control method utilised for the paper.

The utilisation of ORB-SLAM as a reference SLAM algorithm is one that is emulated in a variety of other papers. The paper titled *A Multi-Sensorial Simultaneous Localization and Mapping (SLAM) System for Low-cost Micro Aerial Vehicles in GPS-Denied Environments* by E, Lopez et. al. indicates that ORBSLAM is a visually accurate SLAM algorithm and performs well in well-lit and well-featured environments. The paper linked to the ORBSLAM submission on OpenSLAM titled *ORBSLAM: a Versatile and Accurate Monocular SLAM System* by R. Mur-Artal, et. al. the team identifies in their conclusion that (for the time period in which the algorithm was completely new) the algorithm is extremely well equipped to handle a variety of conditions, with the team directly stating, "To the best of our knowledge, no other system has demonstrated to work in as many different scenarios and with such accuracy. Therefore our system is currently the most reliable and complete solution for monocular SLAM."(Mur-Atal & Montiel & Tardos 2015). The major advantage to ORBSLAM is its ability to only grow the map flexibly, meaning that the map will only develop or expand if the visual content of the scene changes and will store the previous version of the scene after it has been replaced. They also note that ORBSLAM is robust enough that it can recognise its scenery even after what they call a "severe viewpoint change" (Mur-Artal & Montiel & Tardos 2015) which I believe is best interpreted as a claim that the algorithm is capable of handling the kidnap problem - to the degree that it is able to do this however is not explicitly presented in the paper. This will serve as a good basis to want to use the algorithm throughout the project as it is a well-established algorithm with a strong research backing.

The second SLAM algorithm to be utilised in the project is the CEKF-SLAM algorithm presented in the paper *Optimization of the Simultaneous Localization and Map-Building Algorithm for Real-Time Implementation* by J, Guivant & E. Nebot. A simple breakdown of how the algorithm is more efficient than other versions is through its method of feature tracking. It will limit the number of features depending on the resources provided to the algorithm. It will locate the maximum number of landmarks and optimise for those which provide the most relevant information. This means that the algorithm comes, "very close to optimal" according to the authors (Guivant & Nebot 2001, pp. 255). The authors don't particularly attempt to claim that the algorithm outperforms others other than in the fact that it is much more efficient whilst maintaining similar accuracy - mainly in local mapping sizes. They present the algorithm as being desirable for both high-frequency applications where sensors provide data at a high-

frequency (highly featured environment) or for long navigational periods in a local map. Where the implementation of the ORB-SLAM algorithm for the experiment looks at using an algorithm designed to try and limit some of the known faults outlined in the following paragraph, CEKF-SLAM aims to address the computational expense of standard SLAM algorithms whilst trying to maintain an acceptable degree of accuracy. From my own interpretation of the results I would believe that the authors have achieved this, and that the algorithm will serve well in the project.

To discuss the limitations of SLAM, the paper titled *A REVIEW: SIMULTANEOUS LOCALIZATION AND MAPPING IN APPLICATION TO AUTONOMOUS ROBOT* by Agunbiade OY & Zuva T outlines the issues quite well. Whilst the paper is not peer-reviewed, it is still quite relevant as it was published in May 2018 and the information it contains appears to be quite academically credible. Chapter 4 of the paper is where the challenges and research directions for SLAM as a whole are provided and given the number of relevant papers cited, the section appears quite credible. The paper identifies that there is still no one version of SLAM that can handle all situations. It also notes that SLAM algorithms tend to fail or display difficulties, unless specifically programmed, in the areas of; illumination variances and specular difficulties, dynamic environments, kidnap problem and the computational cost of accurate SLAM algorithms. This criteria will serve well as being a focus for the projects investigation.

To add to the discoveries towards SLAM difficulties as noted above, a finding in the paper, *From Monocular SLAM to Autonomous Drone Exploration* by Stumberg, L. et. al. is the understanding that for efficient modern-day SLAM based systems the SLAM aspect can be considered not to be the compute limitation. The authors state that, 'The creation of the occupancy map is visibly the most time consuming part of our method, especially at later time steps when the semi-dense depth reconstruction becomes large.'(Stumberg et. al. 2016). The authors are utilising a low power, monocular drone. The system utilises a third party device to compute the algorithm and return commands to the drone. The authors note that the visual tracking aspect of the algorithm limits its capabilities, stating that, '...depends on the robustness of the visual tracking. If the [drone] moves very fast into … mostly textureless regions, tracking can become difficult.' (Stumberg et. al. 2016). Looking at the results of the paper there is evidence towards the possibility that proposing the idea of SLAM being the compute issue is incorrect, and that for modern SLAM algorithms the navigational component is much more compute intensive. This isn't widely replicated within the general literature covered however it is noted as a point of interest.

Therefore, the literature review on the relevant information relating to the SLAM portion of the project presents that ORBSLAM and CEKF-SLAM are capable of providing the correct basis for evidence against the DNN algorithm. The review of SLAM faults presented by Agunbiade OY & Zuva T presents a strong criterion for measurement on the capabilities of each algorithm within the test environment. The

findings by Stumberg et. al. presents the possibility that for modern day SLAM implementations the navigational aspect presents a larger computational issue than purely the SLAM component on its own which presents doubt on a direct SLAM to CNN analysis. However, it must be noted that it is consistently present within the literature and research sources that SLAM is inherently computationally expensive and thus warrants the investigation.

## 2.2 Neural Network based Automation

There are a limited number of papers that look into the significance of neural network-based control for drone dynamics – the majority application of neural networks for drones is in the feature extraction and machine vision department. However, the paper by Dong Ki Kim and Tsushuan Chen outline a viable method of control using a Convolution Neural Net (CNN) to identify pathways based on frame input (Ki Kim, D & Chen, T 2015). The control method achieved roughly a 70% success rate in finding and identifying its target. This is vastly improved over the results found within the paper by Abbas Sadat, S et al. which showed methods of visual mono-SLAM having only roughly a 30% success rate (Abbas Sadat, S et al., 2014). The main reason for utilising a CNN based control approach was that a CNN controller is not as limited by visual textures and incomplete information as a SLAM approach is and would generally outperform a SLAM algorithm in avoiding collisions with walls due to its training (Ki Kim, D & Chen, T 2015). The network was trained to emulate flight inputs that a pilot might make based on image references and worked extremely well for its desired purpose. This paper provides evidence of the capability of neural networks for complex control and validates the pursuit of this method for a control purpose within this project.

Reviewing the current literature surrounding DNN development and the application of DNN systems appears to be limited in specific applications - if you search for the keywords “DNN” and “Drone” on google scholar the top results all orient around the use of DNN to help identify or categorise scenarios - bar one intriguing result. A paper titled, *A 64-mW DNN-Based Visual Navigation Engine for Autonomous Nano-Drones* published by D. Palossi et. al. sufficiently outlines the development of a DNN. But, its most important findings is the efficiency of the DNN on the system. The system is so efficient that in operation it only utilises 3.5% of the total power envelope for the system whilst maintaining an 18fps input - this is quite impressive considering the same could not be done if the system was based on a SLAM algorithm.

When one attempts to search for a paper on the direct comparative study of SLAM and DNN it is extremely difficult to find any direct comparisons being made. There are conditions such as the paper

titled *Neural Lander: Stable Drone Landing Control Using Learned Dynamics* by G, Shi et. al. where they provided evidence towards the favouring of a neural network-based lander. The Convolutional Neural Network (CNN) they utilised significantly outperformed the standard algorithm. However, it must be made clear that the comparison is made to a mathematically modelled algorithm and not explicitly a SLAM algorithm. The paper also notes a well-known issue for CNN based implementations - the difficulties in collecting sufficient data for training. The authors state that the CNN performs far better than the mathematical model it is being compared to as it reduces the z-axis error from 0.13m to 0m and the x and y axis drift by up to 90%. This is a significant improvement in flight performance. However, possibly the most notable finding of the paper is the relevance of the training of the CNN and the methods that can be used to stabilise the networks behaviour. The utilisation of Spectral Normalisation during the training of the model improves the generalisation capability of the network significantly - this means the network responds more reliably to unclear data or to situations in which it has not directly been trained for. This will be something that I will investigate in the development of the CNN for this project. This paper is a good representation of the capabilities of CNN's compared to other models and acts as a good method to reinforce the case for directly investigating their capabilities compared to that of SLAM based algorithms. Specifically, it serves as a clear indication that the utilisation of a CNN is beneficial for this type of control scenario and reinforces the statement made in the Introduction chapter for the utilisation of a CNN as the base algorithm type.

When assessing the market, a term begins to become prevalent which strongly defines the approach type for this project. The idea of an end-to-end model being deployed. This is simply put as a network in which the entirety of the model is employed to conduct multiple steps in the process. These steps being identification of patterns and objects, path navigation, general flight controls and a variety of additional tasks. The naming scheme of end-to-end is designated due to the capability of the agent to act from input to output solely without the assistance of hardcoded algorithms - such as SLAM, navigational algorithms etc. such that it has not been explicitly defined. These agents begin to build these behaviours solely based on the information that is being fed into their networks - along with the sufficiency of the training program within which they are solely exposed to.

One of the most prominent papers identifying end to end learning and the application of the agent is the paper, *End to End Learning for Self-Driving Cars* published by Nvidia Corporation. The authors, Bojarski, M et.al., noted that the system "optimizes all processing steps simultaneously" and that even with the system being mostly successful, they, "never explicitly trained it to detect … the outline of roads." (Bojarski et. al. 2016). Almost the most impressive aspect of the system is that it was trained on a relatively minimal amount of data and the system learnt to drive in traffic on local roads with or without

the appropriate indicators. The system utilised by the team was built on Torch 7 and ran on an Nvidia DRIVE PX and a Nvidia DevBox. The focus on the use of Torch and Nvidia GPUs as accelerators is supportive of the process utilised within this project. Please note: Torch 7 and PyTorch are not the same. Torch 7 utilises Lua wrappers whereas PyTorch utilises Python wrappers - there are other variations between the two however for the project aim it is not necessary to delineate between them. One thing from this paper I would like to note is the importance of modelling the output control of the network model. The authors stated that using an output of 1/r compared to simply r (r being the radius) prevented the system from experiencing singularity issues as it drove in straight lines (Bojarski et. al. 2016). This can be investigated for its feasibility to be implemented into the control model for this project and presents a solution to possible singularity issues that may be present throughout the project.

Therefore, based on the evidence provided in this section, it is apparent that Artificial Neural Networks are capable of providing autonomous control of mobile robots. It has been specifically identified that Convolution Neural Networks are heavily utilised when visual input is apparent due their high accuracy under these scenarios. The paper published by Nvidia shows that the capability to provide an end to end solution for autonomous control is viable. This section of the literature review has not invested the utilisation of other algorithm structures such as Reinforcement learning, Imitation Learning, Transformers or Large Aggregate Networks as they fall outside of the scope of utilising a CNN for autonomous control. However, it must be noted that these structures are far more recent in design and note investigation in future work.

However, it must be made clear that there is a relatively minor amount of evidence surrounding the use of CNN's entirely without supporting algorithms and therefore its utilisation as a standalone algorithm will serve well for the understanding of the research community. This is the delineation between end to end implementations and supporting implementations of which an end to end solutions is utilised for this project. The large portion of research papers indicate the utilisation of a CNN in line with a mathematically programmed system utilising the CNN as a supporting aspect only.

## 2.3 Training in Simulated Environments

The use case for investing in determining the validity of simulated environments to conduct these tests is best described in the abstract for the paper by Shah, S et. al.  in which they directly state, "Developing and testing algorithms for autonomous vehicles in [the] real world is an expensive and time consuming

process. Also, in order to utilize recent advances in machine intelligence and deep learning we need to collect a large amount of annotated training data in a variety of conditions and environments." (Shah, S et. al. 2017, pp. 1). To which they then proceed to breakdown the topic more in depth throughout the paper. Their point of view reflects the points of view which I currently hold - to put it simply it is that reinforcement learning in its many forms are, "...proving to be a natural way to train various robotics systems." (Shah, S et. al. 2017, pp. 1). The largest limitation to real world training of CNN's is the ability to collect copious amounts of usable data - that is data that is actually appropriate for training a CNN.

Possibly the best proof for the validity of the simulator to emulate real world scenarios and data is that of the experimental results published in the paper. The team emulated a controlled flight that was conducted simultaneously in real time with a real drone and compared it to that of the drone in the simulated environment. Referring to Figure 9 and Figure 10 which is an excerpt from the paper published by Shah, S et. al. indicates that the accuracy of the simulated environment compared to the real-world results are extremely close. They note that small variations in the data likely arise from factors such as integration errors, vehicle model approximations and mild randomisations in winds.

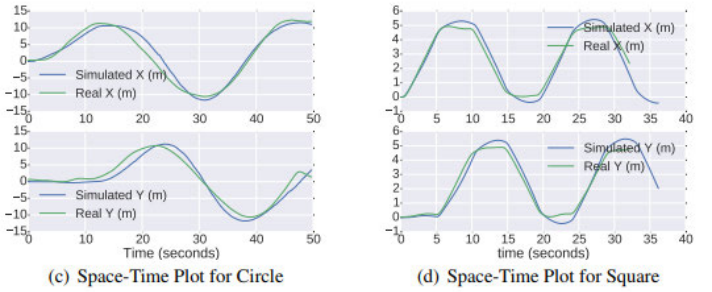



Figure 9 Positional data comparison of simulated and real time drone data (Shah, S et. al. 2017, pp. 12)

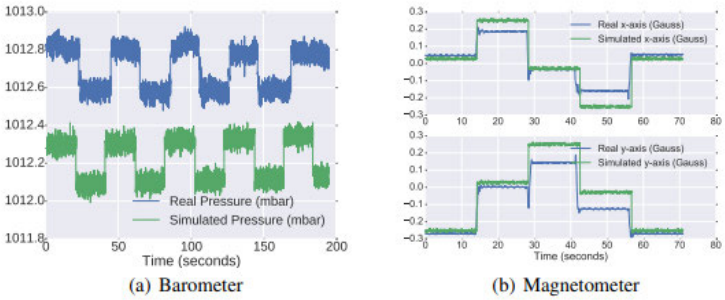



Fig. 10 Barometer and Magnetometer data collected during simulated and real time drone flight (Shah, S et. al. 2017, pp. 13)

When looking at the data presented in Figure 9 we can see that the simulated and real world data align closely. However, the paper notes that sensor data variations do occur and can be explained. The IMU data varies due to the model not accounting for vibrations developed by the rotors which at the time of publishing the paper, had not been modelled yet. As for the Barometer and Magnetometer data, they are

extremely similar. It was noted that the Barometer data varies slightly due to differences in absolute pressure of the environment and due to temperature not being modelled at the time of publishing.

The findings of the paper published in backing of the AirSim simulations engine prove that the data captured and observed in a simulated environment, bar that of the limitations of camera input variability, are close enough to assume that any model trained in it shall be sufficient in a real world deployment. This provides a solid backing to my assumption that simulated environments are sufficient for the comparison of the two algorithm types and that it shall be sufficient for the development and training of the DNN for comparison.

Further evidence in support of the utilisation of the engine for the training and development of a CNN for research purposes is the paper titled, 'Deep Convolutional Neural Network-Based Autonomous Drone Navigation' by Amer, K et. al. This paper aligns closely with the nature of this project in that it utilises the AirSim research toolset and Unreal Engine 4 to develop a CNN for autonomous flight control (Amer et. al. 2019). The paper does not explicitly state benefits or detriments towards AirSim but it can be interpreted that the in-engine tools provided allowed for flexible modelling of their project and that the inbuilt physics for the drone greatly assisted in the completion of the paper. Additionally, the assistive training tools and API's provided for both deployment and development can be believed to have assisted heavily in the success of the project.

Therefore, based on the aforementioned evidence provided in the literature, it can be understood that the AirSim plugin and Unreal Engine 4 are proficient tools that can serve as the underlying structure to conduct the project upon. Although, there are lacking publications and literature available for the utilisation of the tool.

## 2.4 Knowledge Gap

It has been clearly shown throughout the Literature Review that the case for both algorithm types is strong. There is supporting evidence that the performative nature of CNN algorithms exceeds mathematical models and it can be inferred from D. Palossi et. al. that the compute characteristics of a deployed CNN are favourable in comparison to a SLAM algorithm, especially when referenced with the findings by L. Stumberg et. al.

It has been clearly defined due to the lack of evidence that the there is case to develop a direct understanding of research proven SLAM algorithms against the performance of a DNN algorithm for

automated flight within a simulated environment. There are evident cases in which both algorithm types are performance tested singularly yet there is a clear lack of understanding on how the two algorithm types perform within Unreal Engine 4 utilising the AirSim research plugin. There is also a clear lack of understanding of a comparison between the two algorithm types. However, it also presents the fact that SLAM vs CNN is possibly the wrong question. This is why the aim of the project is to establish an understanding of their fundamental characteristics within a simulated environment and to establish consensus on the performance variations of each algorithm type.

Comparing the two algorithms in a simulated environment provides a uniform test environment in which the performance nature of the two algorithm types - both in compute and in inference as defined by Agunbiade OY & Zuva T - can be clearly tested and utilised as validation data. Due to the abstract nature of each algorithm type and their method of deployment it will be difficult to directly associate compute characteristics, however progress can be made in defining a clear understanding of both the deployment aspect, the compute aspect and the limitations of their ability when it comes to operating the algorithms within Unreal Engine 4 utilising the AirSim plugin.

## 2.5 Importance and Possibilities

Having a clear understanding of the most appropriate algorithm type for conducting automation of mobile robots - notably in simulated quadrotor drones - will allow for the industry to begin to make conscious efforts towards providing the optimal solution for their automated products. It is clear that this is a burgeoning industry with quite an infantile understanding of the market options that are present.

Looking at a field which is considerably the most desiring of accurate end-to-end automation is the mining industry. Within the paper written by Jones, E et al., the importance of drone technology for underground mining operations is only just starting to begin being realised (Jones, E et al. 2019). The industry uptake of the HoverMap system and its applications are extensive with obvious improvements in safety, efficiency and productivity for underground mines (Jones, E et al. 2019). The same paper identifies the future of these automated drone systems and some of the requirements that need to be satisfied. Out of the list, there is 1 main topic that applies to the importance of this research project and is categorised as;

- Autonomy Level 3 – Autonomous Exploration
    - Single-click mission execution
    - No waypoints

The ability for level 3 autonomy could be improved as a result of the understandings established within this paper. It is only natural to expect this level of control and it is currently not being satisfied by the current market.

Therefore, it has been shown that the pursuit of this project has real-world and research-based importance. There is an industry demand for defining a clear understanding of the capabilities and characteristics of the two algorithm types in simulated environments. It will better inform the industry and aim to satisfy points of misunderstanding within the industry. The literature review has identified a gap in the knowledge base with respect to these algorithms in real world and simulated environments when assessing their direct comparisons and this project shall serve to begin establishing a precedent on the exact comparison of their characteristics.

# Chapter 3 - Methodology

This section focuses on the aspects required to execute the research project. The project and experimentation methodology will be discussed in detail to set the precedence for Chapters 5, 6 and 7. A risk assessment will be conducted to define appropriate guidelines for the project. Additionally, aspects such as the necessary resources and data points with the method of analysis are noted.

## 3.1 Project Methodology

In order to achieve the outlined objectives in, there is a need for additional prior readings and actions to be completed. The areas of further reading can be broken down to:

- Types of Neural Networks & Neural Network Control Systems
- Construction and training of Neural Networks
- SLAM algorithms
- Machine Vision
- Image filtering and manipulation techniques
- Simulation techniques within Unreal Engine
- Dynamics of drone control theory

Once these further readings – not all are essential readings that contribute to the furthering of the Literature Review but are for personal or understanding purposes – are complete the project can begin to move into the corresponding phase. The phases for the project are outlined as such:

1. Literature Review and Resource Acquirement: This phase focuses on identifying the critical literature relating to the topic and shall be utilised to present the understanding of the current research platform in the project area and present the gap in the knowledge.

2. Development of Simulations Environments: This phase is focused on the development of the simulated environments for experimentations.

3. Development of CNN and Conversion of OpenSLAM Algorithms: This phase is focused on the development of the DNN and the OpenSLAM algorithms for the project. This runs in line with the second phase of this project. See Chapter 5 and 6 for further details.

4. Data Collection and Training of CNN: This phase involves the collection of appropriate training data and subsequent training of the chosen CNN for the project. See Chapter 5 for further details.

5. Conduct Experiments: This phase focuses on the implementation and execution of the algorithm types noted in a controlled environment such that their characteristics can be validated.

6. Assess Results and Update: This phase focuses on analysing the data and identifying resultant patterns or notable results that may need to be re-assessed due to anomalous outcomes.

7. Repeat Experimentations: This phase is conducted as a reassessment phase of the initial experiments conducted in Phase 5. This is conducted in order to either affirm outcomes or to challenge the results of the experiments with evidence.

8. Identification of Results and Future Studies: The final phase is an identification of the results which leads to a clear conclusion for the proposed hypothesis for the project. From here, notable future efforts can be recorded for students who may wish to continue the project in a different direction.

## 3.2 Experimentation Methodology

Initially, it was proposed that there may be a possibility of testing and comparing the algorithms in not only the simulations environment, but also in a controlled real-world scenario. Due to the limitations that became present with the COVID-19 pandemic in Australia and the travel bans placed throughout the first half of the year the ability to entertain this point was nullified. Additionally, personal distress experience throughout this time adversely impacted the project.

The experiment was conducted in Unreal Engine 4 (v. 4.24.x) utilising the AirSim plugin (v. 1.3.1). When looking at the utilisation of the prior software, there is a clear indication that it will be necessary to run the simulations within a Windows environment. This is influenced by the limitations outlined by AirSim in their GIT page stating that Unreal Engine does not currently support editing in a Linux environment. It could be feasible to investigate running on a Mac platform utilising OSX however due to MacOS not currently supporting up to date Nvidia drivers natively it is not a suitable option. In order to execute the

algorithms chosen, the CNN will be run ‘in-the-editor’ by selecting the Python script as an execute at launch script within Visual Studio 2019 giving access to the script for AirSim. As for the two SLAM algorithms, they present a significant amount of difficulty towards successfully running the algorithms. CEKF-SLAM is written natively in MATLAB and AirSim is capable of connecting to SIMULINK in order to execute commands as mentioned in Appendix G however these are not methods supported by AirSim or MATLAB. When looking at running the MATLAB code within Octave as was intended for this project it does not have complete resources as would be found when using MATLAB. This is expected to present significant difficulties and it is likely that MATLAB will be used to run the programs. In order to run ORBSLAM there is a requirement to use RoS Fuerte/Groovy on Ubuntu 12.04. These are both packages that are largely out of date (5+ years old) and will therefore need to run on either a virtual machine with control via utilising UDP out from the Virtual Machine, or it can be run using a Docker container built with only the necessary kernel for RoS Groovy and Ubuntu 12.04 with which ORBSLAM can run. In both cases, the need for UDP control and additional layers of abstraction is expected to provide skewed data with respect to the compute performance characteristics. However, with the finding of a MATLAB variant of ORB-SLAM it is viable to execute the program using the MATLAB AirSim link previously noted. The occupancy grid code outlined in Appendix H provides a strong base for the utilisation of both SLAM algorithms on a common navigational component for the project and aims to improve the quality of evidence for validation.

Therefore, the experimentations were entirely confined to simulated environments and training data collected entirely within the engine itself. The experiments are coordinated into four main areas. Within the four main areas, the experiments are split into 2 phases - individual and combined. These are defined as follows:

The individual phase explicitly assesses each of the focus areas as defined in the Project Scope (Section 1.2.2) and in Chapter 2 (Section 2.1) which are; image illumination issues, dynamic problems and the kidnap problem.

These focus areas will have their experiment philosophy outlined by the following ideology:

- Image Illumination Issues: Attempt to trick the camera inputs, using mirrored/reflective/see through, along with well and poorly lit areas. Shadows and highlights will play a big part in visual based input. Whole point of this aspect is to trick the system into not being able to appropriately identify its environment's features.

- Dynamic problems: moving objects in the environment in both a linear and non-linear path in an attempt to trick both systems into improperly assessing its placement in the environment.

- Kidnap problem: By removing the camera and other sensor inputs the system will have to fly blind for a small amount of time. Doing this generally causes SLAM algorithms to fail when attempting to begin remapping however we will assess how both systems handle this and identify whether it can be trained into the CNN how to solve this issue.

The combined phase incorporates the aspects of all cases together into a few environments or scenarios in order to assess how it will dynamically handle the worst case conditions that it can be exposed to - somewhat of a 'real-world' test of the efficiency and accuracy of the algorithms. There will be four areas designed to emulate the scenarios. These will be built using the textures provided within Unreal Engine 4.

The performance of both algorithms will be recorded on a failure rate basis (number of object collisions or complete flight failure). This is a measure of accuracy and is looking at the statistical performance characteristics of each algorithm. Additionally, the compute load of each algorithm will be recorded to understand the ability to deploy the algorithm in mobile scenarios.

There will not be an attempt made to create a 1:1 scenario in regards to the utilisation of sensors for each algorithm type. The CNN will utilise only a visual monocular input and will make judgements for flight control based on this information. As for the two SLAM algorithms, they require the use of an IMU (virtualised in this case) along with visual monocular input data. It is common for visual SLAM algorithms to require the use of two cameras to achieve visual stereo SLAM however the two chosen SLAM algorithms specialise in the use of a monocular input. This decision was made not only to attempt to provide a reasonable standard in terms of the amount of data each algorithm can work with, but also due to limitations within AirSim. It is possible to implement a multi-camera setup utilising two cameras on the drone however it natively supports a monocular camera in the API and for the sake of stability it was not elected to alter this.

Additionally, literature evidence and anecdotal findings throughout the paper will be utilised to attempt to draw a conclusion on the project's outcome.

## 3.3 Resources

The preliminary hardware resources for the project are listed below in Table 1 along with their associated costs, source/supplier and the acquisition status of the item. Subsequently, the software resources can be found listed in Table 2. The lists will cover the main items necessary for the project and will neglect minor or insignificant resources that can be altered without affecting the outcome of the project by any measurable degree i.e. power supply or case used.

The hardware resources listed below are configured in such a way that it satisfies the recommended operating requirements outlined by both Epic Games for the use of Unreal Engine 4 and by Microsoft Research Team for the appropriate use of AirSim. The hardware resources listed are a reflection of the system conducting the simulations.

Table 1. Hardware Resources

| ITEM | QUANTITY | COST | SOURCE | Acquired |
|---|---|---|---|---|
| AMD 3900x | 1 | $849 | Student | YES |
| 32GB DDR4 RAM | 1 | $499 | Student | YES |
| NVIDIA 2070 Super | 1 | $1199 | Student | YES |
| 1TB SSD | 1 | $289 | Student | YES |

Table 2. Software Resources

| ITEM | QUANTITY | COST | SOURCE | Acquired |
|---|---|---|---|---|
| Unreal Engine 4 | 1 | -- | Epic Games | YES |
| AirSim Plugin | 1 | -- | Microsoft Research Team | YES |
| GIT | 1 | -- | GIT | YES |
| Microsoft Visual Studio 2019 | 1 | -- | Microsoft | YES |
| KITTI Training Dataset | 1 | -- | KITTI | YES |
| Microsoft Visual Studio Code | 1 | -- | Microsoft | YES |
| Python 3.x.x | 1 | -- | Python | YES |
| PyTorch | 1 | -- | Facebook | YES |
| MATLAB | 1 | $115 | MathWorks | YES |

| | | | | |
|---|---|---|---|---|
| Octave | 1 | -- | GNU Octave | YES |
| VirtualBox | 1 | -- | Oracle VM | YES |
| Docker Toolbox | 1 | -- | Docker | YES |

## 3.4 Data Analysis

The results will be analysed on a mean basis where a single condition will be run multiple times and the mean of each run will be compared along with the max, min and spread in time variants such that the mean and variance of each condition can be compared to determine optimal results in varying conditions.

The compute characteristics of each algorithm type will be assessed along with the general compute requirements necessary to develop the environment. A focus will be placed on determining the base level compute requirements for an environment by using manual control of the drone prior to deployment of each algorithm type. This will allow for a relative reference to the compute impact of each algorithm type and to understand when compute limitation is affecting an algorithm's performance.

The data to be analysed can be categorised into 2 sectors - compute characteristics and performance characteristics. The compute characteristics - CPU, RAM, GPU utilisation etc. will be recorded with background tasks accounted for over multiple runs. There will be no temperature controlled or scenario-controlled data included as attempting to maintain these values will be almost impossible and relies too heavily on external manipulations such as ambient temperatures. The performance characteristics are the accuracy and time taken to complete tasks. Repeating each experiment at least 5 times will produce an adequate average value however they will be run as many times as deemed necessary to produce an average result. Training of the CNN is not to be assessed. The training of the CDNN will be included in the discussion and understanding of the project but its direct impact is not a focus of the project.

Additionally, anecdotal evidence and findings from the literature will be assessed to provide a conclusion on the projects findings.

Table 3.5.1 Includes the necessary data that will be produced from each experiment and how it will be interpreted with respect to its data type.

Table 3. Methods of Results Analysis

| FOCUS | FACTOR | ANALYSIS METHOD |
|---|---|---|
| **COMPUTE CHARACTERISTICS** | CPU | Percentage Utilisation |
| **COMPUTE CHARACTERISTICS** | GPU | Percentage Utilisation |
| **COMPUTE CHARACTERISTICS** | RAM | Percentage Utilisation |
| **IMAGE ILLUMINATION** | Shadows | Collision and detection accuracy |
| **IMAGE ILLUMINATION** | Highlights | Collision and detection accuracy |
| **IMAGE ILLUMINATION** | Specular Reflections | Collision and detection accuracy |
| **IMAGE ILLUMINATION** | Transparent materials | Collision and detection accuracy |
| **DYNAMIC PROBLEM** | Linear moving objects in environment | Collision and detection accuracy |
| **DYNAMIC PROBLEM** | Non-linear moving objects in environment | Collision and detection accuracy |
| **KIDNAP PROBLEM** | Removal of inputs | Ability and percentage success of re-localising and returning to path |

## 3.5 Risk Assessment

The risk assessment is based on the risk matrix provided by the Health Care Governance of Australia (AHCG 2019). It is made clear within the document that so long as a risk matrix maintains consistent throughout a document it will satisfactorily highlight the risks involved (AHCG 2019) – as such it shall remain throughout the project. This risk matrix will be used to assess both personal and project-based risks. The risks and hazards involved with the project can be found outlined below in Table 3.6.1 along with their risk/hazard mitigations.

Table 4. Consequence Matrix

| | **CONSEQUENCE** | | | | |
|---|---|---|---|---|---|
| **LIKELIHOOD** | Insignificant (1) | Minor (2) | Moderate (3) | Major (4) | Extreme (5) |
| Rare (1) | L1 | L3 | L6 | L10 | L12 |
| Unlikely (2) | L2 | L5 | L9 | M3 | M7 |
| Possible (3) | L4 | L8 | M2 | M6 | M9 |
| Likely (4) | L7 | M1 | M5 | H1 | H3 |
| Almost Certain (5) | L11 | M4 | M8 | H2 | E1 |

Table 5. Risks/Hazards Reduction Table

| **Risk** | **Mitigation** | **Rank Prior** | **Rank After** |
|---|---|---|---|
| Loss of Data | Maintain, at a minimum, 3 concurrent and up to date copies at all times. Software will be version controlled and backed up using GIT. Documents will be stored in the cloud using Google Docs. | M6 | L10 |
| Confidentiality Breach | Maintain security, ensure proper conduct and encryption are followed if necessary. | M7 | L12 |
| Legal Issues | Contact a lawyer if necessary. Understand the legislation surrounding the topic and the regulations enforced via CASA and other relevant bodies. | M9 | L12 |

| | | | |
|---|---|---|---|
| Personal Harm | Develop and follow an appropriate safety guideline. | M6 | L10 |
| Copyright Infringement/Patent Infringement | Maintain a degree of difference, contact a lawyer if necessary. Different training data is a good separation factor. | M6 | L10 |
| Inability to organise real world tests | The focus of the project is on simulated tests and simulated results. Failure to test the device in a real-world system is outside of the defined scope and is only applicable if time is available. | L7 | L7 |
| Failure to produce working simulations | The failure to produce working simulations does not entirely invalidate the project. Any findings that can produce evidence are considerable and establishing consensus on the topic is extremely beneficial to the research community as it is poorly defined. | M7 | L12 |

## 3.6 Ethical Conduct

There must be an effort upheld to the highest degree to safeguard the lives of any and all who are involved. However, considering that the project is entirely simulations based than the ethical conduct of the project must be such that any and all respective parties and software are licensed or acknowledged appropriately, and any prior work is cited and acknowledged in accordance with the guidelines. Additionally, an effort has been made to conduct all portions of the project aligned with the Engineers Australian Code of Conduct as found in Appendix K.

# Chapter 4 - Development of Simulated Test Environments

The major benefit of running the project in a simulation-based environment is the ability to design and develop environments which can be specifically tuned to provide a desired result. This chapter will discuss the development process of the environments to be used for the project.

There will be three environments produced, each focussing on a different aspect of the difficulties identified for SLAM algorithms. These are as follows:

- Looking at the first environment produced, it will focus on image illumination issues. These being the issues surrounding reflections and specular conditions such as lighting difficulties and optical illusions as defined prior in project.
- The second environment produced will be directly focussed on dynamic components of the environment. These being the ability to have moving objects that can disrupt the localisation capabilities of each algorithm type.
- The third environment is a semi-realistic scenario environment in which the two above issues will be combined. It will be attempted to emulate real world conditions, such as dynamic conditions coming from moving objects that are realistic like trees, grass etc. but there is also a need to use some artificial components to properly test the algorithm strengths such as moving boxes and spheres on defined paths.

I would like to remind the reader about the Kidnap issue as discussed in Section 2 and 3. This will be implemented in the disabling of the camera from the model, essentially removing visual capabilities in a somewhat realistic scenario. A ‘gate’ (called a trigger in the engine assets) will be applied to the map which removes the camera for a randomised period of time between 1 and 5 seconds. This is an arbitrarily chosen time limit which will be applied to the simulations using a randomised time value selected. This aims to truly test each algorithms ability to maintain stability without visual input and then also its ability to re-localise within the environment.

As a quick overview of the building process of an environment, it is a long and difficult process. Removing the steps required to open the application and create a new file with the necessary plug-ins installed, the first true step is actually to create a rough hand sketch of the desired path so that you can

visualise the map to be built. From here, using a BSP brush and geometry it is possible to develop a rough outline of the path that can be built into the environment. This allows for judging size and suitability of the design for the given task. Now, BSP (Binary Space Partitioning) is not realistically suitable for the simulations compared to meshes with textures applied to the models.

Therefore, a rough design process (this is based on my own process, although I didn't always follow this as sometimes jumping between steps is necessary) would look as follows:

1. Create a rough sketch of the environment outline
2. Create a BSP outline of the desired environment
3. Ensure the environment sizing and outline is suitable
4. Replace BSP outline with static mesh
5. Apply textures to static meshes
6. Implement dynamic components such as moving objects and interactive objects
7. Compile Visual Studio project files
8. Test and debug environment

Now, it must be understood that it is extremely difficult to create a realistic 1:1 emulation of a real-world scenario. Unreal engine measures the environment in terms of ‘unreal units’ which are not a representation of any realistic measurements. But they are somewhat relative to the standard cm and are generally treated this way by developers. However, considering that the in-engine meshes do provide static meshes with a rough estimate size (in cm) then these can be used as a reference for developing the in-engine environments.

The model used to estimate sizing is the wall_500x500 model which has the rough dimensions of 500cm width, 500cm height and 20cm in depth. This works well for estimating path length and the width of the path such that it can be properly varied throughout the building phase of the environments. I would like to point out again that a true 1:1 emulation of the real-world is not necessary as the goal with this project is to compare the two algorithms. If the goal was to develop networks for development, then this also would only be a minimal issue and transfer learning on an almost true 1:1 emulation of the environment could still be conducted whilst maintaining the original Deep Learning algorithm. It is reasonable to believe that in the future, as we are beginning to see it occur today, there will be an almost true 1:1 recreation of the real world in a simulated environment such that we could train a multitude of networks on the data that is able to be captured within the simulations. But, I do believe that we will approach the 1:1 emulation

asymptotically and it is not likely that we could truly emulate the randomness in nature down to a particle level for the entirety of the universe - for localised portions of earth perhaps but on a galactic scale it is far beyond today's capabilities.

As much as I would like to completely explain the building process, such as map warping and producing height maps; I think this is best left as a task for the reader if they wish to truly understand the use of the Unreal Engine. I want to use this chapter to discuss and explain the philosophy behind the designs and how I went about building them, this will of course involve explanations but will not walk through exactly how to do it yourself.

When beginning a new ‘level’ the user will be presented with the following screen. From here, the development of the new environment will begin following the steps listed prior. A generic floor and a few other items are generated automatically, however, they are far too small in scale. See below for reference:

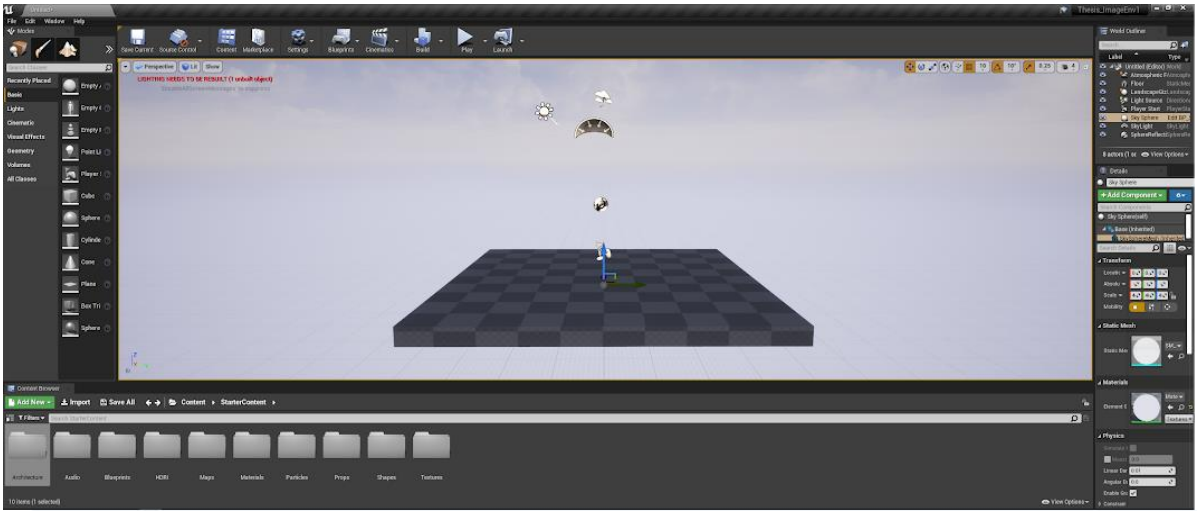


Figure 11. Standard Starting Level with Blueprint Models Removed

Therefore, using the 500x500 wall model for scale we can expand the floor map to look more like the following:

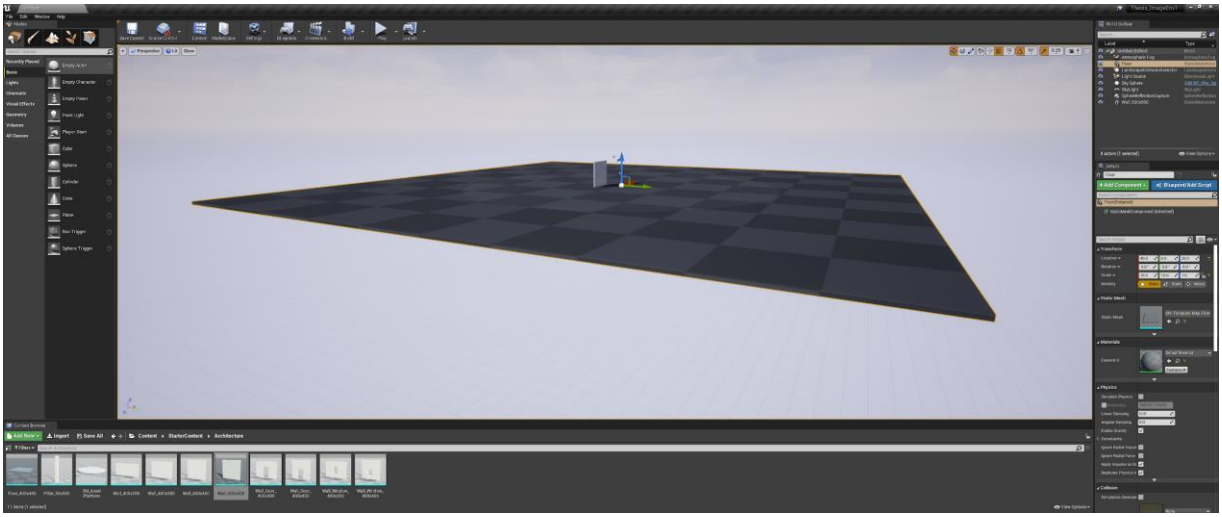


Figure 12. Scaled Floor Plan of Initial Floor Map

The floor plan has been scaled up by a factor of 10 in the x and y direction however the z direction does not need to be modified as its thickness is not relevant for the simulation and it makes sense to keep it as thin as reasonably possible to limit any technical issues. Using this as a base, we can now begin designing the simulated environments for the project.

## 4.1 Landscape 1: Image and Specular Difficulties

For the first landscape, the focus was on producing an environment around the known image illumination issues that SLAM algorithms experience. Essentially, this environment's aim is to utilise textures and lighting in such a way that obscure reflection and specular issues such as glare can be produced in order to hinder the camera and the algorithm's ability to detect its environment. The flow on from this is that poor input data is expected to derail the algorithms ability to conduct SLAM - the mapping and the localisation that the algorithm is built for. Reflections are expected to confuse the feature tracking component of a SLAM algorithm and the glare/lighting issues are expected to do a similar job - however it is through either saturation or removal of data rather than proposing ‘confusing’ data to the algorithm.

For instance, how would a rather generic SLAM algorithm understand that a drone is moving forward with respect to a mirror? How about when reflections display repeated points of reference/interest to the

algorithm? We understand that this is an issue for almost all current algorithms and producing such a confusing environment may help us to understand exactly how each algorithm type performs under these conditions.

Now, a completely mirrored environment is not going to work. Humans even struggle in mirrored environments, that's why they exist as carnival attractions as the confusion is novel for us to experience. Therefore, we do need to introduce some form of texturing and differentiation between segments in the map as a way to indicate progression in the environment. For this level, I believe that following a somewhat corridor-like design will be useful. I am be able to simply populate the environment with the textures required to create the specular difficulties.

Also, having a controlled width means that creating plausible collision events between the drone and the walls can be fairly controlled in which I can place objects that will disrupt a simple straight path through the corridor. Enclosing the path with a floor, walls and a roof means that I can create shadows using a global light source and windows. It also means I can create dimly lit and overly well-lit segments throughout the path. Another major advantage of utilising a completely enclosed environment such as a corridor or hallway style design is that I do not have to set a boundary for the z direction for the drone. This also allows me to determine the capability of the drone to a greater degree.

Building using concrete textures and static meshes, we see the starting area of the map looking as such:

Figure 13. Starting Point of Map

There is a point light placed where the model for a hanging ceiling light resides which is outputting light in a 360 degree window and even without the lighting being appropriately rebuilt, it is obvious that the

room now keeps a reasonable amount of interior lighting and shadows given by the external ‘sun’ source and the windows.

One of the most important things to do during the building process is to continually check the lighting of each section as you build it. This means rebuilding the lighting when a section is finished to check that light bleed is not occurring and that there are no collisions between static meshes that would indicate an incomplete model. Having completed the hallway extension of the above starting point on the map, and rebuilt the lighting, the outcome is as follows:

Figure 14. Rebuilt Lighting for Starting Hallway

Looking at the above image you can see that the light reflects from the windows properly onto the ground level, the point light source is illuminating the starting point appropriately and there doesn't appear to be any flickering or incorrect light in the scene. I will provide an example of an improperly sealed model as follows:

Figure 15. Example of Break in Ceiling Model Allowing Light to Bleed Through

Figure 16. Ceiling Break Fixed - Note there is still light bleed occurring however in the roof to wall attachment

In order to properly combat light bleed on complex textured static meshes, the lighting resolution needs to be increased and a UV map needs to be produced. However, for the project's purposes, incorrect lighting conditions such as this aren't considered to be a fault and therefore a UV map is not required for each static mesh. I will instead aim to take advantage of this using reflective textures and utilise the light bleed as a disruptive source.

Figure 17. Completed Corridor Layout Without Features

Now that the general layout is complete, the interior will need to be populated and the appropriate lighting will need to be implemented. Now, considering I am locked to the original assets provided by Unreal Engine as I am not able to produce complex models within a reasonable time frame, I will be placing textures over the top of simplistic models to achieve a desired end result.

When I began populating the path, I noticed that realistically the path length is too long. I have chosen to shorten the course length to roughly ⅓ its original length as seen below with the red line indicating the end point of the course:

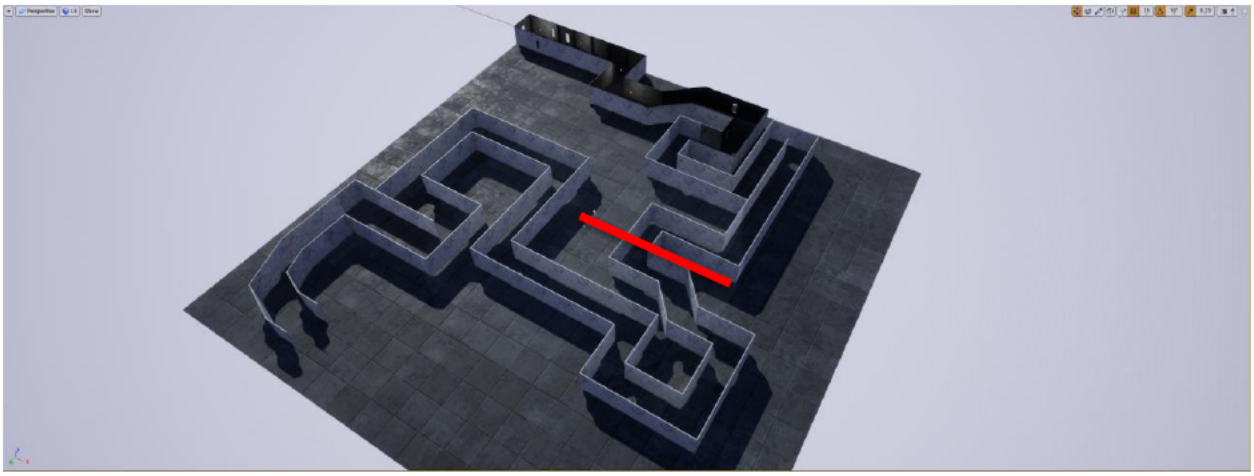


Figure 18. New finishing point indicated by the red line

In order to appropriately understand the visual that I am presenting to the drone I will provide a walkthrough of the environment in the form of screenshots from within Unreal Engine 4. The conditions that the environment is built in is all default settings with the lighting quality set to production.

The starting area for the map is set up in a way that the initial hallway presents no real difficulties to the drone other than the two open windows on either side of the hallway which bring in ‘natural’ light and the two lights in the hallway which through shadows and create a lighting differential within the hallway. If you take note of the brick pillars on either side hallway - these are being used as consistent reference points throughout this map.

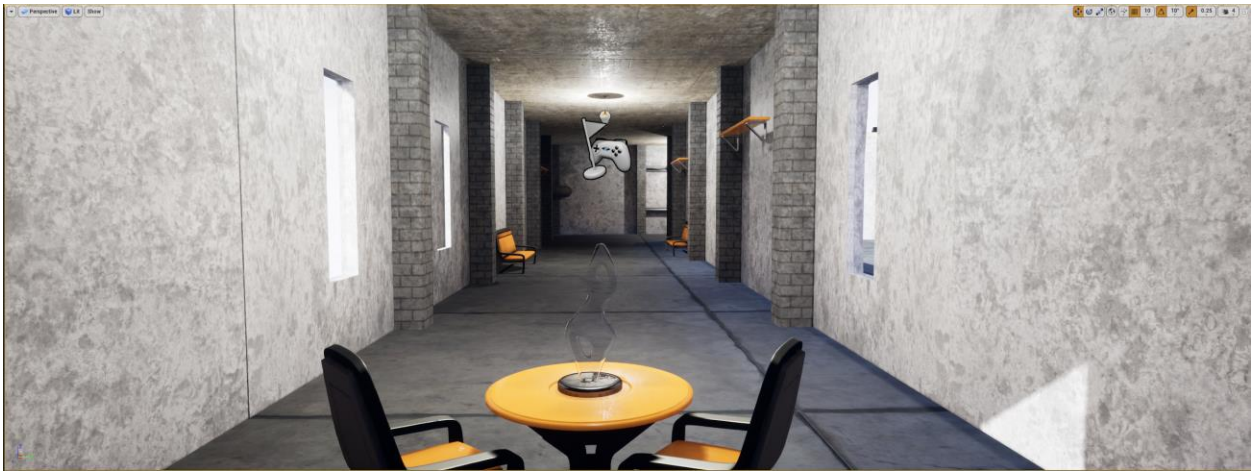


Figure 19. Starting walkway with player start in position

The second hallway is a short run and serves only to test the drones ability to detect the general shape of the hallway and detect the open space to continue moving forward. The light is used to produce glare on the mirrored surfaces however this and the first hallway are not explicitly aimed to ‘fail’ the drone.

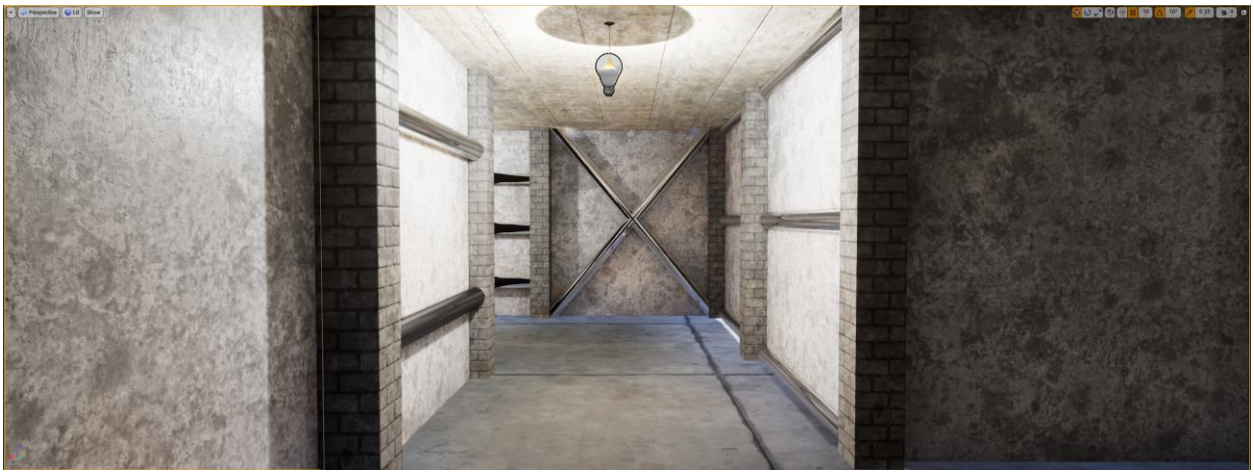


Figure 20. Second hallway

The third run is designed to be the first actual test of specular understanding for each drone. It is almost impossible with this current package to replicate true specular conditions however the inclusion of ray traced lighting would sufficiently improve the lighting accuracy - however the computer used for the environments is not powerful enough to be able to run ray tracing and GPU accelerated compute at the same time without limiting the compute availability for all algorithms.

Figure 21. Beginning of third hallway

Looking at the third hallway run above, the most notable feature in Figure X is the two chrome crossbars connecting the walls. These are used to test each algorithm's ability to detect the edges of the two bars and implement them as features in the mapping process. Looking at their placements, it is obvious that they are not purely there to disrupt the flight path and more to confuse the algorithms.

Figure 22. Middle of third hallway

The middle of the third hallway presents some inconsistencies compared to the previous hallways. From the starting position to this point the specular difficulties have either been orthogonal to the corridor or purely specular from lighting/glare. This hallway uses obscurant structures at angles that aren't consistent with the hallways direction. There is an expectation that non-orthogonal structures such as this will reflect the surroundings in a way that will be the most 'confusing' for each algorithm type. The circular reflective structures are hollow pipes with a darkened metallic reflective metal texture applied such that its reflective nature changes paths depending on the position of the drone with respect to the interior and exterior of the pipe.

Figure 23. End of third hallway

The last portion of the third hallway is set up to present a few varying issues at the same time. The chrome crossbars are difficult to detect edges and even to distinguish its location in the corridor. The major point to note for this section is the light placed at the end of the hallway and the mirror placed on the wall behind that - this is to emulate conditions for glare and reflections simultaneously. This is a condition that any automated mobile robot would be expected to experience in the real world.

Figure 24. Fourth hallway

Moving on from the third hallway is a short fourth corridor. This is purely focussed on reflections and obscure lighting. The window allows for natural light to come into the section and is the only source of light for this corridor. The mirrors are different shapes and sizes such that it is easier to determine if either algorithm type can detect a reflection - when turning the corner from the third section to the fourth the mirrors reflect an obscured view of the continuing path and will be a good test of the algorithms ability to determine a forward path.

Figure 25. Fifth hallway

This path is extremely short but the utilisation of multiple mirrors and an inconsistent light source (the sparks emanating from the ground in the centre of the path) align with the short through lighting creating long vertical shadows will again test each algorithm's ability to detect features in the path.

Figure 26. Sixth hallway

This is a short path that really only serves as a way to re-introduce ‘natural’ light into the next section. Glass has been placed on the right hand wall to produce a wall that's inconsistent with the previous walls and a mirror is placed on the left wall to produce the same effect however this section is not truly aimed at detecting any major performance characteristics and mostly serves as a transfer to the next section.

Figure 27. Seventh hallway

For the seventh hallway it is a glass/see through run. I wanted to use a shape that wasn’t exactly a rectangle made to appear like glass. I wanted a model that was complex in shape and with rounded edges such that it could more appropriately create reflections and obscure light paths. I thus chose the SM_Statue prop and suck its solid base through the surrounding static mesh to hide any non-glass features of the statue as they may become indicating features for each algorithm to detect.

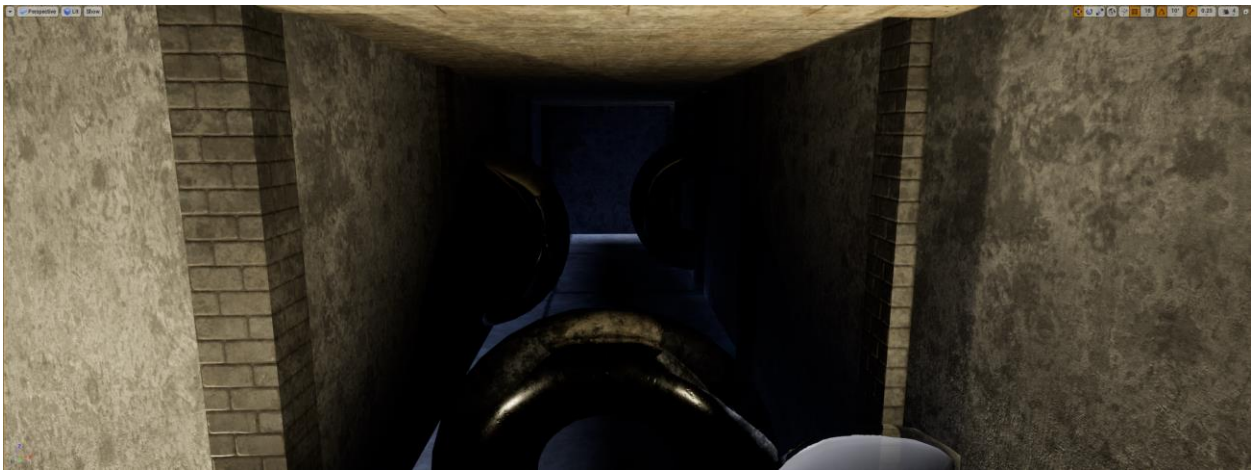


Figure 28. Eighth hallway - darkened room with reflective components to

The final two hallways are aimed at utilising poorly lit corridors and the drones ability to detect edges on objects in the path. The eighth corridor aims to use reflective shapes as they give a more pronounced visual on their location in the path and a poor visual on their true shape and boundary of the geometry. This path is not overly populated and simply provides three obstacles that obstruct the simplest straight path through the run but not much movement is necessary to successfully navigate the corridor.

Figure 29. Final hallway

The final path is difficult to appropriately represent due to the lighting conditions but looking down the path there are simple poles oriented either vertically and horizontally in varying distances from one another. There is no possibility of following the central path through this corridor to successfully complete this section and thus it will require the system to appropriately detect features in the path to be able to complete the course.

Whilst there are a variety of methods available to improve the capability of the lighting for levels like this, such as baked lighting, they will not be implemented in this level. The improvement of the lighting system and lighting accuracy is a feature I would like to improve in future work and I believe that in order to appropriately measure the effect of a lighting system on camera and detection functions, real time ray tracing would need to be implemented. This is a feature that will be native to Unreal Engine 5 and would be a natural progression for this area of study - I believe it would also create a more realistic environment in general compared to rasterization methods of light calculations. But I do also believe that in this case, the inconsistency in lighting conditions and inability of the system to render accurate lighting conditions for all cases presents an interesting specular condition for both algorithms to be exposed to.

Realistically, the goal of creating environments such as this is to emulate the worst-case scenarios that would reasonably occur and the more accurate the lighting representations can be then the more applicable the simulated training is to real world scenarios. Of course, transfer learning can occur and is likely to be the best initial method to transfer to real world applications but the goal would be to be able to capture a network from the simulated scenarios and directly test its applicability to real world scenarios.

## 4.2 Landscape 2: Dynamic Environment and Objects

For the second environment, we are aiming to focus on producing an environment that best focuses on the known dynamic environment issues that SLAM algorithms experience. Essentially, this environments aim is to provide components that move in a variety of patterns in an attempt to confuse the algorithms ability to both localise (as the environment it is localising within is not static) and produce reliable maps (as features are moving not with the drones path and may be either moving linearly, rotating or exponentially accelerating and decelerating). There is no attempt to create difficult specular conditions for this map as was apparent in the previous environment.

The flow of this environment is to slowly introduce an extra layer of difficulty with each ‘room’ that the drone enters. The starting area is relying simply on horizontal and vertical linear displacement at a linear velocity. There is also ample room between moving objects for the drone to stop and wait until the path is clear again.

Moving between the first room to the second room the drone will need to handle a change in path size - shrinking considerably both in height and width to a smaller ‘corridor’ type design. From here the drone will enter the second room. This room is building on what the drone experienced in the first room

however now it is also introducing an artificial change in elevation along with objects that are moving in a linear fashion.

The third room is again more corridor-like where the drone is much more confined for space and must make appropriate use of timing to move through the course. This will be the first room to introduce motion that is not purely horizontal or vertical - the shapes will move in a somewhat circular type of motion within the small space however the velocity will remain linear and consistent.

The fourth room is a short run and doesn't serve as much to test the system as much as it does to provide a path to the next section. This section again uses purely linear motion in a vertical path however the objects are rotated to 45 degrees from the horizontal.

The fifth section introduces linear rotations to the path along with linear movement however the objects will rotate at different velocities to one another. The objects will be central to the path forcing the drone to select a path around the central path of the corridor.

The sixth and final section will present non-linear motion to the drone - objects will move and rotate in a predetermined path but the velocity of the object will not remain constant throughout the path. This is going to be difficult if it is completely randomised so there will be a set point that the object will accelerate to and then a set point for deceleration allowing for it to remain somewhat predictable.

A walkthrough of the environment appears as follows:

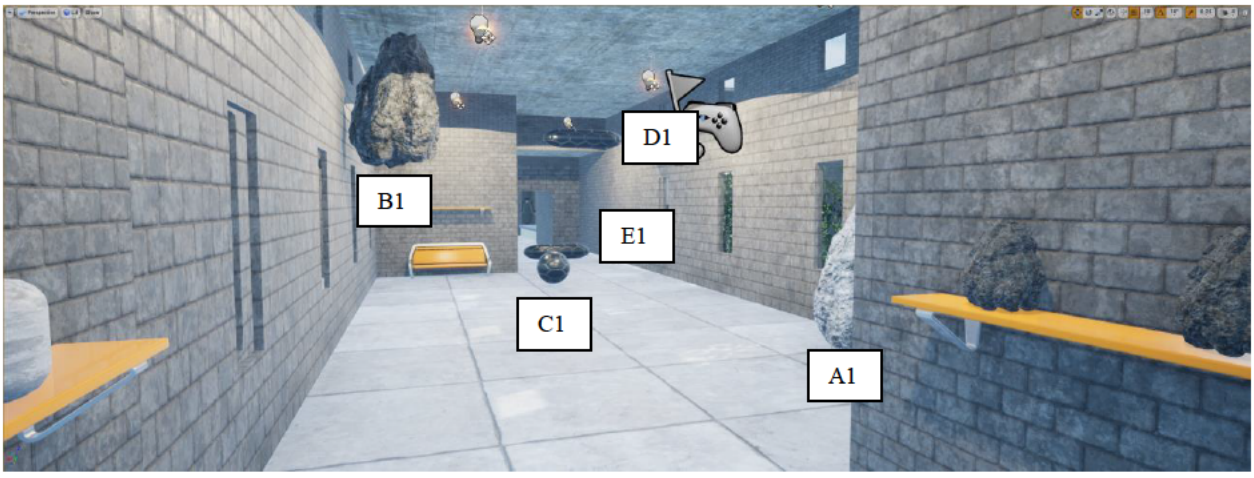



Figure 30. Player starting location and Room 1 - FOCUS: Linear movement

For the labels in Figure 30 above the associated movement information is as follows:

Object A1: Ping pongs horizontally at a constant velocity. No change in height. Duration
Object B1: Ping pongs horizontally at a constant velocity. No change in height.
Object C1: Ping pongs vertically at a constant velocity. No horizontal displacement.
Object D1: Ping pongs vertically at a constant velocity. No horizontal displacement.
Object E1: Ping pongs vertically at a constant velocity. No horizontal displacement.

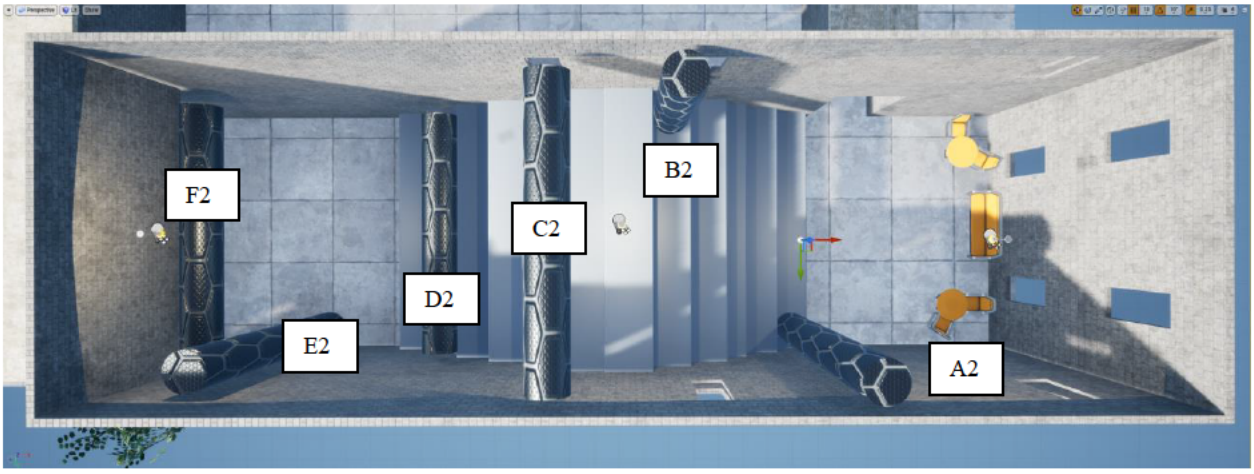



Figure 31. Second Room from corridor entry - FOCUS: Linear movement and elevation change

For the labels in Figure 31 above the associated movement information is as follows:

Object A2: Ping pongs horizontally at a constant velocity. No change in height.
Object B2: Ping pongs horizontally at a constant velocity. No change in height.
Object C2: Ping pongs vertically at a constant velocity. No horizontal displacement.
Object D2: Ping pongs vertically at a constant velocity. No horizontal displacement.
Object E2: Ping pongs horizontally at a constant velocity. No change in height.
Object F2: Ping pongs vertically at a constant velocity. No horizontal displacement.

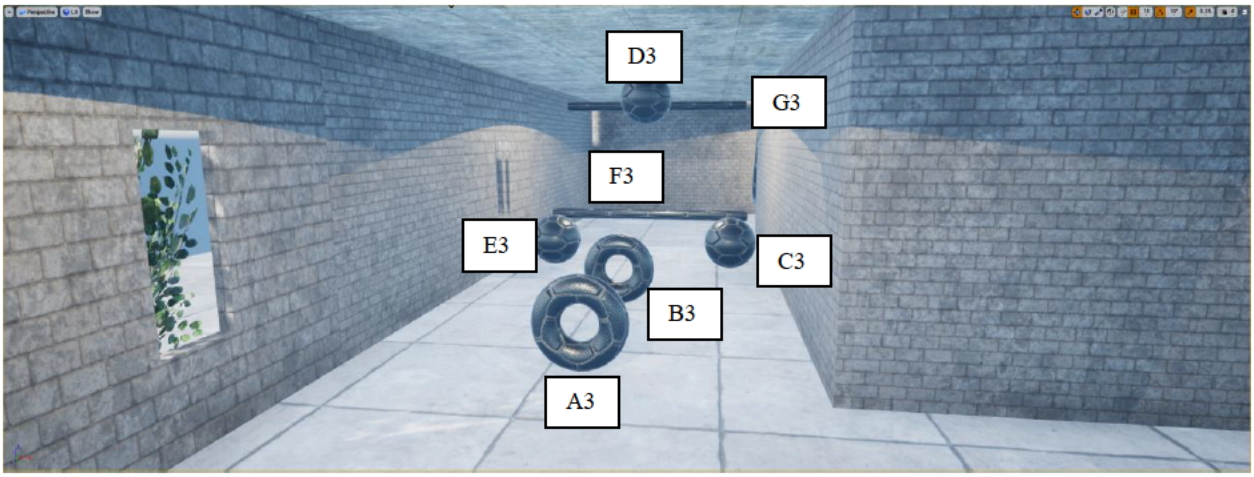



Figure 32. Room three - Focus: Introduce circular paths and maintain linear motion

For the labels in Figure 32 above the associated movement information is as follows:

Object A3: Moves in a figure eight pattern.

Object B3: Moves in a circular path.

Object C3/D3/E3: Moves in a vertical direction with C3/E3 matching movement and D3 opposing.

Object F3/G3: Moves in a vertical direction. No horizontal displacement.

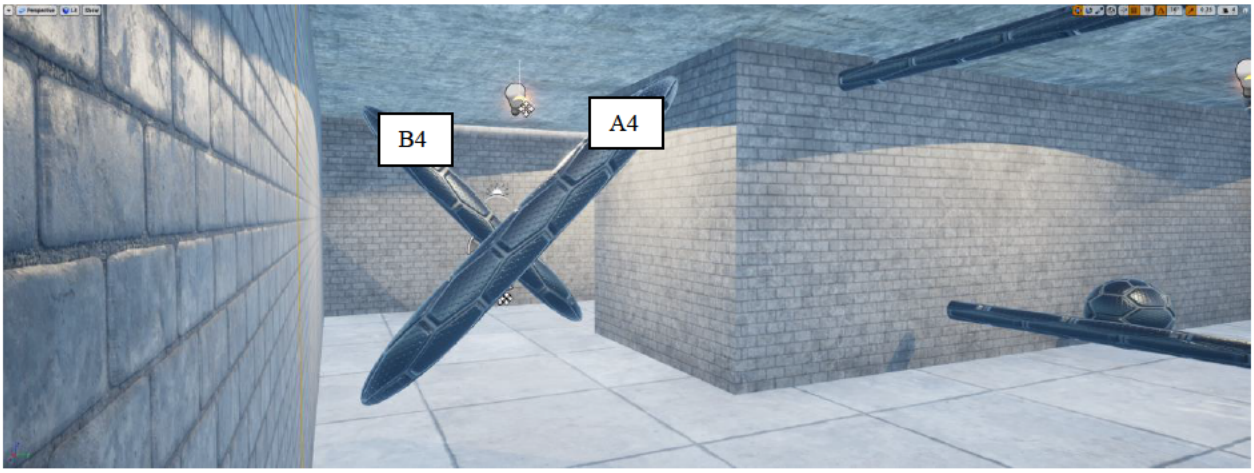



Figure 33. Room four - FOCUS: Linear Motion on 45 degree rotated objects

For the labels in Figure 33 above the associated movement information is as follows:

Object A4: Moves in vertical direction. No horizontal displacement.

Object B4: Moves in vertical direction. No horizontal displacement.

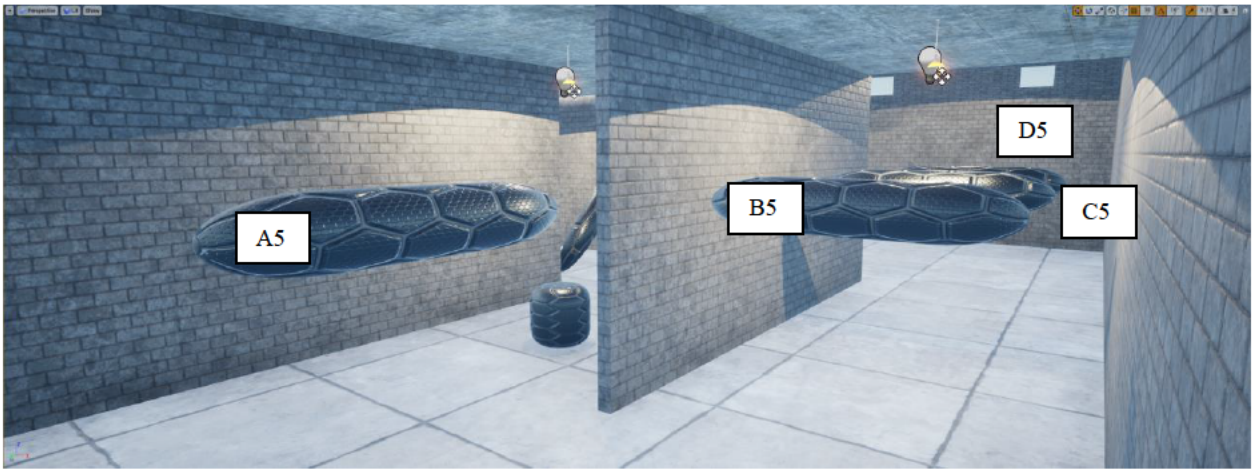



Figure 34. Room Five - FOCUS: Introduction to pure rotational motion with linear speeds

For the labels in Figure 34 above the associated movement information is as follows:

Object A5: Rotates about its centre of mass. No vertical or horizontal movement.
Object B5: Rotates about its centre of mass. No vertical or horizontal movement.
Object C5: Rotates about its centre of mass. No vertical or horizontal movement.
Object D5: Rotates about its centre of mass. No vertical or horizontal movement.

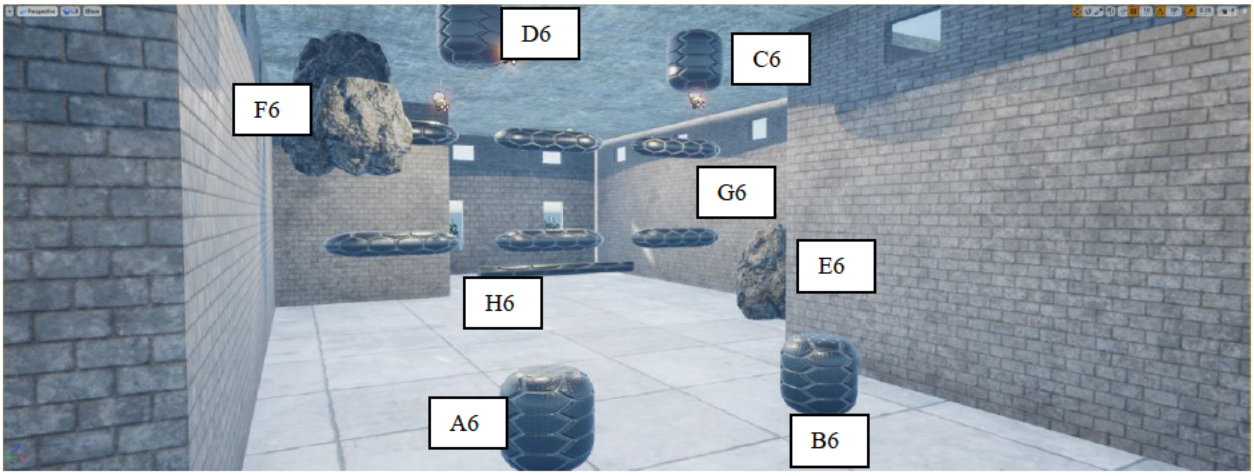



Figure 35. Room Six - FOCUS: Non linear rotation and motion

For the labels in Figure 35 above the associated movement information is as follows:

Object A6: Zigzags, ping pongs vertically.

Object B6: Zigzags, ping pongs vertically.

Object C6: Zigzags in a downwards motion.

Object D6: Zigzags in a downwards motion.

Object E6/F6: Ping pongs about its centre path with a nonlinear acceleration

Object G6: 3x2 rotational array acceleration non-linearly about their centre point

Object H6: Ping pons about its centre path with nonlinear vertical acceleration

The external appearance of the environment and the top down map layout appear as follows. On Figure 37. the top down view - the starting and ending positions are denoted by the letters A (start) and B (end).

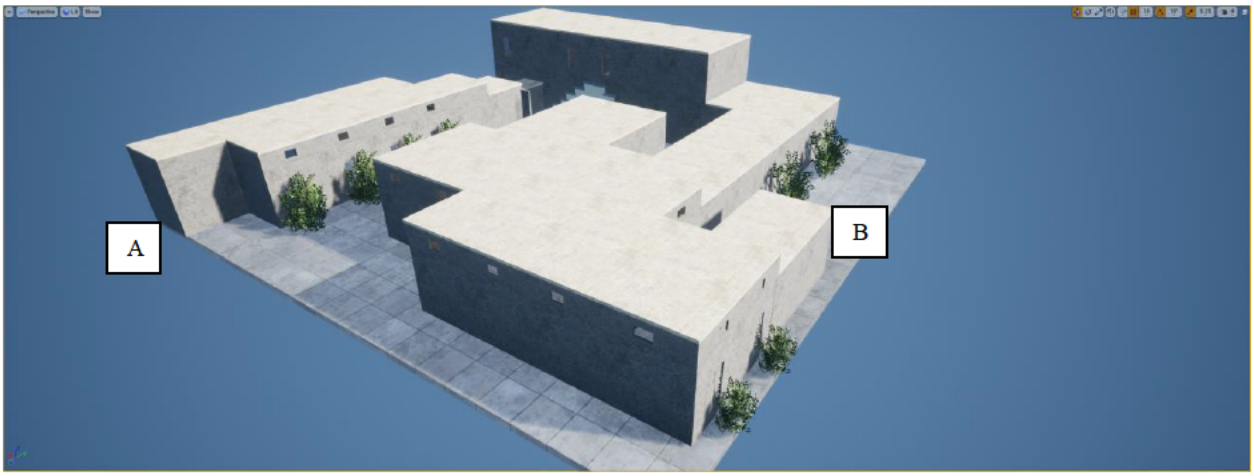



Figure 36. Exterior view of built environment


Figure 37. Top Down View with Roof Removed

There are animation methods for the static meshes utilised in the model of the environment. In general, the preferred method for Unreal Engine to animate a mesh is to create a blueprint class for the mesh and then produce an InterpControlPoint action. This is generally done in a visual flow chart style of programming. For the linear movement of the meshes the following flowchart is used:

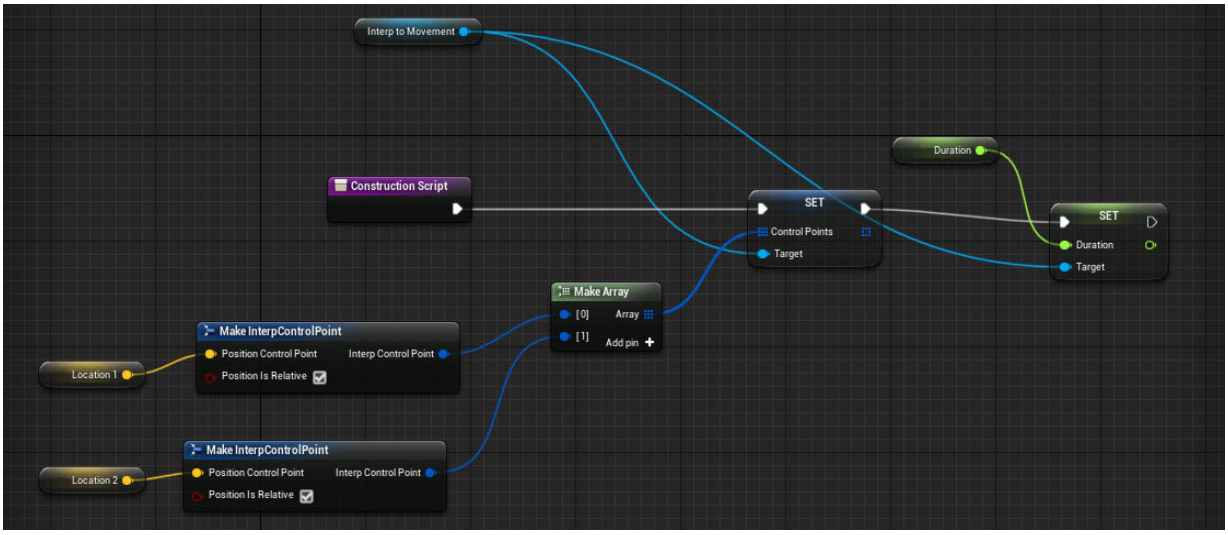



Figure 38. Visual Flowchart Script for Linear Motion

The linear movement script is designed in such a way that it allows the user to copy meshes and paste them around the environment without having to create individual scripts. There are three variables created for the user to alter; Location1, Location2 and Duration. When the user places the object into the environment the two location variables populate an XYZ point in the environment, whereas the duration variable is able to be defined in the 'Details - Default' option box in the PiE. All mesh objects must be set to movable in order to allow the script to run. When altering the script to include additional locations, the Make Array block must contain the number of locations and then the number of location variables must exist to populate the array.

For the nonlinear movement scripts, the design is completely different and requires the use of an animator timeline as opposed to being able to use the construction script path for linear movements. The visual blocks can be seen as follows over the page:

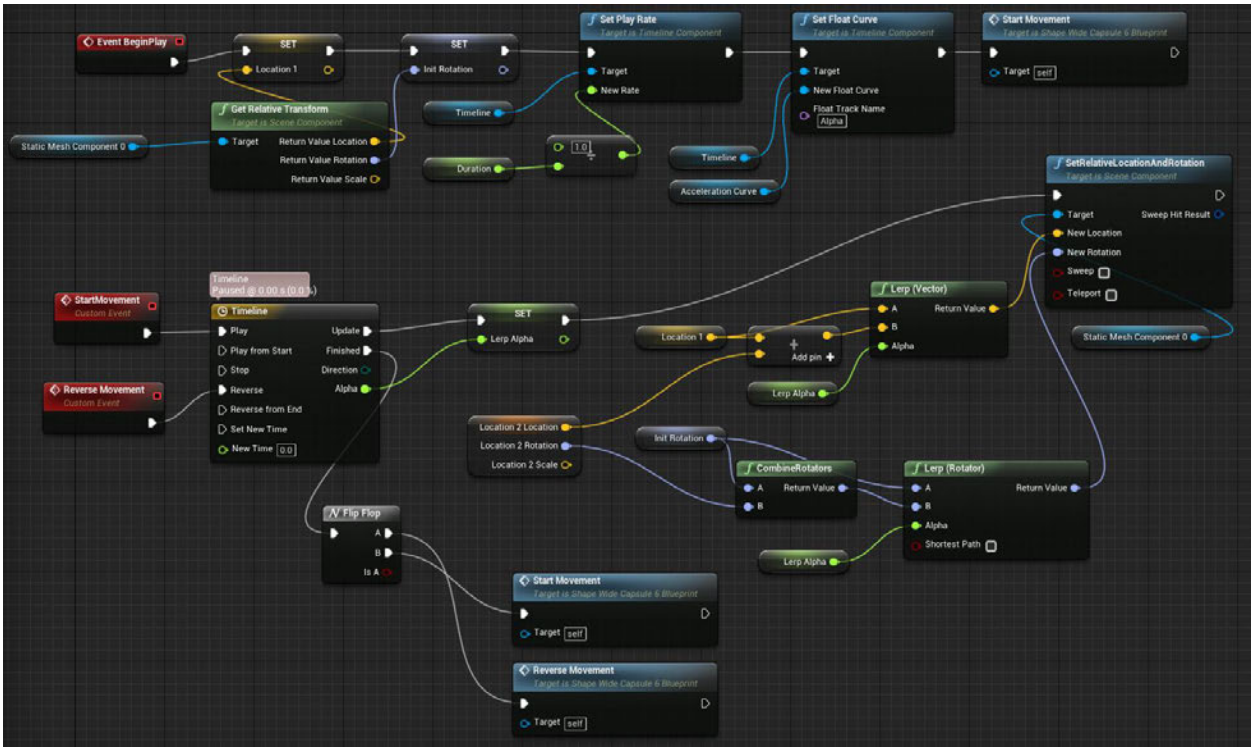



Figure 39. Acceleration Animation Visual Script

The animation visual script relies on three event blocks that are the basis for the script. The starting block is the Event BeginPlay block which serves almost as a header file does for C code. The BeginPlay event captures the initial position and rotation of the object when the play button is clicked and defines the rate at which the animation will execute (within the Timeline animator, the timeline execution path is defined as 1 which indicates that if the play rate is defined at 1, then the vector divide will result in a play rate of 1). Following the play rate defining block is the Set Float Curve block which is where we define the rate at which the path is executed, this essentially means that our acceleration curve is set by this block which is why its float track name is defined at the Alpha output for the Timeline animator. From here we now jump to a Start Movement block which jumps to the StartMovement event. There are actually two blocks that feed into the Timeline animator block, if you look at their connection points, you'll see that it is at the Play and Reverse input nodes. These are not indicators but are event triggers that define the direction that the animation will play. Following on from the Update pin on the Timeline animator the path travels to a Set Lerp Alpha block. This defines the float curve that will be applied to the movement path. Following on from this block the SetRelativeLocationAndRotation block is placed. This block essentially does as the name suggests, it takes an initial position and rotation value and a final position and rotation value and defines the movement path necessary to execute the animation. However, we want the blocks to be modular and not hardcoded, therefore there is a need to create a capture method for the user to modify the starting and ending location in the PiE. This is done using the orange Transformations block which has

had the transform pin split to be a location vector, a location rotation and a scale value. Since the locations are relative to one another (as defined by SetRelativeLocationAndRotation) the two vectors will need to be added together to define Location 2's position in the global space, this is done using a Vector + Vector addition block and then combined into a movement path using the Lerp (Vector) block. Note the Lerp Alpha input is included so that the movement path between the two locations is defined by the AccelerationCurve input created by the user and allows for modular control over the movement type. This is emulated for the rotation aspect of the object however the method that rotator values are combined is using the CombineRotators function block instead of the Vector + Vector addition block. Once the animation has executed, the Timeline animator will look at what is connected to the Finished output pin. Here a Flip Flop block is used such that it will continuously swap between the two on each iteration - this allows the animation to change between forward and reverse motion and ensures a smooth transition occurs.

The float acceleration path is as follows also:

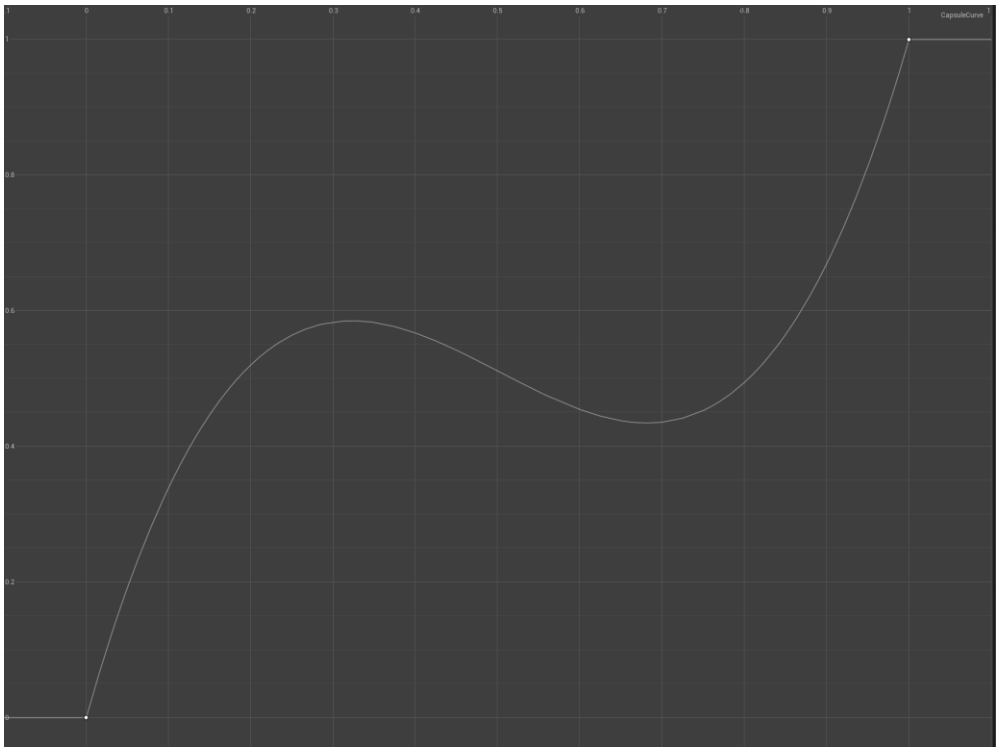



Figure 40. Acceleration Curve labeled CapsuleCurve

The acceleration curve means that the object accelerates to its centre point, rebounds slightly over the centre point, and then continues through its path till the end point as defined by the user. It follows the

curve of a third-degree polynomial with an inflection point about its centre value. In this case the values are between 0 and 1 on both the x and y axis with the inflection point approximated to the coordinate 0.5, 0.5.

## 4.3 Landscape 3: Realistic Complex Environment

The goal with the third landscape was to create an emulation of a somewhat realistic scenario. This scenario would have included factors such as wind, natural moving objects, changes in view distance etc. to emulate a flight through a realistic environment. Although, due to limitations of the compute ability of the computer used for this test, along with the time limitations of the project this section had to be considered null. The results of this section are not directly indicative of any measurable outcome that is not identifiable in Landscape 1 or Landscape 2 and served only as a bridging agent to collate and reinforce any relationships or outcomes determined.

There is the possibility of utilising a pre-built environment from the Unreal Engine Marketplace, however, none of the apparent ‘realistic’ environments met the conditions that were desirable for the project. Thus, this portion of the project had to be forgone to meet timeline requirements, however its necessity for the project's execution was minimal.

## 4.4 Building AirSim

When building AirSim for an individual landscape, I noticed that it was common for a landscape to become corrupted if it was not done correctly. It also occurred occasionally without a cause that I could repeatedly discern and serves as good evidence towards the necessity to retain multiple backups and utilise GIT for local back ups. Therefore, once an environment is finished it is duplicated and the building of AirSim is done on the duplicate version (which is also stored in a separate folder) such that the original copy of the environment remains uncorrupted.

I had many occasions where AirSim would not allow an environment to be rebuilt using the plugins when the environment is entirely custom. The solution is actually to store the environments and the AirSim master files in a disk that is not C:\ as it generally requires admin access to natively run command and batch files from the command line in this directory and can be disrupted when attempting to call them within the PiE.

Therefore, a simplified process of building AirSim for a modified environment or an environment built by the individual is as follows:

- Pull AirSim from the Git Repository
- Move files into D:| directory
- Run build.cmd

From here, the files will be available on the system to run. Also, the example environment ‘blocks’ is available to test out your system. However, it is not yet ready to run on the desired environment (i.e. the custom environments built). To get the plug-in ready for use in the custom environment, the plugin files from the AirSim master files need to be copied into the working directory of the custom environment. Then, the project file for the environment needs to be customised such that is appears similar to the following:

```
{
  "FileVersion": 3,
  "EngineAssociation": "4.24",
  "Category": "",
  "Description": "",
  "Modules": [
    {
      "Name": "",
      "Type": "Runtime",
      "LoadingPhase": "Default",
      "AdditionalDependencies": [
        "AirSim"
      ]
    }
  ],
  "Plugins": [
    {
      "Name": "AirSim",
      "Enabled": true
    }
  ]
}
```

Where sections such as Modules - Name are filled in with the desired project file. The user must then run the clean.bat file to ensure intermediate files are correct and then proceed to run GenerateProjectFiles.bat to generate the necessary Visual Studio file in the format of .sln as this is the file that the actual AirSim simulations tool operates within. Then, opening the file within Visual Studio the user is to select the DebugGame Editor option along with the Win64 build configuration prior to running the program. There are steps not present in the official AirSim guide (as far as I was able to find) which are necessary to

getting the package to work on custom environments. The user will need to set the environment they are working on as the startup project for Visual Studio otherwise they risk the system attempting to open a file that is not existent (it will look in the original C:\ for the UE4 files however it will be unable to find any files or run them if it does due to admin issues). After this the user needs to go to World Settings and change GameMode Override to AirSimGameMode and then when the user attempts to play a level it will automatically populate the PlayerStart object with the desired vehicle (it will prompt the user for either a quadrotor/drone or a car if the user has not specified the vehicle in the settings.json file).

In order to achieve the desired output from the AirSim plugin the settings.json file needs to be updated to include alterations to the standard values. The settings.json code used for all environments is as follows:

```
{
 "SeeDocsAt": "https://github.com/Microsoft/AirSim/blob/master/docs/settings_json md",
 "SettingsVersion": 1.2,
 "SimMode": "Multirotor",
 "ClockSpeed": 1.0,
 "ViewMode" : "Fpv",
 "Recording": {
  "RecordOnMove": false,
  "RecordInterval": 0.05,
  "Cameras": [
     { "CameraName": "0", "ImageType": 0, "PixelsAsFloat": false, "Compress": true }
  ]
 },
 "CameraDefaults": {
  "CaptureSettings": [
   {
    "ImageType": 0,
    "Width": 480,
    "Height": 480,
    "FOV_Degrees": 90,
    "AutoExposureSpeed": 100,
    "AutoExposureBias": 0,
    "AutoExposureMaxBrightness": 0.64,
    "AutoExposureMinBrightness": 0.03,
    "MotionBlurAmount": 0,
    "TargetGamma": 1.0,
    "ProjectionMode": "",
    "OrthoWidth": 5.12
   }
  ]
 },
 "Vehicles": {
  "SimpleFlight": {
   "VehicleType": "SimpleFlight",
   "DefaultVehicleState": "Armed",
   "EnableCollisionPassthrough": false,
   "EnableCollisions": true,
   "AllowAPIAlways": true,
```

```
      "RC": {
        "RemoteControlID": 0,
        "AllowAPIWhenDisconnected": false
      }
    }
  }
}
```

The most notable feature in the above code is that all values are left mostly default, however the camera resolution is changed from 256x144 to 480x480 with the images being fed at 20 frames per second if the system can render at that FPS. Also, to help with the collection of training data, the viewmode for the drone is set to first person view (FPV) so that the pilot's view is a replica of the onboard camera. Additionally, further settings are explained in Chapter 5.

# Chapter 5 - Development & Training of DNN

The development and training of the DNN used for this project is an implementation of a Deep Convolutional Network as has been mentioned throughout the paper. This section aims to divulge the process and decision-making process coinciding with the development of the algorithm for the project.

When looking at the design and development of a Convolutional Neural Network there are general accepted criteria that substantiates the network structure being labelled a CNN. As outlined in Chapter 1, a CNN requires convolutional and pooling layers as its essential underpinnings and a regressor is generally applied to force the network into a decision-making process towards identifying the output classes. The process of developing this will be discussed throughout the chapter.

There is a 'marketplace' for prebuilt CNNs which can be accessed via the PyTorch library by loading a torchvision.model library. For instance, one of the most prominent architectures, AlexNet, can be downloaded and prepared for training by using the following code:

```
Import torchvision as models
Alexnet = models.alexnet(num_classes=n)
```

What this does is it preloads a built architecture and defines the regressor state to have an output of *n* for the number of classes desired to be identified by the network. AlexNet is a quite simple (but revolutionary for its time - 2012) architecture and serves as a common starting point for most simple architecture design in modern systems. Its main identifying features is that it introduced the concept of MaxPool and Dropout and made use of the ReLU activation function which wasn't overly common at the time of its inception (Pointer 2019).

The design architecture for the CNN utilised in this project will be very similar to AlexNet and somewhat approximates the architecture style outlined by VGG-16 which implements a series of 2 to 3 convolutional layers followed by a pooling aggregate layer (Pointer 2019).

VGG-16 can be seen as follows:

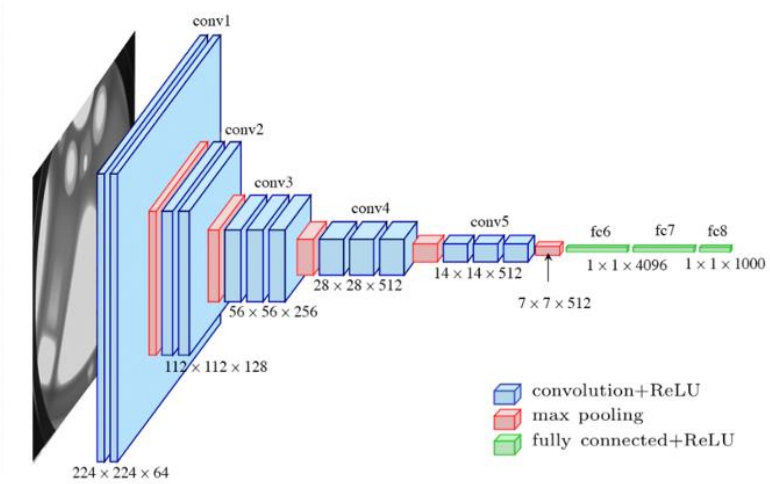



Figure 41. VGG-16 Network Architecture (Khandelwal 2017)

The architecture design for this project went through several iterations, in which it was simplified in its design and simplified in the way it was programmed. Initially, it was very hard coded and was absolvent of the class inheritance and ease of design that was mentioned in Chapter 1 for the utilisation of PyTorch. It was still using PyTorch, but it treated the problem and was almost complex for the sake of complexity. Which is never a good thing. The architecture for this project utilises 6 individual convolutional layers and 3 fully connected layers leading to the output classes. Looking back onto Figure 41. it can be seen that this interprets the architecture of VGG-16 from conv4 through to fc8 however it is fundamentally different in its size per convolutional layer and filter kernel size producing a vastly different network output. The similarities to a proven high performing architecture provides confidence in the networks ability to perform its intended task.

## 5.1 AirSim API & Defining Drone Control

In order to appropriately utilise the Deep Neural Network, there needs to be an appropriate method to not only collect relevant data for training purposes but also a method to parse images from the “camera” in the simulated environment to the active algorithm. AirSim provides an API to connect external algorithms to the environment, written in both C++ and Python.

The example connector API for controlling a drone in Python is shown below:

```
# ready to run example: PythonClient/multirotor/hello_drone.py
import airsim

# connect to the AirSim simulator
client = airsim.MultirotorClient()
client.confirmConnection()
client.enableApiControl(True)
client.armDisarm(True)

# Async methods returns Future. Call join() to wait for task to complete.
client.takeoffAsync().join()
client.moveToPositionAsync(-10, 10, -10, 5).join()

# take images
responses = client.simGetImages([
    airsim.ImageRequest("0", airsim.ImageType.DepthVis),
    airsim.ImageRequest("1", airsim.ImageType.DepthPlanner, True)])
print('Retrieved images: %d', len(responses))

# do something with the images
for response in responses:
    if response.pixels_as_float:
        print("Type %d, size %d" % (response.image_type, len(response.image_data_float)))
        airsim.write_pfm(os.path.normpath('/temp/py1.pfm'), airsim.getPfmArray(response))
    else:
        print("Type %d, size %d" % (response.image_type, len(response.image_data_uint8)))
        airsim.write_file(os.path.normpath('/temp/py1.png'), response.image_data_uint8)
```

(AirSim 2018)

This will be manipulated to be operational for the implementation of the DNN. When considering this API, it provides access to a few of the most important operations that are required to control the drone; access to the live image, access to the drone location, well defined start locations and simulated control inputs in the form of manipulation of the 6 axis inputs.

The recommended method to move the quadcopter using the API is to utilise the command:

```
client.moveByVelocityAsync(vel_x, vel_y, vel_z, duration),join()
```

(AirSim 2018)

However, it must be noted that this inadvertently ignores the use of pitch, yaw and roll mechanics available for the drone. But, how important is it to use these mechanics? Well, using this method the yaw, pitch and roll are automatically inferred in order to move using these mechanics. There is also the command:

```
client.moveByVelocityZAsync(vel_x, vel_y, z, duration),join()
```

(AirSim 2018)

Where again the control is placed into a medium level instance except here the altitude of the drone can be set to remain constant, leaving only the x and y velocity values to be determined. This would be useful if there were no apparent elevation variations between the levels. However, Landscape 2 utilises elevation and as such this method can't be a viable option. This appears to be a command more in line with vehicle control scenarios such as cars and trucks however the API control for those is different to the quadrotor API.

In order to describe to the API how to move the drone, a setting known as the Drivetrain needs to be defined. By setting the drivetrain to always be forward the system will keep the camera facing forward at all times, if it is set to MaxDegreesofFreedom then the algorithm will need to indicate to the drone when to rotate such that the camera aligns with its movement path. To do this, the drivetrain parameter is set to airsim.DrivetrainType.ForwardOnly

The selected method of control will be:

```
client.moveByVelocityAsync(vel_x, vel_y, vel_z, duration),join()
```

(AirSim 2018)

In which the output classes can be simplified down from the 6 classes previously identified for the control of each degree of freedom down to only 3 classes necessary. The output classes for the network can be defined then as: (1) vel_x, (2) vel_y, (3) vel_x. However, if the output is passed through a ReLU gate then the values presented will struggle to operate as the ReLU value is a figure between 0 and 1. Therefore this would indicate that it is necessary to use six classes in which the output classes result in a summed value between the vel_x positive and vel_x negative values to indicate the vel_x value required. Alternatively, the user could indicate a Tanh function and utilises the percent positive or negative for the associated class as its control measure. The appropriate decision here is to utilise the ReLU function and clearly define which option to utilise for the objective output. This is solely due to being simpler in training and deployment as it can be specifically indicated in the training data whether a positive or negative value is required for the velocity of an axis.

## 5.2 Image Collection and Input Into Neural Network

The AirSim plugin provides a variety of methods for the user to capture training data. Within a built environment, such as the standard BLOCKS environment, it is possible to fly the drone manually and capture data utilising the record button located on the standard interface. This will capture data from a

forward-facing virtual camera located on the drones body. As standard, this data is relatively low quality, stored in a resolution of 256 x 144 pixels in the PNG format.

Considering that for the project we are aiming to improve the amount of data captured as to improve the ability of the network to abstract and infer from its presented information, the input resolution will be increased to 480x480 pixels. The colour input bit depth will be left standard and the output format will remain to be PNG as it is effective for the end goal of the project. However, the image will be transformed when importing it into the network.

In order to alter the setting of the built-in data collection method within the AirSim plugin, there is a settings.JSON file which can be manipulated as the user wishes. However, the code settings for the file can be found in Chapter 4.

As standard, the simulator will elect to use the multirotor option (quadrotor drone) and the settings file will appear as seen below with no indication of any non-default settings:

```
{
        "SeeDocsAt": "https://github.com/Microsoft/AirSim/blob/master/docs/settings.md"
        "SettingsVersion": 1.2
}
```

It is possible for the user to alter the location of the camera when operating in CV mode using a variety of different Python API calls. These are as follows: front_center, front_left, front_right, bottom_center, back_center. The standard location is front_center and this will remain the same for the implementation in this project. As it does not need to be altered, it is not included as the default settings are automatically populated.

In order to capture accurate paths, it is possible to place the system into a Computer Vision mode where the physics engine is disabled and only a camera object exists. Here, you can clearly define paths or manually control the object to collect controlled training data. Whilst this is a viable method, the inability to capture physics related information - such as IMU data - limits its capability to be utilised for this project. Additionally, an IMU is not used in the engine as it would natively work in the 'real world' as it provides real world location data and elevations. Realistically, for the algorithms to be applicable there is a need to use actual movement data and this can be collected but only in a different form. This was included in the most recent update to AirSim (v1.3.1) which included native IMU support for the Python API and would have been beneficial to include had it been included at the time of commencement.

When parsing images to the DNN for operation, utilising the prior mentioned Image API in Python allows complete control over the input of the image. It is possible to use the library and push the images to an array using the Numpy library, taking advantage of Numpy's compute benefits. The example provided codes are as follows:

Standard:

```
import airsim #pip install airsim
# for car use CarClient()
client = airsim.MultirotorClient()
responses = client.simGetImages([
    # png format
    airsim.ImageRequest(0, airsim.ImageType.Scene),
    # uncompressed RGB array bytes
    airsim.ImageRequest(1, airsim.ImageType.Scene, False, False),
    # floating point uncompressed image
    airsim.ImageRequest(1, airsim.ImageType.DepthPlanner, True)])
```

(AirSim 2018)

Numpy:

```
responses = client.simGetImages([airsim.ImageRequest("0", airsim.ImageType.Scene, False, False)])
response = responses[0]
# get numpy array
img1d = np.fromstring(response.image_data_uint8, dtype=np.uint8)
# reshape array to 4 channel image array H X W X 4
img_rgb = img1d.reshape(response.height, response.width, 3)
# original image is flipped vertically
img_rgb = np flipud(img_rgb)
# write to png
airsim.write_png(os.path normpath(filename + '.png'), img_rgb)
```

(AirSim 2018)

We can see that the example codes above are extremely similar to the example drone control code using the API shown prior. The client agent is created in order to interact with the AirSim environment in the form of the quadrotor drone i.e. client.Multirotorclient(). Using client.simGetImages() the 'live' feed of the environment can be fed into the algorithm. However, the API needs to understand what type of image to provide. The documentation for AirSim defines the possible ImageType values as being:

```
Scene = 0,
DepthPlanner = 1,
DepthPerspective = 2,
DepthVis = 3,
DisparityNormalized = 4,
Segmentation = 5,
SurfaceNormals = 6,
Infrared = 7
```

(AirSim 2018)

For this application, only the Scene (ImageType = 0) is required. This is the closest 1:1 interpretation of the environment as a standard RGB camera would see it. When bringing an image into a neural network for analysis, there is a need to convert the data into a usable value. We can represent the image as a segment of numbers indicative of the values in that pixel. However, let's first assess pushing the image to the algorithm, and then assessing how it is normalised for use in the network.

The input layer can be considered to be a one dimensional representation of the input image with a total number of nodes indicative of the total number of pixels in the image i.e. 256 x 144 pixel image would result in an input layer of 36,864 not including IMU data or any other input information. The input resolution for the network is 480 x 480 resulting in an input layer of 230,400 nodes not including IMU data or any additional input values. The problem is however, is that this is the case for a singular input layer i.e. a one channel input. A one channel input is a grayscale image. If a three-channel input was to be used then an RGB image is being fed into the network. Therefore, a tuple for each cell is provided that is a representation of the amount of Red, Green and Blue channel that is active for an 8-bit colour scale i.e. 0 - 255. This means that for an RGB image to be fed into the network will have a simulated 3 x 480 x 480 input layer resulting in just under 700,000 nodes at the input level. This is a large number of active nodes in the network.

The normalisation of a layer is the process of modifying the input value of each cell to a discrete range - generally between -1 and 1 or between 0 and 1 it is dependent on the activation function being used. In PyTorch the following code is used to normalise an image for processing:

```
img_transforms = transforms.Compose([
    transforms.Resize((480,480)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225] )])
```

(Pointer 2019)

This code is implemented from the exemplar code provided by the book, ‘Programming PyTorch for Deep Learning’ by Ian Pointer. In a normal scenario, the camera settings of the camera to be used would define the normalisation values. However, in this condition the camera is virtualised and as such the generalised normalisation values provided above are to be used. It is noted in the book that even though

the values will produce minimal amounts of error, it is often still used as the errors are not significant enough to warrant alteration (Pointer 2019).

## 5.3 Convolutional, Pooling & Normalisation Layers

The major benefit to the use of a package such as PyTorch is the simplicity with which generalised networks can be built. For our condition, we mostly do not care about minute alterations to the network and are more concerned with its general production and training. Using the PyTorch library we are able to construct layers and define their connections relatively easily.

> torch nn.Conv2d(in_channels: int, out_channels: int, kernel_size: Union[T, Tuple[T, T]], stride: Union[T, Tuple[T, T]] = 1, padding: Union[T, Tuple[T, T]] = 0, dilation: Union[T, Tuple[T, T]] = 1, groups: int = 1, bias: bool = True, padding_mode: str = 'zeros')

(PyTorch 2019)

The shape and size of each convolutional layer is defined (within PyTorch) by the following equations. If the input shape is defined by (N, $C_{in}$, $H_{in}$, $W_{in}$) and (N, $C_{out}$, $H_{out}$, $W_{out}$) then we get the following:

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times padding[0] - dilation[0] \times (kernel\_size[0] - 1) - 1}{stride[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times padding[1] - dilation[1] \times (kernel\_size[1] - 1) - 1}{stride[1]} + 1 \right\rfloor$$

(PyTorch 2019)

Where N is batch size, C is the number of channels, H is the height of input plane in pixels and W is the width in pixels. (PyTorch 2019)

For the implementation in this project it is only necessary to define the in_channels, out_channels, kernel_size, stride and padding values. Therefore, the code utilised for the initial convolutional layer is as follows:

```
nn.Conv2d(3, 32, kernel_size=10, stride=10, padding=0)
```

Where the convolutional output values are determined by the equation:

$$out(N_i, C_{outj}) = bias(C_{outj}) + \sum_{k=0}^{C_{in}-1} weight(C_{outj}, k) * input(N_i, k)$$

(PyTorch 2019)

Where the * function is a 2D cross-correlation operator.

However, following the convolutional layer there is a need to normalise the incoming data prior to being fed through the ReLU activation function. During research for this project it was found that there is significant discourse surrounding whether to utilise normalisation before or after the ReLU function, however due to significant evidence in AlexNet and similar networks towards the use of normalisation prior to the ReLU then this shall be followed for this project.

Normalisation is applied to the network in a similar fashion to the creation of a layer. It takes a filter of the size of the incoming layers, identifies the number of features (input channels) and then conducts normalisation. In a condition where there are multiple incoming features from an image-based network it is necessary to utilise Batch Normalisation in the 2D plane.

This is implemented by the function BatchNorm2d by PyTorch and is described in the documentation as being implemented by the following:

torch.nn.BatchNorm2d(*num_features*, *eps=1e-05*, *momentum=0.1*, *affine=True*, *track_running_stats=True*)

(PyTorch 2019)

Batch Normalisation is governed by the following equation:

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} \times \gamma + \beta$$

(PyTorch 2019)

Where the mean and standard deviation are calculated per dimension. The factors $\gamma$ and $\beta$ are learnable parameter vectors of the size of the input with $\gamma$ initialised to 1 in its values and $\beta$ initialised with its values at 0 when beginning the process.

It is implemented in the project using the following line for the first layer as follows:

```
nn.BatchNorm2d(32)
```

From here, the layers now pass through a ReLU function. A ReLU is fairly simple in implementation and design. It is a positively linear algorithm ranging from 0 to N and follows a plot as shown in the documentation for PyTorch:

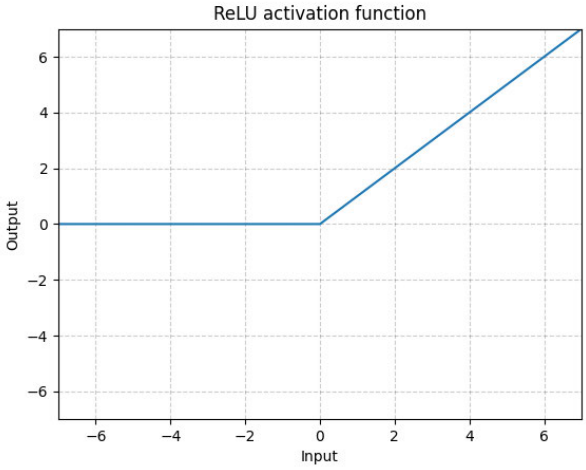



Figure 42. ReLU Output Function (PyTorch 2019)

The ReLU activation utilises the following equation:

$$ReLU(x) = (x)^{+} = \max(0, x)$$

(PyTorch 2019)

This equation is provided by PyTorch in the documentation and simply defines that the output is equivalent to zero until the value exceeds the gate value in which is will output value x linearly increasing as the input value increases.

For the actual network used, there are still another 2 more convolutions prior to reaching the pooling layer, however it can be described here. The pooling can be implemented in 2 methods; MaxPool2d and AdaptiveAvgPool2d. In the convolutional portion of the network, MaxPool2d is utilised and prior to moving into the regressor the AdaptiveAvgPool2d is utilised.

MaxPool2d is utilised in the hidden layers as it is arguably faster in implementation than the AvgPool2d function (Pointer 2019). It is simple when considering it in the terms of its use, depending on the filter size and stride it will take the maximum value of each matrix from its movement across the data. It will output a value based on the following equation:

$$out(N_i, C_j, h, w) = max_{m=0,\ldots,kH-1} max_{n=0,\ldots,kw-1} input\,(N_i, C_j, stride[0] \times h + m, stride[1] \times w + n$$

(PyTorch 2019)

Where kH and kW is the kernel dimensions in height and width. It is implemented using the following code:

> torch.nn.MaxPool2d(*kernel_size: Union[T, Tuple[T, ...]], stride: Optional[Union[T, Tuple[T, ...]]] = None, padding: Union[T, Tuple[T, ...]] = 0, dilation: Union[T, Tuple[T, ...]] = 1, return_indices: bool = False, ceil_mode: bool = False*)
>
> (PyTorch 2019)

However, only the kernel_size and stride values are defined for each hidden layer. It is implemented in its first position as the following:

```
nn.MaxPool2d(kernel_size=3, stride=2)
```

For the final connector to the regressor layers, the AdaptiveAgvPool2d is utilised as it takes an arguably more data inclusive snapshot of the layers and is exclusive of influence for overly maximum values skewing features within the layers. It is implemented via the following code:

> torch.nn.AdaptiveAvgPool2d(*output_size: Union[T, Tuple[T, ...]]*)
>
> (PyTorch 2019)

This takes the average value from each kernel output and constructs a new matrix utilising these values. It is implemented in the code as follows:

```
self.avgpool = nn.AdaptiveAvgPool2d((6,6))
```

Utilising the self-class identifier as it sits as a separate layer compared to the hidden layers which are stored as a features class. From here, the network will identify one of the output nodes with a class in the training data and the association occurs.

## 5.4 Output Values & Fully Connected Final Layers

When looking at the output of the network, it is difficult to properly visualise what the output needs to be. But, if the scenario is broken down it is easy to identify the necessary controls. Referring back to the drone model shown in Section 1.3.1 we can see that a drone has 6 degrees of freedom (x,y,z,pitch,yaw,roll) and therefore we can structure the output of the model around this idea. However, the API control of the system limits exactly how we can interact with the simulations environment.

Remembering back to Section 5.1 it was found that the general recommended control surrounds the manipulation of each axial velocity with the assumed rotations being handled by the AirSim API.

Stepping away from these limitations for the moment, the output structure of the network can be interpreted now as a set of classes aligning with an input to one of the degrees of freedom within which the drone can move. Therefore, the structure in a simple sense would be as follows:

- Input layers and Image Manipulation
- Hidden layer compute (Convolutions, Pooling, Batch Normalisation)
- AveragePooling Aggregate Layer
- Fully connected output layers

The fully connected output layer will have six classes for each of the six degrees of freedom. However, the network itself will not directly interact with the drone. The program will need to take the output value and the indicated class that the network has indicated needs to be active and transform it into a usable value for control. This can be converted into a scalar output value that falls in line with the AirSim plugin standard of providing a unitless scalar input value to the motors for control of the drone.

However, focussing on the building of the fully connected linear layers (this has been referenced to as the regressor throughout the paper) utilises the following code:

torch.nn.Linear(*in_features: int*, *out_features: int*, *bias: bool = True*)

(PyTorch 2019)

Which is fairly simple in its implementation, however in this function the in_features and out_features do not refer to a number of layer, but a number of individual nodes within that layer. These layers close out to a defined number of nodes that are identified as being the classes – noted in the code as *num_classes* which align to an indication of that velocity value for each axis.

The indication percentage of each class is then multiplied by a value such that its velocity input makes sense for the control scenario. It was unsuccessful in attempting to validate the value, but the control scenario was defined out of a value of 10 meaning the maximum output velocity was 10m/s in either direction however this value was summed for a combined output value for each axis velocity.

## 5.5 Selection of training data and training

As was noted in the literature review, the use case of spectral normalisation for training networks is highly important. When looking at the formal documentation for PyTorch we can see that spectral normalisation is built into the package. The documentation outlines that it is implemented using the following:

```
torch nn.utils.spectral_norm(module, name='weight', n_power_iterations=1, eps=1e-12, dim=None)
```

(PyTorch 2019)

Normally the use of spectral normalisation is utilised when developing General Adversarial Networks, however the paper in the literature reviewed clearly outlined its usefulness in training CNN systems for stability.

Applying spectral normalisation utilises the following equation:

$$W_{SN} = \frac{W}{\sigma(W)}, \sigma(W) = max_{h:h\neq 0} \frac{\|Wh\|_2}{\|h\|_2}$$

(PyTorch 2019)

Where its implementation focusses on the weight matrices that are built for the network and rebalancing their values with respect to the standard deviation of their distribution. However, the method of utilising spectral normalisation doesn't currently work with the method of training being implemented and as such will need to be elected as future work.

Moving on from the importance of balancing the network, we can investigate the data that is being fed into the network in order to train it. Now, it must be understood that it is not necessary to feed a complete live view of the environment into the network in order for it to operate. In fact, this would end up producing a negative outcome for a standard CNN. This would most likely result in 'overfitting' where the network optimises too strongly for a certain characteristic too early within the training batches. Theoretically, a network can recover from this however it is not typically seen in actual models and therefore it is better to avoid the issue at the source.

In an attempt to help prevent overfitting of the model, data is selected and shuffled as the network is trained. Realistically, we can treat the network as being agnostic to the reality of the data that it is being fed. What this means is that we do not need to feed the data to the network in the order within which it was captured. Additionally, not every instance of data is required. Frames that are extremely similar, or

data that can be considered essentially replica of another frame of data is to be ignored. This too will result in overfitting. Essentially, the network reinforces a specific task rather than developing a generalised understanding of the data and how to respond to it. If it was desired to develop an understanding of the sequency of data, then an RNN-CNN would be utilised.

Luckily, there is a technique built into PyTorch's training capabilities that allows for this to happen easily. The package allows you to split the training into batches, with each batch's data being able to be a shuffled selection from a data pool. This is done using DataLoader which is built into PyTorch, since the data being imported is a series of images, the torchvision DataLoader will need to be utilised. The example code provided in the documentation for this looks as follows:

```
imagenet_data = torchvision.dataset.ImageNet('path/to/imagenet_root/')
data_loader = torch.utils.data.DataLoader(imagenet_data, batch_size=4, shuffle=True,
num_workers=args.nThreads)
```

(PyTorch 2019)

To break this down simply, the data set is being identified with its stored location on the computer and then the data is imported into the data_loader in 4 batches which the data shuffled and we tell the system to utilise multiple threads when doing this.

The training algorithm for the network utilises an Adam optimisation as opposed to a Stochastic Gradient Descent. As is found in the literature surrounding the development of CNN's by both Ian Point and Grasser and Keng it is somewhat established that the utilisation of Adam presents faster convergences when assessing large amounts of data with multiple output classes. The loss is also defined as being CrossEntropyLoss simply because it is utilised for conditions where there is not a binary output (where BinaryCrossEntropyLoss would be implemented) and shall suffice for the training and optimisation of the network.

A sample of the training data can be found in Appendix J. When assessing the training data collected for the project, the use of the KITTI training database was not considered viable as the conditions expressed in the dataset did not closely align with the use case of the network for this project. Its use would have further convoluted the training process as transfer learning would have needed to have been applied in order for it to be useable. However, the labels and objectives of the data do not match that of this project and as such it was elected not to use the database.

Data in this project is stored such that it can be brought into the network as a class label. In the storage directory for the images, each image is stored as a class denoted by its folder name. It is possible to label images with multiple classes, however this was determined unnecessary for the project and was likely to adversely affect the training of the network. The storage labels are aligned with the output class names mentioned prior in the chapter. Therefore, the class names are; vel_x_pos, vel_x_neg, vel_y_pos, vel_y_neg, vel_z_pos, and vel_z_neg.

## 5.6 Moving Neural Networks Between Development, Training & Production

The code in its development phase is emulated in the training phase as they are essentially one in the same. The codes structure, parameters and all additional information can be stored within the operating directory as a pickle file (known as saving the state DICT). It can be saved using the following code:

```
torch.save(model.state_dict(), PATH)
```

(PyTorch 2019)

And loaded again using the following code as shown within the PyTorch documentation:

```
model = TheModelClass(*args, **kwargs)
model.load_state_dict(torch.load(PATH))
model.eval()
```

(PyTorch 2019)

Now, the model can have its output states assigned to control values in the production code and it becomes the production version of the model. It is possible to do this within one script and indicate operation based on selecting an operating mode, however it is easier to debug the code when the training and production variants are kept separate from one another (from experience in the project).

## 5.7 Discussion of Completed Network Structure

The structure of the network is as follows, it can be found by inputting cnnNet() into the Spyder Terminal after building the network:

Training:

```
cnnNet(
  (features): Sequential(
    (0): Conv2d(3, 32, kernel_size=(10, 10), stride=(10, 10))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
    (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
    (7): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU()
    (9): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (11): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): ReLU()
    (13): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
    (14): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (15): ReLU()
    (16): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1))
    (17): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (18): ReLU()
    (19): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.2, inplace=False)
    (1): Linear(in_features=4608, out_features=1152, bias=True)
    (2): ReLU()
    (3): Dropout(p=0.2, inplace=False)
    (4): Linear(in_features=1152, out_features=1152, bias=True)
    (5): ReLU()
    (6): Linear(in_features=1152, out_features=6, bias=True)
  )
)
```

Production:

```
cnnNet(
  (features): Sequential(
    (0): Conv2d(3, 32, kernel_size=(10, 10), stride=(10, 10))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
    (4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU()
    (6): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
    (7): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU()
    (9): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (11): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): ReLU()
    (13): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
    (14): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (15): ReLU()
    (16): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1))
    (17): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (18): ReLU()
    (19): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (1): Linear(in_features=4608, out_features=1152, bias=True)
    (2): ReLU()
    (4): Linear(in_features=1152, out_features=1152, bias=True)
    (5): ReLU()
    (6): Linear(in_features=1152, out_features=6, bias=True)
  )
)
```

The network structure closely approximates structures such as AlexNet and VGG-16 and can be considered to be a hybridised model based on these two model structures.

The Dropout layers are not present in the production version of the network. This is as it is not desirable to remove data when it is being utilised for control as a Dropout layer in this instance reduces pixel data by up to 20% (denoted by 0.2) during training. However, it is desirable during training to help reduce overfitting and to improve the rate at which the network converges (Pointer 2019).
A large kernel is utilised initially with a size of 10*10 as it helps the system better identify features in larger general locations of the image as identified by Graesser and Keng in the book, ‘Foundations of

Deep Reinforcement Learning' and serves better for scene understanding as opposed to feature identification.

The code for Training can be found in Appendix D. The old code for the network that was attempted initially can also be found. It failed to take advantage of Pythons inheritance properties as much as the final model code did and that was evidently the reason for its inability to work. The code would compile; however, this is deceiving and when attempting to view the network errors such as what follows would occur:

```
TypeError: __init__() missing 3 required positional arguments: 'lr', 'epochs', and 'batch_size'
```

Which is actually the incorrect issue. This issue was originally being given by an incorrect match in the BatchNorm2d function due to a 128 layer convolutional input attempting to be normalised to 32 rather than 128 as is the required input. After fixing this the following error was received:

```
TypeError: new() received an invalid combination of arguments - got (float, int), but expected one of:
 * (torch.device device)
 * (torch.Storage storage)
 * (Tensor other)
 * (tuple of ints size, torch.device device)
      didn't match because some of the arguments have invalid types: (!float!, !int!)
 * (object data, torch.device device)
      didn't match because some of the arguments have invalid types: (!float!, !int!)
```

Which is a common problem when attempting to program the network structure in this method. It will result in incorrect value types being assigned to the nn.Convd2d and other layers such that are input as a float value which will not work with PyTorch. Fundamentally, this architecture design was flawed and failed to take into account the Pythonic nature of PyTorch as well as it could have. As such it was dropped during development as it was unable to get running or accept any incoming data into the network. The network can be found in Appendix D.

A lot of time was 'wasted' on attempting to fix an arguably flawed code structure that is found in Appendix I and ultimately should have been recognised sooner that large issues were apparent. For the completion of the control scenario for the drone, an agent needs to be developed further.

# Chapter 6 - Executing OpenSLAM Algorithms

The two algorithms selected for this project were selected upon the criteria that they were research proven algorithms and they were freely available for individuals to utilise for their own research purposes – with appropriate credit given to the original authors of course. OpenSLAM hosts a variety of SLAM based algorithms that meet these criteria. Background research that had been conducted prior to the selection of the algorithms indicated that ORBSLAM was already a viable option to attempt to outperform the known SLAM issue criteria as outlined in the Literature Review. However, CEKF-SLAM was chosen based on its description given on the site and for the reason that it presented reasonably usable code. Upon further findings throughout this project, it has been determined that whilst ORB-SLAM is a viable algorithm for the project, CEKF-SLAM is less viable. It should have reasonably been investigated using the much more recent EKFmonocularSLAM algorithm provided by OpenSLAM which also aligns much more closely with the intended use case for SLAM in this project. As the name suggests, EKFmonocularSLAM utilises a monocular input however its advantage is the ability to determine camera movement based on this image in a 6DOF format and produce a sparse 3D map of the salient point features. If the project is so to be carried further by other individuals, EKFmonocularSLAM and LSD-SLAM are two viable SLAM algorithms worth investigating as they are much more recent and prove to be very effective in their utilisation.

In the initial stages of the project, it was intended to convert the algorithms over to Python such that they can be run as native scripts for AirSim without needing to utilise external connectors of UDP streams. A tool exists for the conversion of MATLAB code into Python code called LiberMate. However, the tool has since been deprecated and a list of known issues with the conversion are noted on the GIT repository. Most notably is the inability to convert command style inputs from MATLAB to Python and that not all MATLAB functions are able to be mapped to any reasonable Python (SciPy/Numpy) equivalent. There are other variations which made this not viable such as the fact that Python and MATLAB behave very different with the way they handle arrays and that certain MATLAB specific instructions, such as those linked to SIMULINK, are solely unable to be remapped within Python. When initially converting the programs over to Python, it was misleading in the progress that was being achieved as simple linking or header files could be directly ported to Python due to their simple nature. Its estimated that about 40% of the original code for CEKF-SLAM was ported from MATLAB to Python however as mentioned, any code linked to the utilisation of MATLAB specific features and functions did not have a direct Python equivalent port and it was chosen that for the sake of reasonability, the source code will remain in its original format and will need to be linked to AirSim and Unreal Engine 4 in order to operate.

Additionally, looking at ORB-SLAM and its conversion to Python was entirely not feasible. Its utilisation of features that are specific to RoS and RoS libraries are not able to be ported to Python. Additionally, it is code that is written in C++ which is entirely different in its operation to Python. Python is an interpreted language and is not converted to machine code at runtime as opposed to C++ which is. Being that C++ is a compiled language as opposed to Python being an interpreted language there are fundamental differences in the code architecture which limit direct ports – actions such as Boolean expressions operate in a fundamentally different method in each language along with variable types and the limitations of inheritance differences between each of the languages. Although it was initially desired to conduct a conversion, it was determined not to be feasible and in reality it is not a conversion of the code that would be necessary but a complete re-write of the algorithms from first principles in order to achieve a successful result. This applies to both MATLAB and C++ variants of the chosen algorithms.

Additionally, as has been mentioned throughout the paper, SLAM is not inherently a method of navigation and requires a navigational component in order to perform appropriately. The use of visual odometry can be combined when a navigational component known as an occupancy grid in which the least populous grid on the map aligns with an open frame with which the mobile robot can move. The application of an occupancy grid for this project was intended to be implemented through the Nvidia Isaac package, however it only natively supports stereo camera input to develop its occupancy grid. MathWorks provides example code for Monocular Visual Odometry based occupancy grid development with a notable aspect – it utilises a pretrained network to identify depth in the image and produce a occupancy grid in the form of a Free Space Confidence map on the incoming image.

The discussion of each algorithm, and the method utilised to run the algorithms and link them to AirSim will be discussed individually in the following sections.

## 6.1 ORBSLAM

ORBSLAM in the form provided by OpenSLAM is designed to run on RoS Fuerte/Groovy on a Unix which is indicated to be Ubuntu 12.04 in the run files. These are two far outdated platforms which are not directly available for download from the original sources anymore due to deprecation. However, they are accessible in the form of a legacy download through the RoS website. The team at RoS recommend running RoS Fuerte/Groovy in a docker environment. This was tested with a download build being successfully built from the source *fabian/ubuntu-12.04-ros-fuerte* but it is a ‘headless’ environment

meaning that it does not project a visual output of the environment and can only be interacted with via the command line. This is appropriate for general cases however it limits the visualisation of the SLAM algorithms and lessens the ability to debug issues in the container and as such it had to be abandoned as a plausible option to run the software. Additionally, I was unable to get the docker container to establish a UDP stream outside of the container and as such any reasonable number of findings that could be achieved utilising this method were considered to be null. However, it does establish that this method is not viable for test utilisation cases.

The authors of ORBSLAM state its prerequisites in the README file from the download from OpenSLAM as being:

1. Boost - a package for Linux based systems and is utilised to launch multiple threads of a workload
2. RoS Fuerte/Groovy/Hydro - RoS (Robot Operating System) is utilised in this case for retrieving and interpreting incoming image data (rosbag), and for visualisation of the process (rviz, image_view) and is required to run on the following OS;
3. Ubuntu 12.04 - This is the proven OS outlined in the prerequisites and is out of date as of publishing this paper.
4. g2o dependencies - this was used to perform the optimisations on the code, utilising the Eigen3, BLAS, LAPACK and CHOLMOD libraries.
5. DBoW2 - components of this library serve as the underlying components that allow for place recognition and feature matching in the data. It is reliant on OpenCV.

Therefore, running ORB-SLAM in the nature as it was provided directly from OpenSLAM may not be viable. It was investigated the plausibility of running the environment within a Virtual Machine. Ubuntu 12.04.5 LTS is openly available through Ubuntu directly in the form of a LTS image however it stopped receiving development support in April, 2017. However, the limitations of operating with a Virtual Machine are noticeable especially on consumer hardware with minimal resources that are able to be allocated. If the Virtual Machine is allocated 50% resources than the main desktop is left with 50% resources available to conduct rendering, background tasks, etc. This means a fair analysis is entirely unfeasible unless an entire GPU could be allocated to the Virtual Machine to provide equal compute capability. If this cannot be done than any performance comparisons would be inadequate as they would not be under the same controlled conditions. Subsequently, algorithm performance (beyond just compute characteristics) would be hindered due to removal of resources also. In the condition that an algorithm

which requires a locked FPS cannot receive the locked FPS than it will be out of phase from the incoming data and unable to react at appropriate intervals. This would result in inappropriate key frame data being presented to the SLAM algorithms.

After establishing that this was also not a viable option to run the algorithm, it was found that MathWorks has adapted ORB-SLAM as a direct option to run on MATLAB. The algorithm itself is labelled as vSLAM however it utilises ORBSLAM as the underpinning algorithm and thus can serve as a viable basis to operate the algorithm. This also allows for the limitations of a Virtual Machine to be removed as MATLAB has complete compute capability available to the algorithm during runtime i.e. the machines resources are not pre-allocated as they are in a Virtual Machine.

Running ORB-SLAM in its MATLAB variant as provided by MathWorks was ultimately decided to be the chosen implementation.

The complete code for ORB-SLAM in its MATLAB form is included in Appendix E with complete acknowledgement given to MathWorks for the code.

## 6.2 CEKF-SLAM

CEKF-SLAM is a MATLAB based algorithm built on top of EKF-SLAM as noted by the authors README file. CEKF-SLAM utilises MATLAB toolboxes and as such is required to run directly in MATLAB. There are packages available that will allow the user to run the code directly in GNU Octave however the packages are not direct ports and feasibility of operation for the code cannot be considered to be reliable. Therefore, running directly within MATLAB will be necessary.

In order to run CEKF-SLAM, the user is it load the loop902.mat file and run “data = cekfslam(lm,wp)” in the command window in MATLAB. However, the algorithm still needs to be linked to AirSim and there is still the requirement to utilise a navigational agent for both ORB-SLAM and CEKF-SLAM.

The complete CEKF-SLAM code used in this project can be found in Appendix F.

## 6.3 Linking MATLAB with AirSim

Considering that the algorithm types selected are natively operable in MATLAB there is a need to identify a method to link MATLAB to AirSim such that usable control and connection can be established. It is generally recommended to utilise a PIXHAWK controller via a UDP stream to connect MATLAB to AirSim however this is only relevant in cases where a physical device is being controlled. As the is an entirely simulations focussed project the utilisation of a PIXHAWK was deemed unviable.

There is the option of utilising a Python based MATLAB link as posted by user *hashikemu* onto their personal GIT page (Hashikemu 2018). The opposing method is to utilise a UDP connection written by user *xiaolin360* which utilises a C++ established UDP connection into AirSim such that MATLAB can operate under UDP conditions. The method utilised were based on code developed by *hashikemu* and must be noted that established methods of connecting AirSim to other applications has been determined to be ill-defined in the source material and in common materials from other projects and as such attracts attention as a research topic within itself. AirSim is extremely capable and defining an open connection standard is a must to ensure it gets appropriately utilised for research purposes where the code cannot natively be executed within the operating directory of the environment.

The Python code to link AirSim to MATLAB can be found in Appendix F.

# Chapter 7 - Experimentation Conditions

## 7.1 Defining Experiment Conditions

Referring back to Chapter 3, the points of interest for the experimentations surround around identifying the characteristics of each algorithm type and its performance in regard to;

- Image Illumination Issues
- Dynamic Problems
- Kidnap Problem
- Compute - System Utilisation

Within Unreal Engine 4 utilising the AirSim plugin. Running the experimentation portion of the project led to a large amount of changes to the project structure and additionally led to speculation about the question proposed for the project and whether or not the correct question has been proposed. These points shall be addressed in depth in Chapter 8 and 9.

It was determined that the SLAM algorithms are not capable of being run directly within Unreal Engine due to their underlying requirements as mentioned in Chapter 6. It was also found that during the investigation and conversion of the SLAM algorithms to python that it was not viable directly porting the code between languages due to variations in Python code and C++/MATLAB along with varying libraries being unavailable for Python as addressed in Chapter 6. As a result of this, the experimentation environment had to be altered in order to attempt to produce a viable outcome.

To ensure that the test conditions are as similar as possible, the temperature control implementation that can be utilised revolves around setting all fan curves and pump speed values to aggressive within Corsair ICUE and NZXT CAM to ensure there is enough airflow and cooling capability available and prevent thermal throttling. This creates a scenario of maximum possible cooling at all times and limits the ingress of thermal throttling as much as possible in the apparent data and results. As a result, this is a step towards ensure optimal compute availability at all times and limit loss of compute power due to thermal throttling.

This was done by modifying the custom fan curve providing the most aggressive fan control and the extreme output curve on the pump following the following settings within the Corsair iCUE software. It

must be noted that the CPU is water-cooled using a Corsair H100i Platinum AIO and has shown to be effective in maintaining appropriate operating temperatures. The Corsair iCUE conditions are:


Figure 43. Corsair iCUE Cooling Settings

Additionally, the NZXT CAM case fan settings can be seen utilising the following fan characteristics:

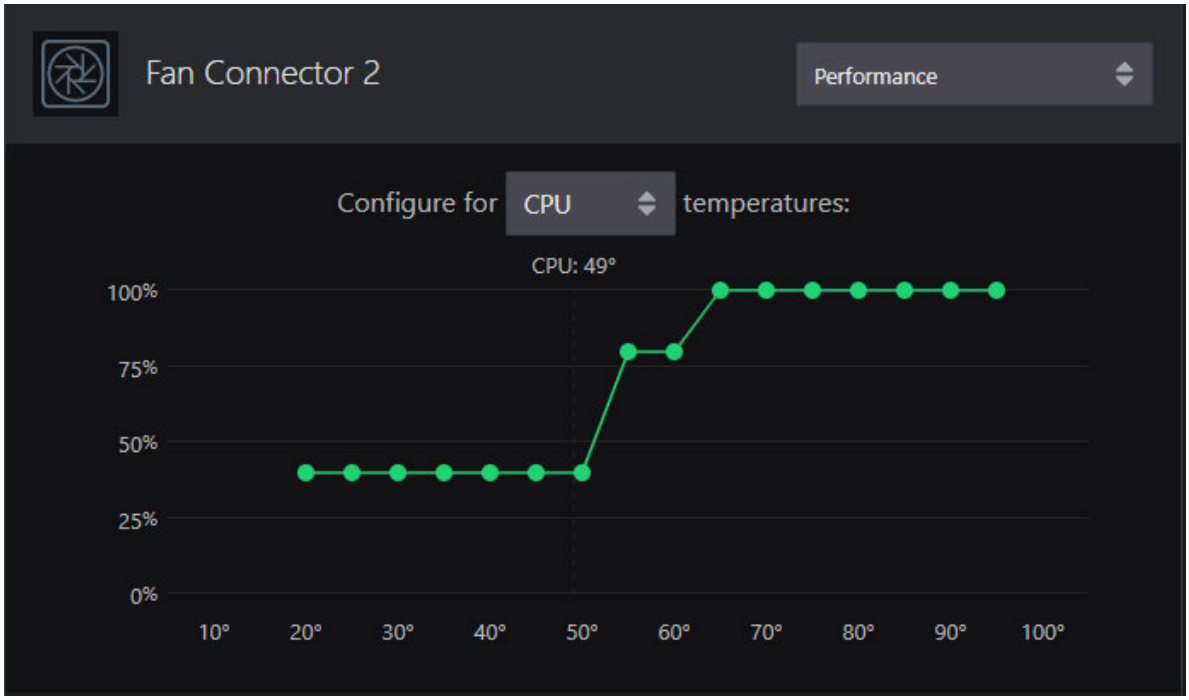



Figure 44. NZXT CAM Fan Settings

What is labelled EXTREME within Corsair iCUE is generally matched by settings labelled Performance within the NZXT CAM software. The GPU is set to a predefined aggressive profile and will remain in its default BIOS settings. There will be no overclocking of the CPU, GPU or RAM to achieve speeds that are not profiled within the specifications of the equipment and serve to add stability when executing the project software.

Monitor resolutions are not expected to produce a measurable effect on the execution of the algorithms. This is due to the input image being rendered from a virtual camera on the drone and not from a screen capture or direct screen rendering of the experimentation environment. If the conditions for rendering were directly tied to the screen resolution then the alteration in performance would be very apparent. With the main monitor resolution being 3440 x 1440 there are 4,953,600 active pixels compared to 480 x 480 which is 230,400 active pixels. This is a 21.5 times increase in pixel data over the rendered camera input and if the conditions remain linear for rendering load when aligned to pixel data then an over 20 times impact could be seen in rendering performance.

## 7.2 Image Illumination and Specular Obscuration

Assessing the characteristics of each algorithm type within the scenario of Image Illumination and Specular Obscuration revolves around the algorithms ability to limit collisions. For the cases of SLAM, the algorithms can be ran such that significant features are identified within the environment however the identification of features is not a common task for both algorithm types. The SLAM based algorithms will produce some form of a map depending on whether it is ORB-SLAM or CEKF-SLAM and will identify notable features in the data from here however the CNN network is ignorant to specific features as a result of its training and is likely to converge on emulating the operations of an occupancy grid. This means the CNN will attempt to identify open zones that are viable for the drone to move and subsequently feed instructions based on class outputs as opposed to the mathematically constructed decisions based on a mathematically determined visual occupancy grid.

By identifying each algorithms ability to navigation within this environment, it can be inferred each algorithms ability to understand local data about its surroundings. Now, it is not ignorant to believe that all algorithms types will be capable of executing flight through the map with no apparent collisions. If this scenario occurs it would be obtrusive to present a more complex scenario in the aims that it intends to make either algorithm fail. If the case arises, then it shall be assessed to determine if the initial map

design aligned strongly with the criteria identified for visual issues for SLAM algorithms. If the environment is determined to be insufficiently aligned, then a new environment will be made that more strongly correlates with the required criteria.

The criteria has been mentioned throughout the project. However, to clearly state the criteria it is:

- Produces sufficient reflections such that obscuration of a clear pathway is present
- Produces sufficient 'glare' such that camera exposure is obscured
- Produces sufficiently dark conditions and shadows such that edge and feature detection are limited
- Produces sufficient light scattering such that inconsistent lighting is present

Looking at Chapter 4 Section 4.1 the environment has been sufficiently documented such that these criteria can be assessed. The criteria, 'Produces sufficient reflections that obscure clearly defined paths' is notably met through the utilisation of reflective and mirrored surfaces in positions that make a well-defined path somewhat difficult to discern. The criteria of, 'Produces sufficient 'glare' such that camera exposure is obscured' is met through the use of particle effects and mirrored surfaces exposed after open windows using natural light for glare in hallway 3 and hallway 4. The criteria of 'Produces sufficiently dark conditions and shadows such that edge and feature detection are limited' is apparent in the last 2 hallways such that limited light exposure and apparent objects in the path fulfil this requirement. The criteria of 'Produces sufficient light scattering such that inconsistent lighting is present' is fulfilled by the seventh hallway which uses ground-based lighting scattered through glass structures to test feature acquisition in these conditions. It must be understood that the method by which the camera is utilised by AirSim does not entirely emulate a 'real-world' camera in its visual state.

The drone will be initialised and started in the exact same location every time as indicated by the Player Location marker in the map to ensure initial conditions remain the same. The run is considered complete once the drone has exited the final hallway. The points of measurement for the experiments are as follows:

1. Collection of collision data as a number of collisions counter reading
2. CPU utilisation as percentage of total capacity and core utilisation
3. GPU utilisation as a percentage of total capacity
4. RAM utilisation
5. Assessment of general flight behaviour

## 7.3 Dynamic Objectives

Assessing the characteristics of each algorithm type within the scenario of Dynamic Objectives or Dynamic Environments revolves around the algorithms ability to limit collisions and judge future positional data. This is designed to test the state estimation features of SLAM algorithms and analyse its ability to detection future positions based on objects in the environment moving without any defined relationship to the drone itself. This directly impacts the localisation and mapping aspects of SLAM algorithms and serves as a strong measure of their ability to execute the algorithm under stress conditions. For the condition of the CNN, it is not building a map (not intentionally) of its environment as a reference and therefore is a good measure of its ability to be reactive to environmental states and determine whether or not end to end CNN algorithms trained similarly on similar data can perform under these conditions.

By identifying each algorithms ability to navigate and fly within this dynamic environment. It can be estimated that these conditions will present a significant amount of difficulty for each algorithm – both SLAM and CNN – and directly aligns with a known fault for ORB-SLAM which is that for fast moving or large moving objects within its environment it can have difficulties executing its SLAM requirements.

This experiment, and subsequent experiment type, has been designed to strongly coincide with the necessary criteria that would suffice the identification of an algorithms ability operate within a dynamic environment. The criteria has been mentioned separately throughout the paper, but to clearly state it, it is:

- Produces objects with sufficient vertical and horizontal linear velocity movement in the environment
- Produces sufficient elevation and open-space change within the environment
- Produces objects with sufficient nonlinear movement velocities and paths in the environment
- Produces objects with sufficient rotational elements at a linear velocity
- Produces objects with sufficient rotational elements at a non-linear velocity

Looking at Chapter 4 Section 4.2 it can be understood that the environment sufficiently aligns with the necessary requirements to execute the dynamic conditions experimentation. As with the previous experiment, the drone will be initialised in the exact same location with every pass as indicated by the player location marker in the environment. The run is considered complete once the drone has exited the final room and will be manually stopped. The points of measure for the experiment are as follows:

1. Collection of collision data as a number of collisions counter reading
2. CPU utilisation as percentage of total capacity and core utilisation

3. GPU utilisation as a percentage of total capacity
4. RAM utilisation
5. Assessment of general flight behaviour

## 7.4 Kidnap Issue

Assessing the characteristics of each algorithm type with the scenario of a ‘kidnap’ state revolves around the algorithms ability to re-initialise within the environment after the removal of input data for the algorithms. This is designed to specifically test the ability of an algorithm to continue about its flight path even once it has been artificially moved within the path. This breaks the consistent state update and places the drone within an environment within which it may have no prior features that can be aligned with the prior environment – leaving it without a reference and subsequent ability to directly localise. This is not expected to be an issue for the CNN algorithm as it is ignorant to sequential data and acts purely on the input image data. Removing the camera input is expected to introduce unknown behaviours however once it receives input data again it is assumed that it will begin to react to its input data as if the ‘kidnap’ scenario had never occurred.

By identifying how each algorithm performs under this condition it can be inferred the beneficial algorithm to utilise in a case where this may be a suspected scenario. This experiment is environment or landscape agnostic – it is designed to be executed on both Landscape 1 and Landscape 2 and is implemented through the use of removal of the camera input and transport between locations on the map using a teleportation within Unreal Engine 4. This sounds difficult, but essentially if the hitbox detects that the drone has collided with its boundaries, then a timer is set to input a zeroed out input image and the drone is moved from location A on the map to location B via an immediate update to the X, Y and Z coordinates. It will retain the orientation it had prior to the ‘teleportation’.

The points of measure for the experiment are as follows:

1. Time to localise
2. Localised successfully Y/N
3. Assessment of general flight behaviour

## 7.5 Realistic Environment

It was intended to repeat the aforementioned experiments on an environment that as closely emulated a realistic scenario as possible to determine whether or not the findings of the prior experiments where indicative of worst case scenario performance only or if whether the resulting data was truly indicative of the performance of the algorithms.

However, this portion of the project was determined to be cut due to limitations in the design process for the map. The time to build a realistic environment as opposed to a corridor like design in both Landscape 1 and Landscape 2 is exponentially (in terms of my capability) longer and was thus dropped as a viable option.

Data captured from this experiment would have served as a strong reference for the algorithm performance in general conditions and would have likely provided strong evidence for or against an algorithm type. For instance, if the CNN algorithm heavily outperformed in all tests, but the SLAM based algorithms outperformed in the realistic environment then it could be inferred that under very tightly controlled scenarios the CNN is more beneficial and that the SLAM would be more beneficial for real world applications. This type of finding would support the current industry belief in SLAM algorithms and provide strong evidence against the hypothesis that CNNs would outperform SLAM algorithms.

Regardless, further investigation into this area can be noted for future work and can be an optimal area of investigation for future projects due to the limited data and supporting evidence within the field.

## 7.6 Experiment Execution

Each experiment is intended to be run a minimum of five times to ensure that a sufficient amount of data is acquired with enough variation to produce a mean value with standard deviations of the data. Exceeding three experiment attempts allows for beyond a 66% reasonability towards the result (as would be found generally in experiments repeated three times) with up to an 80% reasonability on the outcome being true and exactly repeatable.

# Chapter 8 - Results and Discussion

## 8.1 Defining Baseline Performance Values

In order to appropriately measure the compute impact that each algorithm has on the system, there is a baseline value that must be determined for each landscape. In order to accurately estimate this baseline, there is a level of background noise that must be maintained. The PC will be running Windows 10 with all settings set as default and those that are changed do not directly affect the performance. Windows security is active. All tasks except those necessary for the simulations will be closed and any unnecessary background tasks will be closed (i.e. NZXT CAM).

The data will be captured within the Visual Studio 2019 data logging software for debugging purposes. It will allow the user to track CPU utilisation (even per core utilisation) and GPU utilisation. These two points of data are plotted with respect to time. Therefore, appropriately timing the start of the test runs will allow for a measurable baseline to be developed. An issue in using this method however is that it does not allow the user to collect information on the memory usage during the debugging process (the debugging process is necessary to run the AirSim plugin in the PiE).

The baseline system load, without any applications open (from a cold boot of windows) leaves the CPU utilisation at roughly 1-2% (arguably in the range of error as such low utilisation generally ignores impact of cumulative small tasks). The GPU is close to 1% utilisation and the RAM (memory) usage idles at roughly 18% of 32gb which is approximately 5.76gb.

Looking at Landscape 1, this is expected to be the most compute intensive due to a high number of reflections and the lighting conditions that are placed on the environment. Dynamic lighting from the sparks and adjacent mirror exist and are computationally expensive for the GPU. The data will be captured using AirSim as the GameMode under global details with the drone being flown manually to reduce any additional computational load. Understanding this, the baseline load can be determined as follows:

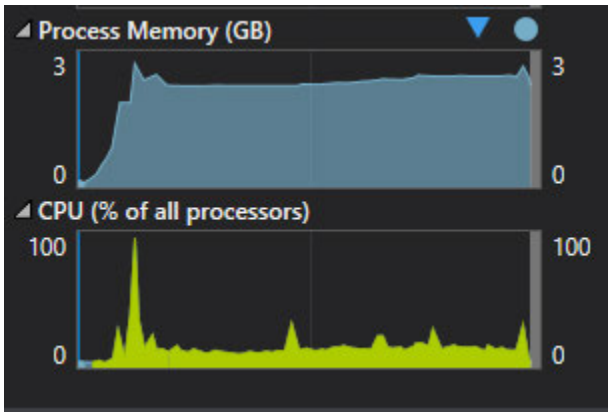



Figure 45. CPU Compute Load and RAM Utilisation Capture from Visual Studio 2019 – Landscape 1

It can be determined that the CPU utilisation and minimal, only seeing spikes when the system initialises flight as can be seen in the initial quarter of the CPU processor utilisation graph. As for RAM utilisation, it is also quite minimal only using an average of 2.8GB of RAM continuously. The GPU fails to be able to have its utilisation captured within Visual Studio when using AirSim – it is unsure if this is a bug or due to other causes. However, the GPU was captured using Performance Monitor. Please note, this is not an entirely accurate representation of the GPU utilisation and will be used as a reference only.

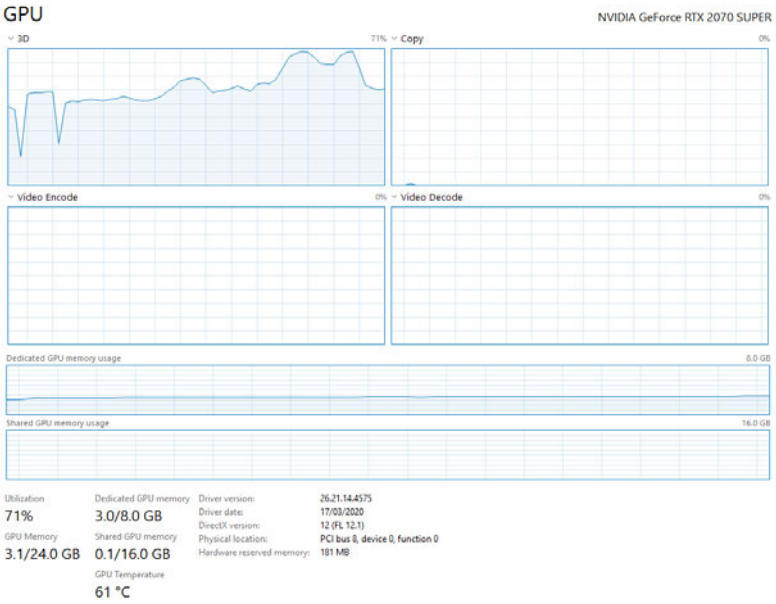



Figure 46. GPU Compute Load for Landscape 1

Looking at the compute characteristics of this environment there are worrying results presented. The compute environment, when entering heavily reflective or conditions such as can be see in Figures 23, 24 and 26 in Chapter 4 of the project the GPU utilisation reaches 100% utilisation. This leaves no available

computer capability for either algorithm to be able to be executed on the GPU. Even though CUDA acceleration far exceeds the compute speeds of a CPU for CNN deployment there may be no option but to train the CNN on the GPU and deploy it on the CPU such that it is not artificially limited by Landscape 1's compute profile.

Looking at Landscape 2, this is expected to be the least compute intensive due to less complex scenarios revolving the environment. There are no 'tricks' being played on the system in the form of mirrors or shadows, the system simply needs to compute linear and nonlinear movement paths and render the scene in real time. The lighting is purely global with simple lighting fixtures used to remove shadowing as much as possible and to highlight internal objects. The data will be captured using AirSim as the GameMode under global details with the drone being flown manually to reduce any additional computational load. Understanding this, the baseline load can be determined as follows:

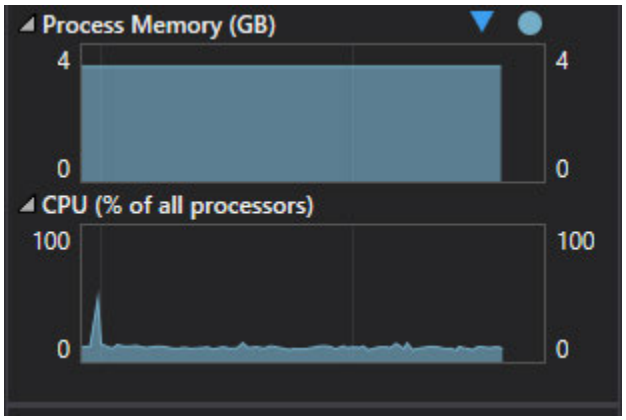



Figure 47. CPU and RAM Profile for Landscape 2

As was the case with Landscape 1, CPU and RAM utilisation are minor and negligible with consideration to the total capacity of the system. Assessing the GPU profile provides the following:

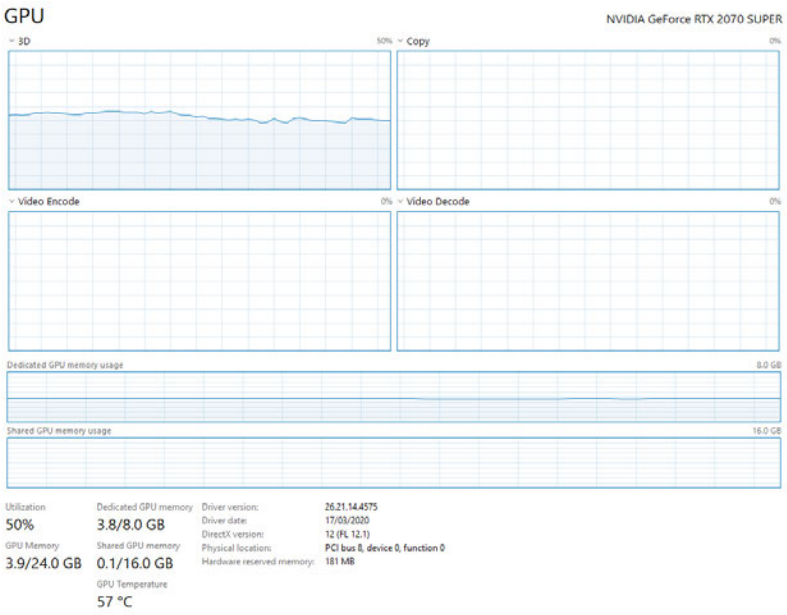



Figure 48. GPU Profile for Landscape 2

In this environment the utilisation of the GPU is far below the average utilisation as seen in Landscape 1. At not point throughout test flights was it possible to force the GPU to exceed 60% utilisation and as such this can be considered a much stronger environment for the assessment of the algorithm's utilisation compared to Landscape 1.

With a basic understanding of the generalised compute load for each complete landscape it is understood that Landscape 1 may present errors in the form of a small compute window available for either algorithm to be executed. Under this scenario, the SLAM algorithms running on the CPU are expected to perform the best however utilising a control scenario of the CNN and SLAM algorithms being run on the CPU and GPU where possible shall minimise computational limitations due to underlying compute loads. This already begins to present a minor underlying issue with the utilisation of simulations for a project such as this and lends to the benefits of possibly utilising a dual GPU array - being able to allocate a single GPU to render the landscape and a single GPU to process the algorithms. This would be possible using Nvidia 2070 Super GPU's through an NVLink bridge however they can also be left disconnected and this would leave GPU0 (the one in the top most slot for the majority of motherboards - taking the 16x lane) being the primary card for rendering the landscapes and then the left over GPU can be addressed through PyTorch and allocated by defining device as being GPU1. It is not understood what the outcome would be for this scenario and it does not fall within the scope of the project to entertain the idea. Additionally, not utilising NVLink would result in the GPU's being unable to parallelise the workload and share the GPU buffer for additional shared memory.

The results from this section can be used to establish a baseline performance value for each scenario and allows for the results to be normalised such that the common compute elements can be removed and a clear understanding of the compute load of each algorithm can be established.

## 8.2 Results Identification & Understanding

Understanding results of a direct comparison within the Unreal Engine environment utilising AirSim was not able to be achieved. A significant amount of progress has been made towards the completion of the goal however software limitations and hurdles discovered along the way have hampered the ability to complete the task. Ultimately however, a conclusion can be drawn from the collective literature and current completed work.

When assessing the use case of AirSim and Unreal Engine 4 as a viable option for the execution of the project, it has more than served its purpose as a research tool. However, it does present cases of immaturity in the software. I would argue that it is not solely the fault of AirSim however for these issues. Looking at what AirSim offers, it can be boiled down very simply as being an interpretive plugin for Unreal Engine 4 in order to conduct complex realistic control of a drone or vehicle. To which I would say it is perfect at doing this and deserves far more recognition than what it receives. However, attempting to operate within the environment is where the shortcomings occur – but this is not a direct fault of AirSim.

When looking at ORB-SLAM and CEKF-SLAM they are both provided in this project in the form of executable MATLAB files. However, AirSim fails to provide direct useability with MATLAB through its API. A layer needs to be built in order to interpret MATLAB commands and computation and relay this to the AirSim engine. This was largely under looked at the beginning of the project and ultimately served as the most difficult condition to resolve throughout the project. A solution was found in the form of a Python linking file for MATLAB however it was unable to get to an operational condition for the project. Looking at Chapter 6, an in-depth break down on the attempts to run both ORB-SLAM and CEKF-SLAM within the environment is given however a clear result was not able to be achieved. It appears as though AirSim is only able to ‘play nice’ with code written that is able to run directly as either a C++ or Python script. This makes sense when considering that the AirSim API is provided only with respect to a C++ or Python option.

## 8.3 Project Outcome Assessment

Whilst the project is clearly deep into development and well on its way towards being able to provide feasible and useable data, it is not quite there yet with regards to being able to provide strong evidence for or against the hypothesis of the project in terms of data produced by the author. However, using the anecdotal evidence discovered throughout the project alongside the literature backing, projections on the results can be made. This can be found in the following chapter. Additionally, the project strongly aligns with the aim presented both in this document and in the Project Outline as found in Appendix A. however it was necessary to make alterations away from the initial Project Outline in terms of project execution. The initial Project Outline was somewhat immature, and the initial position compared to the current findings clearly established that this is an immature area of research which will benefit strongly from the steps and procedures outlined within this dissertation. Developing a clear method of utilising PyTorch with AirSim is limited in the literature as only TensorFlow and CNTK are apparent in the AirSim GIT Repository. Additionally, the supporting documentation surrounding running MATLAB and AirSim together is extremely lacking and this project has identified this clearly validating evidence.

Considering that the aim was to establish consensus on the comparison of the two algorithm types in a simulated environment, a clear result can be drawn that under these conditions and utilising the same software, there is a clear benefit towards the use of CNNs to achieve the end goal. Attempting to utilise research proven SLAM algorithms projects faith in their reliability however it fails to take into account limitations of the environments within which they will be executed.

Overall, this project has provided evidence for the case of CNNs over SLAM based algorithms however the findings are not conclusively strong enough to draw a defining decision on the process. A resulting finding has been a clear outlining of the limitations of the software utilised in this project and clear definition of methods to both develop viable simulated environments for testing, which is limited in the literature, and methods of executing simulations within simulated environments and as such holds research significance.

# Chapter 9 - Conclusion and Future Work

## 9.1 Project Conclusion

The conclusion of the project presents inconclusive data but does present anecdotal data with a strong literature backing. The literature presents positive indications that CNN or similar algorithms will outperform SLAM based algorithms when considering an end to end implementation but the inability to conduct all experiments limited the ability to draw a conclusive decision. There are positive indications in the direction of Deep Learning (CNN) algorithms being the 'better' algorithm type for pure control in the literature, but these reference real world test cases. The paper 'Deep Convolutional Neural Network-Based Autonomous Drone Navigation' presents a strong case for the performance of CNN's directly within AirSim however it does not present the opposing case for a SLAM based algorithm. From a personal orientation, it has been found that building and using CNNs are far more accessible and 'easy' but this does not indicate whether the algorithm type outperforms SLAM based algorithms. To dot point the personal findings surrounding CNNs from this project:

- It is easier to develop and implement CNN for control scenarios
- It is less complex to bug test CNN algorithms
- Data and training are the main difficulty with CNN

The findings determined surrounding SLAM are rather limited. In this case it was determined the appropriate method with which they can be ran within AirSim and the Unreal Engine development environment. Their performance cases could not be clearly established however this project serves as a strong base for future research and clearly defines the processes and necessary steps required to build upon the project. Looking at the Convolutional Neural Network utilised for this project, its structure has been clearly defined and utilises strong literature-based evidence in the design of its architecture and serves as a strong base for future research in this area to be built on top of.

It was found that a far larger time requirement was necessary and it realistically may not have been achievable within the window provided. The development of the simulated environments resulted in an estimate 300 hours spent building and resolving issues in Unreal Engine 4. It can be stated that a project such as this should only utilised prebuilt environments, and this is somewhat supported when reviewing the literature. The paper by Amer et. al. used prebuilt environments and can be considered a strong

recommendation for future work in this area. However, when taking into consideration the need to test specific scenarios this is not entirely feasible to use third party or pre-built environments.

When taking into consideration the project, the initial proposed question could have been better oriented. Looking into a pure CNN vs SLAM is largely problematic due to the large number of variables that fall in line with one another with the only similarity being that they provide the ability for complex control. It is not as simple as asking 'CNN or SLAM' as they are fundamentally different in their architecture design, even though they are built upon for autonomous purposes. When looking at end to end CNN vs SLAM, one must consider what is gained and lost in each algorithm. SLAM produces a map and a knowledgeable, clearly defined path. CNN, in this implementation, produces response to input without much knowledge of how this movement impacts its path or mission execution.

Ultimately, referring back to the aim we can see that the project has provided a significant amount of background information and significant findings in the literature that aim to assist the findings of the project towards the determination of whether or not a Deep Learning based algorithm or a SLAM based algorithm is the optimal control algorithm within simulated environments. The AirSim API appeared to 'overpromise' on its capabilities – however it is likely that this is no fault of the developer. AirSim sits as a plugin on top of Unreal Engine and as such a lot of its limitations are extended to AirSim. Taking into consideration the listed aim of attempting to provide a clear understanding of each algorithms characteristics it can be determined that this paper provides strong evidence in both areas and does satisfy the aim of the project to a degree.

The findings of this project have clearly established the methods necessary to conduct research in this area and serves as a strong point of reference for future work in this area. The methodology required to execute the algorithms, the necessary controls and methods to measure and validate findings have been outlined and are relevant findings in this area of literature. It has clearly established the necessary process required to develop and build a simulated environment that meet the necessary requirements of simulated experiments such as this and of clearly defined criteria within which future research can be conducted. It has also been clearly defined the limitations of this simulated environment and the ability to execute algorithms within the environment. The baseline performance values of simulated environments such as the ones present in this project have been clearly established and serves to present the system with which would be necessary to conduct the experimentations to a satisfactory degree if the same simulations engine is to be utilised.

Additionally, the impact that COVID-19 had on the outcome of the project was vast. Although this was a simulations based assignment, being unable to go on campus and conduct meetings with my supervisor did create disconnect however I feel as if I was able to create a good amount of communication when necessary and provided all necessary documentations for all submissions in the course even when they were not required. The impact that COVID-19 had on my personal situation severely hampered my ability in the initial stage of the project. During the complete lockdown with my income not guaranteed and uncertainty on my personal housing conditions it was difficult to direct focus in any other direction. I feel I have been robbed of the opportunity to truly display the capability of this project, however my heart goes out to those who have sadly lost their lives during the pandemic and it is truly grounding to have lived through a pandemic such as this.

## 9.2 Future Work Identification

Looking beyond the completion of this project there are a multitude of future tasks that can be defined in order to establish a clear definitive result. These future tasks are:

- Further implementation of the navigational agent for the SLAM based MATLAB programs such that complete drone control and operability within AirSim is achieved
- Further training of the CNN such that its control usability becomes more apparent with the completion of an agent to act of network outputs
- Completion of the realistic environment such that it can be used to delineate between pure controlled scenario results and realistic scenario data

This would present the completion and ability to provide conclusive evidence towards the performance execution characteristics of each algorithm type. Moving on from this, it would be a strong area to look into the variation in outcome between the simulated environment and a real-world application in a controlled scenario with CASA approval.

The utilisation of SLAM and CNN, more appropriately, the combination of SLAM and Deep Learning is not a new field of research. There are a plethora of possibilities when assessing Deep Learning and SLAM against one another. It would be interesting to investigate the plausibility of a Deep Learning algorithm to approximate a SLAM algorithm such that it could reduce the compute complexity of current SLAM. This could also be utilised to replace the navigational agent of the SLAM algorithm, taking in the map data from the environment and make decisions based on a trained conditional agent. It is not silly to

assume that a Deep Learning algorithm could be taught to approximate the actions of a SLAM algorithm and be fed into another network which utilises the output data to act as a navigational agent. Evidently, in cases such as this utilising an RNN in some form has evidence in the literature of being beneficial for control.

One thing that has been clearly established in this project is that AirSim, when combined with Unreal Engine 4, provides a strong research base to conduct complex simulation and experimentation that would ultimately be next to impossible to achieve in any other environment. Building on the further research capabilities that AirSim provides, its evidently clear that the team has intended AirSim to be focussed on running algorithms that are more ‘modern’. Looking at this, it would be interesting to determine the complex control capacities of AirSim. Is it able to be used for control learning such as development of PID controllers for university courses? Is it able to be used for emulation of tasks such as the Warman Challenge and allow students to test virtual versions of their models prior to the final test? The possibilities of AirSim and Unreal Engine are apparently evident however it would be beneficial for the industry to establish a clear understanding of what is and what is not completely possible when solely utilising AirSim.

# References


Abbas Sadat, S, Chutskoff, K, Jungic, D, Wawerla & Vaughan, R 2014, 'Feature-Rich Path Planning for Robust Navigation of MAV's with Mono-SLAM', 2014 IEEE International Conference on Robotics & Automation (ICRA), Hong Kong, China, pp. 3870-5

AHCG 2019, Risk Matrix, Consequence and Likelihood Tables, n.a., Health Care Governance, viewed 13 October 2019 <http://www.healthcaregovernance.org.au/docs/risk-matrix.doc>

AirSim 2018, *Core API*, Microsoft Research, viewed March 2020, <https://microsoft.github.io/AirSim/apis/>

AirSim 2018, *Image API*, Microsoft Research, viewed March 2020, <https://microsoft.github.io/AirSim/Image_apis/>

AirSim 2018, *Settings*, Microsoft Research, viewed March 2020, <https://microsoft.github.io/AirSim/settings/>

C. Clifford 2018, *Elon Musk: 'Mark my words – A.I is far more dangerous than nukes'*, CNBC, viewed 15 September 2020, <https://www.cnbc.com/2018/03/13/elon-musk-at-sxsw-a-i-is-more-dangerous-than-nuclear-weapons.html#:~:text=Tesla%20and%20SpaceX%20boss%20Elon,the%20danger%20of%20artificial%20intelligence.&text=%E2%80%9CThe%20biggest%20issue%20I%20see,tends%20to%20plague%20smart%20people>

CASA 2019, Important Drone Safety Information, n.a., 1905.2370, Civil Aviation Safety Authority, 13 October 2019 <https://www.casa.gov.au/sites/default/files/drone-safety-rules-factsheetenglish.pdf>

Centre for Neuro Skills n.a., *Neuronal Firing*, Centre for Neuro Skills, viewed 18 August 2020, <https://www.neuroskills.com/brain-injury/neuroplasticity/neuronal-firing/>

DeepAI n.a., *Perceptron*, Web Page, Deep AI, viewed 14 September 2020 < https://deepai.org/machine-learning-glossary-and-terms/perceptron>

E. A. Wan, R. van der Merwe, "The Unscented Kalman Filter," in Kalman Filtering and Neural Networks, Hoboken, New Jersey, USA: Joh Wiley & Sons, Inc, 2001, ch. 7, pp. 1 - 50. [Online]. Available: https://www.pdx.edu/biomedical-signal-processing-lab/sites/www.pdx.edu.biomedical-signal-processing-lab/files/ukf.wan_.chapt7_.pdf

G. Zunino. (2002,February). Simultaneous Localization and Mapping for Navigation in Realistic Environments. presented at MFIIS, 2001. [Conference Paper].<https://www.researchgate.net/publication/3955081_Simultaneous_localization_and_mapping_in_domestic_environments>

Garcia, S, Elena Lopez, M, Barea, R, M. Bergasa, L, Gomez, A & J. Molinos, E 2016,'Indoor SLAM for Micro Aerial Vehicles Control using Monocular Camera and Sensor Fusion', 2016 International Conference on Autonomous Robot Systems and Competitions, Alcala de Hanres, Spain, pp. 205-10

Grace W. Lindsey 2020, *Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future*, PhD, University College London, London, viewed 06 October 2020 < https://arxiv.org/ftp/arxiv/papers/2001/2001.07092.pdf>

Guivant, J & Nebto, E 2001, 'Optimization of the Simultaneous Localization and Map-Building Algorithm for Real-Time Implementation', IEEE Transactions on Robotics and Automation, June 2001, VOL 17., NO 3., pp. 242-257

Hashikemu 2018, *AirSim and MATLAB integration using python problem #1035*, GITHUB, viewed September 2020, <https://github.com/microsoft/AirSim/issues/1035>

Huang, R, Tan P & M. Chen, B 2015, 'Monocular Vision-Based Autonomous Navigation System on a Toy Quadcopter in Unknown Environments', 2015 International Conference on Unmanned Aircraft Systems (ICUAS), Denver Marriot Tech Centre, Colorado, pp. 1260-69.

Indiana University South Bend n.a., *Neural Networks*, Indiana University South Bend, viewed 02 October 2020, <https://www.cs.iusb.edu/~danav/teach/c463/12_nn.html>

Jones, E, Sofonia, J, Caneles, C, Hrabar, S & Kendoul, F 2019, 'Advances and applications for automated drones in underground mining operations', Deep Mining 2019 Conference, The Southern African Institute of Mining and Metallurgy, Muldersdrift, pp. 323-35

K. Amer., M. Samy., M. Shaker., M. ElHelw 2019, *Deep Convolutional Neural Network-Based Autonomous Drone Navigation*, Centre for Informatics Science, Nile University, Giza, Egypt, viewed 17 August 2020, <https://arxiv.org/ftp/arxiv/papers/1905/1905.01657.pdf>

Ki Kim, D & Chen, T 2015, 'Deep Neural Network for Real-Time Autonomous Indoor Navigation', paper, Cornell University, New York

Lopez, E et. al, 2017, 'A Multi-Sensorial Simultaneous Localization and Mapping (SLAM) System for Low-Cost Micro Aerial Vehicles in GPS-Denied Environments', Sensors, 17, 802., viewed 26 May 2020

M. Bojarski., D. Del Testa., D. Dworakowski., B. Firner., B. Flepp., P. Goyal., L. Jackel., M. Monfort., U. Muller., J. Zhang., X. Zhang., J. Zhao 2016, *End to End Learning for Self-Driving Cars*, Nvidia Corporation, pp. 1-9.

MathWorks n.a., *Monocular Visual Simultaneous Localization and Mapping*, MathWorks, viewed 11 August 2020, < https://au.mathworks.com/help/vision/ug/monocular-visual-simultaneous-localization-and-mapping.html;jsessionid=cac7d447c33a653595a1693e69a0>

MathWorks n.a., *Ocupancy Grids*, MathWorks, viewed 11 August 2020, < https://au.mathworks.com/help/robotics/ug/occupancy-grids.html>

MathWorks n.a., *Create Occupancy Grid Using Monocular Camera and Semantic Segmentation*, MathWorks, viewed 11 August 2020, < https://au.mathworks.com/help/driving/ug/create-occupancy-grid-using-monocular-camera-sensor.html>

Mur-Artal, R & Montiel, J & Tardos, J 2015, 'ORB-SLAM: A Versatile and Accurate Monocular SLAM System', IEEE Transactions on Robotics, 18 September, pp. 1-17.

N. Correll, 'Introduction to Robotics #4: Path-Planning', 2011. [Online]. Available: <http://correll.cs.colorado.edu/?p=965>

Palossi, D et. al. 2019, 'A 64mW DNN-based Visual Navigation Engine for Autonomous Nano-Drones', IEEE Internet of Things, 14 May , pp. 1-15.

PyTorch 2019, *AdaptiveAvgPool2D*, PyTorch Documentation, viewed July 2020, <https://pytorch.org/docs/stable/generated/torch.nn.AdaptiveAvgPool2d.html#torch.nn.AdaptiveAvgPool2d>

PyTorch 2019, *BatchNorm2D*, PyTorch Documentation, viewed July 2020, <https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html#torch.nn.BatchNorm2d>

PyTorch 2019, *Conv2D*, PyTorch Documentation, viewed July 2020, <https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html#torch.nn.Conv2d>

PyTorch 2019, *Linear*, PyTorch Documentation, viewed July 2020, <https://pytorch.org/docs/stable/generated/torch.nn.Linear.html#torch.nn.Linear>

PyTorch 2019, *MaxPool2D*, PyTorch Documentation, viewed July 2020, <https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html#torch.nn.MaxPool2d>

PyTorch 2019, *ReLU*, PyTorch Documentation, viewed July 2020, <https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html#torch.nn.ReLU>

PyTorch 2019, *TORCH.NN.UTILS.SPECTRAL_NORM*, PyTorch Documentation, viewed July 2020, <https://pytorch.org/docs/stable/generated/torch.nn.utils.spectral_norm.html?highlight=spectral#torch.nn.utils.spectral_norm>

PyTorch 2019, *TORCH.SAVE*, PyTorch Documentation, viewed July 2020, <https://pytorch.org/docs/stable/data.html?highlight=image%20loader>

PyTorch 2019, *TORCH.UTILS.DATA*, PyTorch Documentation, viewed July 2020, <https://pytorch.org/docs/stable/data.html?highlight=image%20loader>

S. B. Williams, G. Dissanayake, H. Durrant-Whyte. (2002, May). An efficient approach to the simultaneous localisation and mapping problem. presented at 2002 IEEE ICRA. [Conference Paper], <https://ieeexplore.ieee.org/document/1013394>

S. Lefkowitz 2019, *Professors Perceptron paved the way for AI – 60 years too soon*, Cornell Chronicle, Ithaca, NY, viewed 04 October 2020, < https://news.cornell.edu/stories/2019/09/professors-perceptron-paved-way-ai-60-years-too-soon>

S. Riisgaard, M.R. Blas, "About SLAM," in SLAM for Dummies, 1st ed. Cambrdige, Massachusetts, USA: DSPACE@MIT, 2004, ch. 3, pp. 6. [Online]. Available: <https://dspace.mit.edu/bitstream/handle/1721.1/36832/16-412JSpring2004/NR/rdonlyres/Aeronautics-and-Astronautics/16-412JSpring2004/A3C5517F-C092-4554-AA43-232DC74609B3/0/1Aslam_blas_report.pdf>

Shah, S et. al. 2017, 'AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles', Whitepaper, Microsoft Research, Redmond, WA, USA

Shi, G et. al. 2019, 'Neural Lander: Stable Drone Landing Control Using Learned Dynamics', 2019 ICRA, 4 March, pp. 1-7

T. Lacey, 'Tutorial: The Kalman Filter," in n.a., n.a. ed. Cambridge, Massachusetts, USA: MIT, n.a., ch. 11, pp. 133 – 140 [Online]. Available: <http://web.mit.edu/kirtley/kirtley/binlustuff/literature/control/Kalman%20filter.pdf>

TED | Raffaello D'Andrea, 2013, The astounding athletic power of quadcopters | Raffaello D'Andrea, video, viewed 12 March 2020, <https://www.youtube.com/watch?v=w2itwFJCgFQ>

V. Khandelwal 2017, *The Architecture and Implementation of VGG-16*, Towards AI, viewed 21 September 2020, <https://medium.com/towards-artificial-intelligence/the-architecture-and-implementation-of-vgg-16-b050e5a5920b>

Y. Xu, K. Xu, J. Wan, Z. Xiong, Y. Li. 2018, 'Research on Particle Filter Tracking Method Based on Kalman Filter', *2018 2nd IEEE IMCEC*, May 2018 Available: <https://ieeexplore.ieee.org/document/8469578>

J. Leonar & H. Durrant-Whyte 1991, 'Simultaneous Map Building and Localization for an Autonomous Mobile Robot', *International Workshop on Intelligent Robots and Systems*, Osaka, Japan, pp. 1442-7

W. Burgard., C. Stachniss., K. Arras. & M. Bennewitz n.a., *Introduction to Mobile Robotics – SLAM: Simultaneous Localization and mapping*, PDF Document, University of Freiburg, Germany, viewed 22 September 2020, < https://lor.usq.edu.au/usq/file/4994b638-35fd-407c-8877-07dfea95d5f4/6/USQ-Harvard-AGPS-Referencing-Guide-printPDF-June2020.pdf>

# Appendices

**Appendix A: Project Specification**

**Appendix B: Project Timeline**

**Appendix C: CNN Code for Training and Development**

**Appendix D: ORB-SLAM**

**Appendix E: CEKF-SLAM**

**Appendix F: Linking Code for MATLAB to AirSim**

**Appendix G: MATLAB Occupancy Grid**

**Appendix H: Failed CNN Code for Training**

**Appendix I: Sample of Collected Training Data**

**Appendix J: Engineers Australia Code of Ethics**

# Appendix A

## Project Specification

| | |
|---|---|
| For: | Brayden McConnell |
| Title: | Comparison of SLAM and Neural Networks for Autonomous Control |
| Major: | Mechatronic Engineering |
| Supervisor/s: | Dr. Tobias Low |
| Confidentiality: | -- |
| Enrolment: | ENG4111 – ONC S1, 2020<br>ENG4112 – ONC S2, 2020 |
| Project Aim: | The aim of the dissertation is to investigate the capabilities of Neural Network algorithms in a simulated environment and to compare their capabilities to that of SLAM based algorithms. |

**Program:** **Version 2, 1st April 2020**

1. Complete review of literature surrounding Neural Network based control theory and neural network architecture design.
2. Setup a local GIT environment on a local home server for version control and back up purposes.
3. Develop the test environment in Unreal Engine and build the AirSim package add-on.
4. Develop a model of a monocular, 4-rotor drone with visual input in the test environment.
5. Review and investigate optimal frame rates for visual input for estimate compute times required between frames along with the appropriate input resolution.
6. Identify the appropriate structure to develop the Neural Network on.
7. Develop a Neural Network for testing and utilise 2 pre-written OpenSLAM algorithms for comparison. Convert code and install plug-ins as necessary for operation in Unreal Engine using Visual Studio plug-in for live code update and builds.
8. Error test the SLAM algorithms using the generic 'blocks' environment provided with AirSim for debugging purposes.
9. Train the Neural Network using image data (sourced) and in-engine environment exposure/training available via AirSim.
10. Design and build 3 main courses for experimentation in each area of interest:
    a. Area 1: focusses on issue of Image and Illumination. The area will involve difficulties such as reflections and darkened environments. Image feature recognition will be tested on surfaces with varying colours, mirrored/translucent objects and shading/lighting variations.

b. Area 2: focusses on dynamic environments. This includes objects that move linearly and non-linearly within the world. This will also include tests on environment feature density and texture density of surfaces.
c. Area 3: focusses on the ‘kidnap’ problem. This is the issue of understanding position change in space when non-recorded movement has occurred. The repercussions will be tested by moving without visual input for an underdetermined amount of time and identifying system reliability afterwards.

11. Design and build 4 new areas to mimic reasonable real-world scenarios. Navigation through a cavernous tunnel (lighting varied), Dense forestry (Rainforest), Dense city environment and the interior of a building. All assets are provided by Unreal and AirSim. Environments will contain aspects of the 3 aforementioned focus areas.
12. Throughout the aforementioned experiments. The system compute load will be recorded and analysed.

*Note: The Neural Network and SLAM algorithms will be bug tested and improved throughout all experiments. A final revision will be determined and experimental data from all runs will be compared.

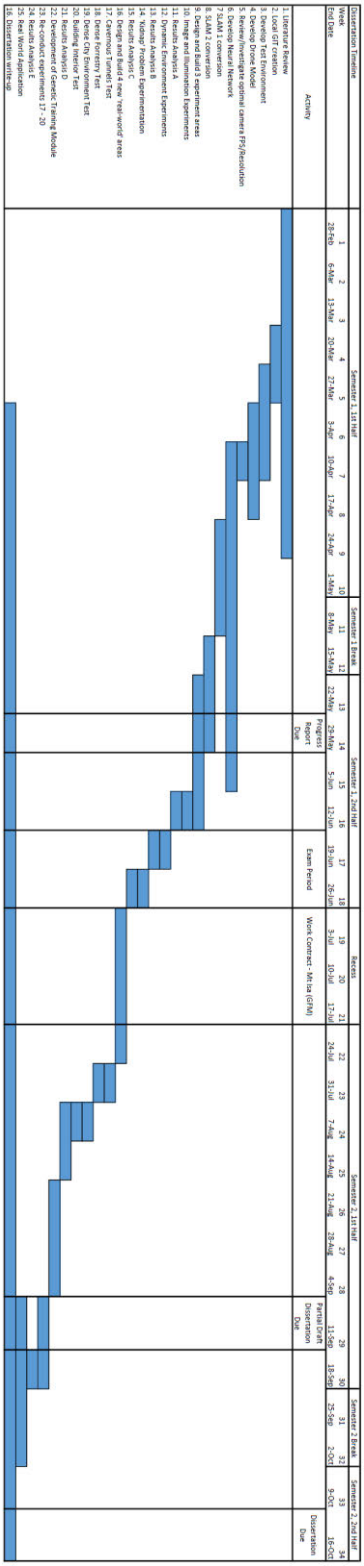
*If time and resources permit:*

13. Develop a genetic training system for continuous self-improvement of the neural network – survival of the fittest, top 10% – 15% of the highest performing weightings repopulate the next generation
14. Real world tests of simulator trained networks. Emulate (without a high level of accuracy) a desired location. After receiving CASA approval to run an unmanned test flight there will be a comparison of flight capabilities between simulated and real-world environments. A remote control will be used with a kill switch to disable the battery in the event that it is necessary.

*Note: step 14 relies on the approval of CASA. This is a ‘nice’ additional step and not a necessary step required to compare algorithm types. This is a comparison of algorithm performance between simulated and real-world environments and falls slightly out of the main focus of the research.

Contact Frequency: Weekly meetings varying between face to face and Zoom based meetings to display code and simulations progress. The day to day actions and meeting minutes will be tracked in a shared google docs document and ‘to-do’ tasks will be outlined in a shared Trello task board.

# Appendix B


Dissertation Timeline
Week
End Date
Activity
Semester 1- 1st Half
Semester 1 Break
Semester 1- 2nd Half
Recess
Semester 2- 1st Half
Semester 2 Break
Semester 2- 2nd Half
1: Literature Review
2: Identify Hardware
3: Develop Test Environment
4: Develop Drone Model
5: Review/Investigate optimal camera FPS/Resolution
6: Develop Neural Network
7: SLAM-2 conversion
8: SLAM-3 conversion
9: Design and Build 3 experiment areas
10: Image and Illumination Experiments
11: Results Analysis A
12: Path Planning Experiment Experiments
13: Results Analysis B
14: Vidmap Problem Experimentation
15: Results Analysis C
16: Design and Build 4 new 'real-world' areas
17: Dense Forestry Test
18: Dense City Environment Test
19: Building Interior Test
20: Building Analysis D
21: Development of Generic Training Module
22: Re-conduct experiments 17 - 20
23: Re-conduct experiments 13 - 16
24: Results Analysis E
25: Real World Application
26: Dissertation write-up
1 28-Feb
2 6-Mar
3 13-Mar
4 20-Mar
5 27-Mar
6 3-Apr
7 10-Apr
8 17-Apr
9 24-Apr
10 1-May
11 8-May
12 15-May
13 22-May
14 29-May
15 5-Jun
16 12-Jun
17 19-Jun
18 26-Jun
19 3-Jul
20 10-Jul
21 17-Jul
22 24-Jul
23 31-Jul
24 7-Aug
25 14-Aug
26 21-Aug
27 28-Aug
28 4-Sep
29 11-Sep
30 18-Sep
31 25-Sep
32 2-Oct
33 9-Oct
34 16-Oct
Progress Report Due
Exam Period
Work Contract - Mr Isa (GFM)
Partial Draft Dissertation Due
Dissertation Due

# Appendix C: CNN Code for Training and Development

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils.data
import torch.nn.functional as F
import torchvision
from torchvision import transforms
from PIL import Image, ImageFile

class cnnNet(nn.Module):
    def __init__(self, num_classes=6):
        super(cnnNet, self).__init__()
        self.features = nn.Sequential(
                nn.Conv2d(3, 32, kernel_size=10, stride=10, padding=0),
                nn.BatchNorm2d(32),
                nn.ReLU(),
                #nn.MaxPool2d(kernel_size=3, stride=2),

                nn.Conv2d(32, 32, kernel_size=3, padding=0),
                nn.BatchNorm2d(32),
                nn.ReLU(),
                #nn.MaxPool2d(kernel_size=3, stride=2),

                nn.Conv2d(32, 32, kernel_size=3, padding=0),
                nn.BatchNorm2d(32),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=3, stride=2),

                nn.Conv2d(32, 64, kernel_size=3, padding=0),
                nn.BatchNorm2d(64),
                nn.ReLU(),

                nn.Conv2d(64, 128, kernel_size=3, padding=0),
                nn.BatchNorm2d(128),
                nn.ReLU(),

                nn.Conv2d(128, 128, kernel_size=3, padding=0),
                nn.BatchNorm2d(128),
                nn.ReLU(),
                nn.MaxPool2d(kernel_size=3, stride=2),)

        self.avgpool = nn.AdaptiveAvgPool2d((6,6))

        self.classifier = nn.Sequential(
                nn.Dropout(p=0.2), #randomly zero out 20% of data points
                nn.Linear(128*6*6, 1152), # Fully connected
                nn.ReLU(),
                nn.Dropout(p=0.2), #randomly zero out 20% of data points
                nn.Linear(1152, 1152),   # Fully connected
                nn.ReLU(),
                nn.Linear(1152, num_classes)) # Fully connected

        def forward(self, x):
            x = self.features(x)
            x = self.avgpool(x)
            x = torch.flatten(x, 1)
            x = self.classifier(x)
            return x

        def train(model, optimizer, loss_fn, train_loader, val_loader, epochs=250, device="cuda"):
            for epoch in range(epochs):
                training_loss = 0.0
                valid_loss = 0.0
                model.train()
                for batch in train_loader:
                    optimizer.zero_grad()
                    inputs, targets = batch
```

```
            inputs = inputs.to(device)
            targets = targets.to(device)
            output = model(inputs)
            loss = loss_fn(output, targets)
            loss.backward()
            optimizer.Adam()
            training_loss += loss.data.item() * inputs.size(0)
        training_loss /= len(train_loader.dataset)

        model.eval()
        num_correct = 0
        num_examples = 0
        for batch in val_loader:
            inputs, targets = batch
            inputs = inputs.to(device)
            output = model(inputs)
            targets = targets.to(device)
            loss = loss_fn(output,targets)
            valid_loss += loss.data.item() * inputs.size(0)
            correct = torch.eq(torch.max(F.softmax(output), dim=1)[1], targets).view(-1)
            num_correct += torch.sum(correct).item()
            num_examples += correct.shape[0]
        valid_loss /= len(val_loader.dataset)

        print('Epoch: {}, Training Loss: {:.2f}, Validation Loss: {:.2f}, accuracy = {:.2f}'.format(epoch, training_loss,
        valid_loss, num_correct / num_examples))

def check_image(path):
    try:
        im = Image.open(path)
        return True
    except:
        return False

img_transforms = transforms.Compose([
    transforms.Resize((480,480)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0 225] )
                    ])

train_data_path = "./train/"
train_data = torchvision.datasets.ImageFolder(root=train_data_path,transform=img_transforms, is_valid_file=check_image)
val_data_path = "./val/"
val_data = torchvision.datasets.ImageFolder(root=val_data_path,transform=img_transforms, is_valid_file=check_image)
batch_size=64
train_data_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size,shuffle=True)
val_data_loader  = torch.utils.data.DataLoader(val_data, batch_size=batch_size, shuffle=True)

if torch.cuda.is_available():
    device = torch.device("cuda")
else:
    device = torch.device("cpu")
```

Credit for the inspiration of this code must be given to Ian Pointer as parts are exerts from the book, ‘Programming PyTorch for Deep Learning: Creating and Deploying Deep Learning Applications’ published in 2019.

# Appendix D: ORB-SLAM in MATLAB

```
% Set random seed for reproducibility
rng(0);

% Create a cameraIntrinsics object to store the camera intrinsic parameters.
% The intrinsics for the dataset can be found at the following page:
% https://vision.in.tum.de/data/datasets/rgbd-dataset/file_formats
% Note that the images in the dataset are already undistorted, hence there
% is no need to specify the distortion coefficients.
focalLength    = [535.4, 539.2];    % in units of pixels
principalPoint = [320.1, 247.6];    % in units of pixels
imageSize      = size(currI,[1 2]);  % in units of pixels
intrinsics     = cameraIntrinsics(focalLength, principalPoint, imageSize);

% Detect and extract ORB features
scaleFactor = 1.2;
numLevels   = 8;
[preFeatures, prePoints] = helperDetectAndExtractFeatures(currI, scaleFactor, numLevels);

currFrameIdx = currFrameIdx + 1;
firstI       = currI; % Preserve the first frame

isMapInitialized  = false;

% Map initialization loop
while ~isMapInitialized && currFrameIdx < numel(imds.Files)
    currI = readimage(imds, currFrameIdx);

    [currFeatures, currPoints] = helperDetectAndExtractFeatures(currI, scaleFactor, numLevels);

    currFrameIdx = currFrameIdx + 1;

    % Find putative feature matches
    indexPairs = matchFeatures(preFeatures, currFeatures, 'Unique', true, ...
        'MaxRatio', 0.9, 'MatchThreshold', 40);

    preMatchedPoints  = prePoints(indexPairs(:,1),:);
    currMatchedPoints = currPoints(indexPairs(:,2),:);

    % If not enough matches are found, check the next frame
    minMatches = 100;
    if size(indexPairs, 1) < minMatches
        continue
    end

    preMatchedPoints  = prePoints(indexPairs(:,1),:);
    currMatchedPoints = currPoints(indexPairs(:,2),:);

    % Compute homography and evaluate reconstruction
    [tformH, scoreH, inliersIdxH] = helperComputeHomography(preMatchedPoints, currMatchedPoints);

    % Compute fundamental matrix and evaluate reconstruction
    [tformF, scoreF, inliersIdxF] = helperComputeFundamentalMatrix(preMatchedPoints, currMatchedPoints);

    % Select the model based on a heuristic
    ratio = scoreH/(scoreH + scoreF);
    ratioThreshold = 0.45;
    if ratio > ratioThreshold
        inlierTformIdx = inliersIdxH;
        tform          = tformH;
    else
        inlierTformIdx = inliersIdxF;
        tform          = tformF;
    end

    % Computes the camera location up to scale. Use half of the
```

```
    % points to reduce computation
    inlierPrePoints  = preMatchedPoints(inlierTformIdx);
    inlierCurrPoints = currMatchedPoints(inlierTformIdx);
    [relOrient, relLoc, validFraction] = relativeCameraPose(tform, intrinsics, ...
        inlierPrePoints(1:2:end), inlierCurrPoints(1:2:end));

    % If not enough inliers are found, move to the next frame
    if validFraction < 0.9 || numel(size(relOrient))==3
        continue
    end

    % Triangulate two views to obtain 3-D map points
    relPose = rigid3d(relOrient, relLoc);
    minParallax = 3; % In degrees
    [isValid, xyzWorldPoints, inlierTriangulationIdx] = helperTriangulateTwoFrames(...
        rigid3d, relPose, inlierPrePoints, inlierCurrPoints, intrinsics, minParallax);

    if ~isValid
        continue
    end

    % Get the original index of features in the two key frames
    indexPairs = indexPairs(inlierTformIdx(inlierTriangulationIdx),:);

    isMapInitialized = true;

    disp(['Map initialized with frame 1 and frame ', num2str(currFrameIdx-1)])
end % End of map initialization loop

if isMapInitialized
    close(himage.Parent.Parent); % Close the previous figure
    % Show matched features
    hfeature = showMatchedFeatures(firstI, currI, prePoints(indexPairs(:,1)), ...
        currPoints(indexPairs(:, 2)), 'Montage');
else
    error('Unable to initialize map.')
end

% Create an empty imageviewset object to store key frames
vSetKeyFrames = imageviewset;

% Create an empty worldpointset object to store 3-D map points
mapPointSet   = worldpointset;

% Create a helperViewDirectionAndDepth object to store view direction and depth
directionAndDepth = helperViewDirectionAndDepth(size(xyzWorldPoints, 1));

% Add the first key frame. Place the camera associated with the first
% key frame at the origin, oriented along the Z-axis
preViewId     = 1;
vSetKeyFrames = addView(vSetKeyFrames, preViewId, rigid3d, 'Points', prePoints,...
    'Features', preFeatures.Features);

% Add the second key frame
currViewId    = 2;
vSetKeyFrames = addView(vSetKeyFrames, currViewId, relPose, 'Points', currPoints,...
    'Features', currFeatures.Features);

% Add connection between the first and the second key frame
vSetKeyFrames = addConnection(vSetKeyFrames, preViewId, currViewId, relPose, 'Matches', indexPairs);

% Add 3-D map points
[mapPointSet, newPointIdx] = addWorldPoints(mapPointSet, xyzWorldPoints);

% Add observations of the map points
preLocations   = prePoints.Location;
currLocations  = currPoints.Location;
preScales      = prePoints.Scale;
currScales     = currPoints.Scale;
```

```
% Add image points corresponding to the map points in the first key frame
mapPointSet   = addCorrespondences(mapPointSet, preViewId, newPointIdx, indexPairs(:,1));

% Add image points corresponding to the map points in the second key frame
mapPointSet   = addCorrespondences(mapPointSet, currViewId, newPointIdx, indexPairs(:,2));

% Run full bundle adjustment on the first two key frames
tracks       = findTracks(vSetKeyFrames);
cameraPoses  = poses(vSetKeyFrames);

[refinedPoints, refinedAbsPoses] = bundleAdjustment(xyzWorldPoints, tracks, ...
    cameraPoses, intrinsics, 'FixedViewIDs', 1, ...
    'PointsUndistorted', true, 'AbsoluteTolerance', 1e-7,...
    'RelativeTolerance', 1e-15, 'MaxIteration', 50);

% Scale the map and the camera pose using the median depth of map points
medianDepth   = median(vecnorm(refinedPoints.'));
refinedPoints = refinedPoints / medianDepth;

refinedAbsPoses.AbsolutePose(currViewId).Translation = ...
    refinedAbsPoses.AbsolutePose(currViewId).Translation / medianDepth;
relPose.Translation = relPose.Translation/medianDepth;

% Update key frames with the refined poses
vSetKeyFrames = updateView(vSetKeyFrames, refinedAbsPoses);
vSetKeyFrames = updateConnection(vSetKeyFrames, preViewId, currViewId, relPose);

% Update map points with the refined positions
mapPointSet   = updateWorldPoints(mapPointSet, newPointIdx, refinedPoints);

% Update view direction and depth
directionAndDepth = update(directionAndDepth, mapPointSet, vSetKeyFrames.Views, newPointIdx, true);

% Visualize matched features in the current frame
close(hfeature.Parent.Parent);
featurePlot   = helperVisualizeMatchedFeatures(currI, currPoints(indexPairs(:,2)));

% Visualize initial map points and camera trajectory
mapPlot       = helperVisualizeMotionAndStructure(vSetKeyFrames, mapPointSet);

% Show legend
showLegend(mapPlot);

% ViewId of the current key frame
currKeyFrameId    = currViewId;

% ViewId of the last key frame
lastKeyFrameId    = currViewId;

% ViewId of the reference key frame that has the most co-visible
% map points with the current key frame
refKeyFrameId     = currViewId;

% Index of the last key frame in the input image sequence
lastKeyFrameIdx   = currFrameIdx - 1;

% Indices of all the key frames in the input image sequence
addedFramesIdx    = [1; lastKeyFrameIdx];

isLoopClosed      = false;

% Main loop
while ~isLoopClosed && currFrameIdx < numel(imds.Files)
    currI = readimage(imds, currFrameIdx);

    [currFeatures, currPoints] = helperDetectAndExtractFeatures(currI, scaleFactor, numLevels);

    % Track the last key frame
```

```
    % mapPointsIdx:   Indices of the map points observed in the current frame
    % featureIdx:     Indices of the corresponding feature points in the
    %                 current frame
    [currPose, mapPointsIdx, featureIdx] = helperTrackLastKeyFrame(mapPointSet, ...
        vSetKeyFrames.Views, currFeatures, currPoints, lastKeyFrameId, intrinsics, scaleFactor);

    % Track the local map
    % refKeyFrameId:      ViewId of the reference key frame that has the most
    %                     co-visible map points with the current frame
    % localKeyFrameIds:   ViewId of the connected key frames of the current frame
    [refKeyFrameId, localKeyFrameIds, currPose, mapPointsIdx, featureIdx] = ...
        helperTrackLocalMap(mapPointSet, directionAndDepth, vSetKeyFrames, mapPointsIdx, ...
        featureIdx, currPose, currFeatures, currPoints, intrinsics, scaleFactor, numLevels);

    % Check if the current frame is a key frame.
    % A frame is a key frame if both of the following conditions are satisfied:
    %
    % 1. At least 20 frames have passed since the last key frame or the
    %    current frame tracks fewer than 80 map points
    % 2. The map points tracked by the current frame are fewer than 90% of
    %    points tracked by the reference key frame
    isKeyFrame = helperIsKeyFrame(mapPointSet, refKeyFrameId, lastKeyFrameIdx, ...
        currFrameIdx, mapPointsIdx);

    % Visualize matched features
    updatePlot(featurePlot, currI, currPoints(featureIdx));

    if ~isKeyFrame
        currFrameIdx = currFrameIdx + 1;
        continue
    end

    % Update current key frame ID
    currKeyFrameId  = currKeyFrameId + 1;

% Add the new key frame
    [mapPointSet, vSetKeyFrames] = helperAddNewKeyFrame(mapPointSet, vSetKeyFrames, ...
        currPose, currFeatures, currPoints, mapPointsIdx, featureIdx, localKeyFrameIds);

    % Remove outlier map points that are observed in fewer than 3 key frames
    [mapPointSet, directionAndDepth, mapPointsIdx] = helperCullRecentMapPoints(mapPointSet,
directionAndDepth, mapPointsIdx, newPointIdx);

    % Create new map points by triangulation
    minNumMatches = 20;
    minParallax = 3;
    [mapPointSet, vSetKeyFrames, newPointIdx] = helperCreateNewMapPoints(mapPointSet, vSetKeyFrames, ...
        currKeyFrameId, intrinsics, scaleFactor, minNumMatches, minParallax);

    % Update view direction and depth
    directionAndDepth = update(directionAndDepth, mapPointSet, vSetKeyFrames.Views, [mapPointsIdx;
newPointIdx], true);

    % Local bundle adjustment
    [mapPointSet, directionAndDepth, vSetKeyFrames, newPointIdx] =
helperLocalBundleAdjustment(mapPointSet, directionAndDepth, vSetKeyFrames, ...
        currKeyFrameId, intrinsics, newPointIdx);

    % Visualize 3D world points and camera trajectory
    updatePlot(mapPlot, vSetKeyFrames, mapPointSet);

% Initialize the loop closure database
    if currKeyFrameId == 3
        % Load the bag of features data created offline
        bofData         = load('bagOfFeaturesData.mat');
        loopDatabase    = invertedImageIndex(bofData.bof);
        loopCandidates  = [1; 2];

    % Check loop closure after some key frames have been created
```

```
    elseif currKeyFrameId > 20

        % Minimum number of feature matches of loop edges
        loopEdgeNumMatches = 50;

        % Detect possible loop closure key frame candidates
        [isDetected, validLoopCandidates] = helperCheckLoopClosure(vSetKeyFrames, currKeyFrameId, ...
            loopDatabase, currI, loopCandidates, loopEdgeNumMatches);

        if isDetected
            % Add loop closure connections
            [isLoopClosed, mapPointSet, vSetKeyFrames] = helperAddLoopConnections(...
                mapPointSet, vSetKeyFrames, validLoopCandidates, currKeyFrameId, ...
                currFeatures, currPoints, intrinsics, scaleFactor, loopEdgeNumMatches);
        end
    end

    % If no loop closure is detected, add the image into the database
    if ~isLoopClosed
        currds = imageDatastore(imds.Files{currFrameIdx});
        addImages(loopDatabase, currds, 'Verbose', false);
        loopCandidates= [loopCandidates; currKeyFrameId]; %#ok<AGROW>
    end

    % Update IDs and indices
    lastKeyFrameId  = currKeyFrameId;
    lastKeyFrameIdx = currFrameIdx;
    addedFramesIdx  = [addedFramesIdx; currFrameIdx]; %#ok<AGROW>
    currFrameIdx  = currFrameIdx + 1;
end % End of main loop
% Optimize the poses
vSetKeyFramesOptim = optimizePoses(vSetKeyFrames, minNumMatches, 'Tolerance', 1e-16, 'Verbose', true);

% Update map points after optimizing the poses
mapPointSet = helperUpdateGlobalMap(mapPointSet, directionAndDepth, ...
    vSetKeyFrames, vSetKeyFramesOptim);

updatePlot(mapPlot, vSetKeyFrames, mapPointSet);

% Plot the optimized camera trajectory
optimizedPoses  = poses(vSetKeyFramesOptim);
plotOptimizedTrajectory(mapPlot, optimizedPoses)

% Update legend
showLegend(mapPlot);

% Load ground truth
gTruthData = load('orbslamGroundTruth.mat');
gTruth     = gTruthData.gTruth;

% Plot the actual camera trajectory
plotActualTrajectory(mapPlot, gTruth(addedFramesIdx), optimizedPoses);

% Show legend
showLegend(mapPlot);

% Evaluate tracking accuracy
helperEstimateTrajectoryError(gTruth(addedFramesIdx), optimizedPoses);

function [features, validPoints] = helperDetectAndExtractFeatures(Irgb, ...
    scaleFactor, numLevels, varargin)

numPoints   = 1000;

% In this example, the images are already undistorted. In a general
% workflow, uncomment the following code to undistort the images.
%
% if nargin > 3
%     intrinsics = varargin{1};
```

```
% end
% Irgb  = undistortImage(Irgb, intrinsics);

% Detect ORB features
Igray  = rgb2gray(Irgb);

points = detectORBFeatures(Igray, 'ScaleFactor', scaleFactor, 'NumLevels', numLevels);

% Select a subset of features, uniformly distributed throughout the image
points = selectUniform(points, numPoints, size(Igray, 1:2));

% Extract features
[features, validPoints] = extractFeatures(Igray, points);
end

function [H, score, inliersIndex] = helperComputeHomography(matchedPoints1, matchedPoints2)

[H, inliersLogicalIndex] = estimateGeometricTransform2D( ...
    matchedPoints1, matchedPoints2, 'projective', ...
    'MaxNumTrials', 1e3, 'MaxDistance', 4, 'Confidence', 90);

inlierPoints1 = matchedPoints1(inliersLogicalIndex);
inlierPoints2 = matchedPoints2(inliersLogicalIndex);

inliersIndex  = find(inliersLogicalIndex);

locations1 = inlierPoints1.Location;
locations2 = inlierPoints2.Location;
xy1In2     = transformPointsForward(H, locations1);
xy2In1     = transformPointsInverse(H, locations2);
error1in2  = sum((locations2 - xy1In2).^2, 2);
error2in1  = sum((locations1 - xy2In1).^2, 2);

outlierThreshold = 6;

score = sum(max(outlierThreshold-error1in2, 0)) + ...
    sum(max(outlierThreshold-error2in1, 0));
end

function [F, score, inliersIndex] = helperComputeFundamentalMatrix(matchedPoints1, matchedPoints2)

[F, inliersLogicalIndex]   = estimateFundamentalMatrix( ...
    matchedPoints1, matchedPoints2, 'Method','RANSAC',...
    'NumTrials', 1e3, 'DistanceThreshold', 0.01);

inlierPoints1 = matchedPoints1(inliersLogicalIndex);
inlierPoints2 = matchedPoints2(inliersLogicalIndex);

inliersIndex  = find(inliersLogicalIndex);

locations1    = inlierPoints1.Location;
locations2    = inlierPoints2.Location;

% Distance from points to epipolar line
lineIn1   = epipolarLine(F', locations2);
error2in1 = (sum([locations1, ones(size(locations1, 1),1)].* lineIn1, 2)).^2 ...
    ./ sum(lineIn1(:,1:2).^2, 2);
lineIn2   = epipolarLine(F, locations1);
error1in2 = (sum([locations2, ones(size(locations2, 1),1)].* lineIn2, 2)).^2 ...
    ./ sum(lineIn2(:,1:2).^2, 2);

outlierThreshold = 4;

score = sum(max(outlierThreshold-error1in2, 0)) + ...
    sum(max(outlierThreshold-error2in1, 0));

end

function [isValid, xyzPoints, inlierIdx] = helperTriangulateTwoFrames(...
```

```
    pose1, pose2, matchedPoints1, matchedPoints2, intrinsics, minParallax)

[R1, t1]   = cameraPoseToExtrinsics(pose1.Rotation, pose1.Translation);
camMatrix1 = cameraMatrix(intrinsics, R1, t1);

[R2, t2]   = cameraPoseToExtrinsics(pose2.Rotation, pose2.Translation);
camMatrix2 = cameraMatrix(intrinsics, R2, t2);

[xyzPoints, reprojectionErrors, isInFront] = triangulate(matchedPoints1, ...
    matchedPoints2, camMatrix1, camMatrix2);

% Filter points by view direction and reprojection error
minReprojError = 1;
inlierIdx  = isInFront & reprojectionErrors < minReprojError;
xyzPoints  = xyzPoints(inlierIdx ,:);

% A good two-view with significant parallax
ray1       = xyzPoints - pose1.Translation;
ray2       = xyzPoints - pose2.Translation;
cosAngle   = sum(ray1 .* ray2, 2) ./ (vecnorm(ray1, 2, 2) .* vecnorm(ray2, 2, 2));

% Check parallax
isValid = all(cosAngle < cosd(minParallax) & cosAngle>0);
end

function isKeyFrame = helperIsKeyFrame(mapPoints, ...
    refKeyFrameId, lastKeyFrameIndex, currFrameIndex, mapPointsIndices)

numPointsRefKeyFrame = numel(findWorldPointsInView(mapPoints, refKeyFrameId));

% More than 20 frames have passed from last key frame insertion
tooManyNonKeyFrames = currFrameIndex >= lastKeyFrameIndex + 20;

% Track less than 90 map points
tooFewMapPoints     = numel(mapPointsIndices) < 90;

% Tracked map points are fewer than 90% of points tracked by
% the reference key frame
tooFewTrackedPoints = numel(mapPointsIndices) < 0.9 * numPointsRefKeyFrame;

isKeyFrame = (tooManyNonKeyFrames || tooFewMapPoints) && tooFewTrackedPoints;
end

function [mapPointSet, directionAndDepth, mapPointsIdx] = helperCullRecentMapPoints(mapPointSet,
directionAndDepth, mapPointsIdx, newPointIdx)
outlierIdx    = setdiff(newPointIdx, mapPointsIdx);
if ~isempty(outlierIdx)
    mapPointSet   = removeWorldPoints(mapPointSet, outlierIdx);
    directionAndDepth = remove(directionAndDepth, outlierIdx);
    mapPointsIdx  = mapPointsIdx - arrayfun(@(x) nnz(x>outlierIdx), mapPointsIdx);
end
end

function rmse = helperEstimateTrajectoryError(gTruth, cameraPoses)
locations       = vertcat(cameraPoses.AbsolutePose.Translation);
gLocations      = vertcat(gTruth.Translation);
scale           = median(vecnorm(gLocations, 2, 2))/ median(vecnorm(locations, 2, 2));
scaledLocations = locations * scale;

rmse = sqrt(mean( sum((scaledLocations - gLocations).^2, 2) ));
disp(['Absolute RMSE for key frame trajectory (m): ', num2str(rmse)]);
end

function [mapPointSet, directionAndDepth] = helperUpdateGlobalMap(...
    mapPointSet, directionAndDepth, vSetKeyFrames, vSetKeyFramesOptim)
%helperUpdateGlobalMap update map points after pose graph optimization
posesOld     = vSetKeyFrames.Views.AbsolutePose;
posesNew     = vSetKeyFramesOptim.Views.AbsolutePose;
positionsOld = mapPointSet.WorldPoints;
```

```
positionsNew = positionsOld;
indices = 1:mapPointSet.Count;

% Update world location of each map point based on the new absolute pose of
% the corresponding major view
for i = 1: mapPointSet.Count
    majorViewIds = directionAndDepth.MajorViewId(i);
    tform = posesOld(majorViewIds).T \ posesNew(majorViewIds).T ;
    positionsNew(i, :) = positionsOld(i, :) * tform(1:3,1:3) + tform(4, 1:3);
end
mapPointSet = updateWorldPoints(mapPointSet, indices, positionsNew);
end
```

Full acknowledgement must be given to MathWorks for the conversion of ORB-SLAM into MATLAB. The code can be found at:

https://au.mathworks.com/help/vision/ug/monocular-visual-simultaneous-localization-and-mapping.html

Full acknowledgement must also be given to the original authors of ORB-SLAM, Raul Mur-Artal, J. M. M. Montiel & Juan D. Tardos.

# Appendix E: CEKF-SLAM

```
function data= cekfslam(lm, wp)
%function data= cekfslam(lm, wp)
%
% INPUTS:
%   lm - set of landmarks
%   wp - set of waypoints
%
% OUTPUTS:
%   data - a data structure containing:
%       data.i          : number of states
%       data.true       : the vehicle 'true'-path (ie, where the vehicle *actually* went)
%       data.path       : the vehicle path estimate (ie, where SLAM estimates the vehicle went)
%       data.state(k).x: the SLAM state vector at time k
%       data.state(k).P: the diagonals of the SLAM covariance matrix at time k
%       data.finalx     : the estimated states when a simulation is done
%       data.finalcov   : the estimated states covariance when a simulation is done
%       data.finalcorr  : the estimated states correlations when a simulation is done
%
%To run this simulator:
%   1. load loop902.mat to the workspace
%   2. run "data = cekfslam(lm,wp)" in the command window
%
% NOTES:
%   This program is a compressed extended Kalman filter(CEKF) based SLAM simulator.
%   To use, create a set of landmarks and vehicle waypoints (ie, waypoints for the desired vehicle path).
%   The program 'frontend.m' may be used to create this simulated environment - type
%   'help frontend' for more information.
%       The configuration of the simulator is managed by the script file
%   'configfile.m'. To alter the parameters of the vehicle, sensors, etc
%   adjust this file. There are also several switches that control certain
%   filter options.
%
% Thanks to Tim Bailey and Juan Nieto 2004.Version 1.0
%
% Zhang Haiqiang 2007-11-22
%
% ALGORITHM USED:
%   This program adopts Compressed Extended Kalman Filter to SLAM, and
%   when SWITCH_BATCH_UPDATE = 0, I used the sparsity of Observation
%   Jacobian Matrix to reduce computation complexity.
%
% MODELS:
%   The motion model is setup to be like a Pioneer3-AT robot(skid-steering),
%   The observation mode are setup to be like a LMS200.
%
% NOTES:
%   It is VERY important that the data association should always be
%   correct, if wrong data association occurs, the whole state will
%   probably diverge.
%
% Zhang Haiqiang 2007-11-20
% Zhang Haiqiang 2007-5-11
%

format compact
configfile;

% setup plots
if SWITCH_ANIMATION_ON == 1
    scrsz= get(0,'ScreenSize')*0.75;
    fig=figure('Position',[0 0 scrsz(3) scrsz(4)]);
    plot(lm(1,:),lm(2,:),'b*')
    hold on, axis equal, grid on
    %plot(wp(1,:),wp(2,:), 'g', wp(1,:),wp(2,:),'g.')
    MAXX = max([max(lm(1,:)) max(wp(1,:))]);
    MINX = min([min(lm(1,:)) min(wp(1,:))]);
    MAXY = max([max(lm(2,:)) max(wp(2,:))]);
```

```
   MINY = min([min(lm(2,:)) min(wp(2,:))]);
   axis([MINX-10 MAXX+10 MINY-10 MAXY+10])
   xlabel('metres'), ylabel('metres')
   set(fig, 'name', ' Compressed EKF-SLAM via Pioneer3-AT & LMS200')
   h= setup_animations;
   veh= 0.5*[1 1 -1 -1; 1 -1 1 -1]; % vehicle animation
   %plines=[]; % for laser line animation
   pcount=0;
end

%
% zhq: 'stem' the diag of the state covariance matrix
%
if SWITCH_VISULIZE_THE_EVOLUTION_OF_COVARIANCE_DIAG == 1
   fig_DiagOfStateCovMatrix = figure;
   set( fig_DiagOfStateCovMatrix, 'Name', 'diag of the state covariance matrix');
   axes_DiagOfStateCovMatrix = axes;
end
%%%

% initialise states
global vtrue XA PA XB PB PAB
vtrue= zeros(3,1);% true pose of the vehicle
XA = zeros(3,1); % part A of SLAM state
PA = zeros(3);    %
XB = zeros(1);    % part B of SLAM state
PB = zeros(1);    %
PAB =zeros(1);   % cross covariance of part A and B

% CEKF auxiliary parameters
global PsiXB OmegaPB PhiPAB
PsiXB = zeros(1);
OmegaPB = zeros(1);
PhiPAB = zeros(1);
% CEKF
global g_current_ala_center g_current_ala_center_cov
g_current_ala_center = zeros(2,1);
g_current_ala_center_cov = zeros(2);

% CEKF predict auxiliary parameter
global JXA
JXA= zeros(1);

%
global GDATA

% initialise other variables and constants
dt= DT_CONTROLS; % change in time between predicts
dtsum= 0; % change in time since last observation
iwp= 1; % index to first waypoint
W = 0; % initial rotation speed

% time lapsed since the vehicle started
g_sim_time = 0;

if SWITCH_OFFLINE_DATA_ON == 1  || SWITCH_ANIMATION_ON == 1
   initialise_store(); % stored data for off-line
end

QE= Q; RE= R;
if SWITCH_INFLATE_NOISE, QE= 2*Q; RE= 8*R; end % inflate estimated noises (ie, add stabilising noise)
if SWITCH_SEED_RANDOM, randn('state',SWITCH_SEED_RANDOM), end

%
if SWITCH_PROFILE, profile on -detail mmex, end

% main loop
counter = 0;
frame_counter= 0;
while iwp ~= 0
```

```
    g_sim_time= g_sim_time + dt;
    counter = counter+1;

   % zhq: visulize the diag of the state covariance matrix
   if SWITCH_VISULIZE_THE_EVOLUTION_OF_COVARIANCE_DIAG == 1
      dP = diag(PA);
      stem( axes_DiagOfStateCovMatrix, 1:length(dP), dP );
      set( axes_DiagOfStateCovMatrix, 'XLim', [1  length(dP)], 'XTick', 1: length(dP), 'YLim', [ 0 ceil(max(dP)+1e-9)],  'YTick', 0:
ceil(max(dP)+1e-9)/10: ceil(max(dP)+1e-9) );
      if SWITCH_ANIMATION_ON == 0, drawnow, end
   end
   %%%

   % compute true data
   [W,iwp] = compute_rotationspeed(wp, iwp, AT_WAYPOINT, W, MAXW, dt);
   if iwp==0 && NUMBER_LOOPS > 1
      iwp=1;
      NUMBER_LOOPS= NUMBER_LOOPS-1;
   end % perform loops: if final waypoint reached, go back to first

   vtrue = vehicle_model(vtrue, V, W,dt);
   [Vn, Wn] = add_control_noise (V,W,Q,SWITCH_CONTROL_NOISE);

   % CEKF predict step
   predict(Vn,Wn,QE, dt);

   % CEKF update step
   dtsum= dtsum + dt;
   %if dtsum >= DT_OBSERVE %zhq: dtsum >= DT_OBSERVE - 1e-6 is better
   if dtsum >= DT_OBSERVE - 1e-6 || iwp == 0
      dtsum= 0;
      [z]= get_observations(vtrue, lm, MAX_RANGE);
      z= add_observation_noise(z,R, SWITCH_SENSOR_NOISE);

      % try your best to make sure the data associate works correctly!
      [zf,idf, zn]= data_associate(XA,PA,z,RE, GATE_REJECT, GATE_AUGMENT);
      check_data_association(idf);

      if size(zf,1) > 0
         update(zf,RE,idf,SWITCH_BATCH_UPDATE);
      else
         if size(PAB,1) ~= 1
            if size(PhiPAB,1) ~= 1
               PhiPAB=JXA*PhiPAB;
            else
               PAB=JXA*PAB;
            end
            JXA=zeros(1);
         end
      end

      if size(zn,1) >0, augment(zn,RE); end

      if switch_active_local_area(RESTRICING_ALA_R) ~= 0
         full_states_update();
         reassign_states(ENVIRONING_ALA_R);
      end
   end

   % simulation is almost finished
   if  iwp == 0, full_states_update(); end

   % offline data store
   if SWITCH_OFFLINE_DATA_ON == 1, store_data(); end

   % plots
   if SWITCH_ANIMATION_ON == 1
      xt= TransformToGlobal(veh,vtrue);
      xv= TransformToGlobal(veh,XA(1:3));
      set(h.xt, 'xdata', xt(1,:), 'ydata', xt(2,:))
```

```
      set(h.xv, 'xdata', xv(1,:), 'ydata', xv(2,:))
      set(h.xfa, 'xdata', XA(4:2:end), 'ydata', XA(5:2:end))
      if size(XB,1) ~= 1, set(h.xfb, 'xdata', XB(1:2:end), 'ydata', XB(2:2:end)), end

      ptmp= make_covariance_ellipses(XA(1:3),PA(1:3,1:3));
      pcova(:,1:size(ptmp,2))= ptmp;
      if dtsum==0
         set(h.cova, 'xdata', pcova(1,:), 'ydata', pcova(2,:))
         pcount= pcount+1;
         if pcount == 15
            set(h.pth, 'xdata', GDATA.path(1,1:GDATA.i), 'ydata', GDATA.path(2,1:GDATA.i))
            set(h.pthtrue, 'xdata', GDATA.true(1,1:GDATA.i), 'ydata', GDATA.true(2,1:GDATA.i))
            pcount=0;
         end
         if ~isempty(z)
            plines= make_laser_lines (z,XA(1:3));
            set(h.obs, 'xdata', plines(1,:), 'ydata', plines(2,:))
            pcova= make_covariance_ellipses(XA,PA);
         end

         %set(h.timeelapsed, 'String', num2str(g_sim_time))

         if size(XB,1) ~= 1 && size(OmegaPB,1) == 1
            pcovb = make_covariance_ellipses_xb(XB,PB);
            set(h.covb, 'xdata', pcovb(1,:), 'ydata', pcovb(2,:))
         end
      end

      [strict_circle, environ_circle] = make_range_circles(RESTRICING_ALA_R, ENVIRONING_ALA_R);
      set(h.restrict, 'xdata', strict_circle(1,:), 'ydata', strict_circle(2,:))
      set(h.environ, 'xdata', environ_circle(1,:), 'ydata', environ_circle(2,:))

      drawnow
      if SWITCH_RECORD_THE_PROCESS==1 && mod(counter,30) == 1,
         frame_counter= frame_counter+1;
         FRAMES(:,frame_counter)=getframe;
      end
   end
end %end of while

if SWITCH_OFFLINE_DATA_ON == 1, finalise_data(); end

if SWITCH_ANIMATION_ON == 1
   set(h.pth, 'xdata', GDATA.path(1,:), 'ydata', GDATA.path(2,:))
   set(h.pthtrue, 'xdata', GDATA.true(1,:), 'ydata', GDATA.true(2,:))  % zhq-draw true path
   set(h.timeelapsed, 'String', num2str(g_sim_time))
   drawnow
   if SWITCH_RECORD_THE_PROCESS == 1
      frame_counter= frame_counter+1;
      FRAMES(:,frame_counter)=getframe;
      movie2avi(FRAMES,'a.avi','quality',100);
   end
end


if SWITCH_PROFILE, profile report, end

GDATA.finalx = [XA; XB];
GDATA.finalcov = [PA PAB; PAB' PB];
vari = diag(GDATA.finalcov).^(1/2);
GDATA.finalcorr = GDATA.finalcov./ (vari*vari');

data= GDATA;
clear global vtrue XA PA XB PB PAB PsiXB OmegaPB PhiPAB
clear global g_current_ala_center g_current_ala_center_cov
clear global JXA GDATA

%
%
```

```
function h= setup_animations()
h.xt= patch(0,0,'b','erasemode','xor'); % vehicle true
h.xv= patch(0,0,'r','erasemode','xor'); % vehicle estimate
h.pth= plot(0,0,'r.','markersize',2,'erasemode','background'); % vehicle path estimate
h.pthtrue= plot(0,0,'b.','markersize',2,'erasemode','background'); % vehicle path true
h.obs= plot(0,0,'k','erasemode','xor'); % observations
h.timeelapsed = annotation('textbox',[0.89 0.9 0.1 0.05]);
h.xfa= plot(0,0,'r+','erasemode','xor'); % estimated features of part A
h.cova= plot(0,0,'r','erasemode','xor'); % covariance ellipses
h.xfb= plot(0,0,'k+','erasemode','xor'); % estimated features of part B
h.covb= plot(0,0,'k','erasemode','xor'); % covariance ellipses
h.restrict= plot(0,0,'k','erasemode','xor', 'LineWidth',1, 'LineStyle','-');
h.environ= plot(0,0,'k','erasemode','xor', 'LineWidth',2, 'LineStyle','-');
%
%

function p= make_laser_lines (rb,xv)
% compute set of line segments for laser range-bearing measurements
if isempty(rb), p=[]; return, end
len= size(rb,2);
lnes(1,:)= zeros(1,len)+ xv(1);
lnes(2,:)= zeros(1,len)+ xv(2);
lnes(3:4,:)= TransformToGlobal([rb(1,:).*cos(rb(2,:)); rb(1,:).*sin(rb(2,:))], xv);
p= line_plot_conversion (lnes);

%
%

function p= make_covariance_ellipses(x,P)
% compute ellipses for plotting state covariances
N= 10;
inc= 2*pi/N;
phi= 0:inc:2*pi;

lenx= length(x);
lenf= (lenx-3)/2;
p= zeros (2,(lenf+1)*(N+2));

ii=1:N+2;
p(:,ii)= make_ellipse(x(1:2), P(1:2,1:2), 2, phi);

ctr= N+3;
for i=1:lenf
   ii= ctr:(ctr+N+1);
   jj= 2+2*i; jj= jj:jj+1;

   p(:,ii)= make_ellipse(x(jj), P(jj,jj), 2, phi);
   ctr= ctr+N+2;
end

%
%

function p= make_ellipse(x,P,s, phi)
% make a single 2-D ellipse of s-sigmas over phi angle intervals
s=2.448; %corresponding cdf is 0.95
r= sqrtm(P);
a= s*r*[cos(phi); sin(phi)];
p(2,:)= [a(2,:)+x(2) NaN];
p(1,:)= [a(1,:)+x(1) NaN];

% % Use the following codes for a naive and visually better ellipse
% cdf=0.95;
% k=sqrt( -2*log(1-cdf) );
% px=P(1,1);py=P(2,2);pxy=P(1,2);
% if px==py,theta=pi/4; else theta=1/2*atan(2*pxy/(px-py));end
% r1=px*cos(theta)^2 + py*sin(theta)^2 + pxy*sin(2*theta);
% r2=px*sin(theta)^2 + py*cos(theta)^2 - pxy*sin(2*theta);
% T=[cos(theta) -sin(theta); sin(theta) cos(theta) ];
% pts=k*T*[sqrt(r1) 0; 0 sqrt(r2)]*[cos(phi); sin(phi)];
```

```
% p(1,:)=[pts(1,:)+x(1) NaN];
% p(2,:)=[pts(2,:)+x(2) NaN];

%
%

function p= make_covariance_ellipses_xb(x,P)
% compute ellipses for plotting state part B covariances
N= 10;
inc= 2*pi/N;
phi= 0:inc:2*pi;

lenx= length(x);
lenf= lenx/2;
p= zeros (2,(lenf)*(N+2));

ctr= 1;
for i=1:lenf
   ii= ctr:(ctr+N+1);
   jj= 2*i-1; jj= jj:jj+1;

   p(:,ii)= make_ellipse(x(jj), P(jj,jj), 2, phi);
   ctr= ctr+N+2;
end


%
%

function [circle1, circle2] = make_range_circles(r1, r2)
%
global g_current_ala_center
phi = 0:2*pi/50:2*pi;
aa = [cos(phi); sin(phi)];
circle1(1,:) = [r1*aa(1,:) + g_current_ala_center(1) NaN];
circle1(2,:) = [r1*aa(2,:) + g_current_ala_center(2) NaN];
circle2(1,:) = [r2*aa(1,:) + g_current_ala_center(1) NaN];
circle2(2,:) = [r2*aa(2,:) + g_current_ala_center(2) NaN];

%
%

function initialise_store()
% offline storage initialisation
global GDATA XA PA vtrue
GDATA.i=1;
GDATA.path= XA;
GDATA.true= vtrue;
GDATA.state(1).x= XA;
GDATA.state(1).P= diag(PA);

%
%

function store_data()
% add current data to offline storage
global GDATA XA XB PA PB vtrue
CHUNK= 5000;
if GDATA.i == size(GDATA.path,2) % grow array in chunks to amortise reallocation
   GDATA.path= [GDATA.path zeros(3,CHUNK)];
   GDATA.true= [GDATA.true zeros(3,CHUNK)];
end
i= GDATA.i + 1;
GDATA.i= i;
GDATA.path(:,i)= XA(1:3);
GDATA.true(:,i)= vtrue;
if size(XB,1) > 1
   GDATA.state(i).x= [XA; XB];
   GDATA.state(i).P= [diag(PA); diag(PB)];
else
```

```
   GDATA.state(i).x = XA;
   GDATA.state(i).P= diag(PA);
end


%
%

function finalise_data()
% offline storage finalisation
global GDATA
GDATA.path= GDATA.path(:,1:GDATA.i);
GDATA.true= GDATA.true(:,1:GDATA.i);

function check_data_association(list)
%
a= sort(list);
if length(a) > 1
   for i = 1:1:length(a)-1,
      if (a(i+1)-a(i) < 0 5)
         list
         error('data association error!')
      end
   end
end
```

Due to the large size of CEKF-SLAM, all parts cannot be reproduced in the Appendix. The main file, cekfslam.m is reproduced without supporting files; add_control_noise, add_observation_noise, augment, compute_rotationspeed, data_associate, frontend, full_states_update, get_observations, line_plot_conversions, loop902, observe_model, pi_to_pi, plot_feature_loci, predict, reassign_states, swtich_active_local_area, TransformToGlobal, update & vehicle_model.

Please see the following location to view the full CEKF-SLAM code:

https://openslam-org.github.io/cekfslam.html

# Appendix F: Linking Code for MATLAB to AirSim

```
%% 初期化
%初期化
clc
clear

%パス追加 windows起動ごとにリセットされるっぽいのでここでチェックして、必要に応じて加える
if count(py.sys.path,'C:\Users\) == 0
    insert(py.sys.path,int32(0),'C:\Users\);
end

%画像配列の確保
img h=288;
img_w=512;
img_px=img_h*img_w;
resimg=zeros(img_w,img_h,3,'uint8'); %y*xのサイズになるらしい
resimg2=zeros(img_h,img_w,3,'uint8'); %転置先の配列

%figureの確保
f_vel=figure;
hold on;
f xy=figure;
hold on;
f cam=figure;

%キー入力受付用figure
%hf=figure('position',[0 0 eps eps],'menubar','none');


%% 接続
%connect to the AirSim simulator
client=py.AirSimClient.CarClient; %クライアントの宣言
client.confirmConnection();
client.enableApiControl(true); %接続とAPIの有効化
car controls = py.AirSimClient.CarControls; %コントローラーの宣言


%% 制御ループ
client.simPause(logical(true))
while(1)
%状態更新
client.simContinueForTime(0.01) %10ms間隔で更新

%車両状態の取得と表示
car_state = client.getCarState();
state t=car state.timestamp;
state_vel = car_state.speed;
state xp = car state.kinematics true.position.x val;
state yp = car state.kinematics true.position.y val;
state_zp = car_state.kinematics_true.position.z_val;


%カラー画像の取得
response = cell(client.simGetImages(py.list({    py.AirSimClient.ImageRequest(int8(0), int8(0),
logical(false), logical(false))    })));
imgdata=uint8(response{1,1}.image data uint8);
u=1;
for t = 1:img_h*img_w
    resimg(t)=          imgdata(u);
    resimg(t+img_px)=   imgdata(u+1);
    resimg(t+img_px*2)= imgdata(u+2);
    u=u+4;
end
resimg2=permute(resimg,[2 1 3]);
```

```
%車両状態の指定
car_controls.throttle = 0.30;
car controls.steering = 1.0;

%入力の適用
client.setCarControls(car controls);

%グラフ表示
figure(f xy);
plot (state_xp,-state_yp,'o-');
daspect([1 1 1]);
figure(f vel);
plot (state_t,state_vel,'o-');
figure(f cam);
image(resimg2)
daspect([1 1 1]);

%pause(0.0001) %py.time.sleep(20)


end

client.simPause(logical(false))


%% 終了
client.reset()
client.enableApiControl(false)
```

Full acknowledgements must be given to user Hashikemu who is the author of this linking code as can be found at:

https://github.com/microsoft/AirSim/issues/1035

# Appendix G: Pretrained Occupancy Grid for Monocular Input

```
% Download the pretrained network.
pretrainedURL = 'https://www.mathworks.com/supportfiles/vision/data/segnetVGG16CamVid.mat';
pretrainedFolder = fullfile(tempdir,'pretrainedSegNet');
pretrainedSegNet = fullfile(pretrainedFolder,'segnetVGG16CamVid.mat');
if ~exist(pretrainedFolder,'dir')
    mkdir(pretrainedFolder);
    disp('Downloading pretrained SegNet (107 MB)...');
    websave(pretrainedSegNet,pretrainedURL);
    disp('Download complete.');
end

% Load the network.
data = load(pretrainedSegNet);
net = data.net;

% Read the image.
I = imread('seq05vd_snap_shot.jpg');

% Segment the image.
[C,scores,allScores] = semanticseg(I,net);

% Overlay free space onto the image.
B = labeloverlay(I,C,'IncludedLabels',"Road");

% Display free space and image.
figure
imshow(B)

% Use the network's output score for Road as the free space confidence.
roadClassIdx = 4;
freeSpaceConfidence = allScores(:,:,roadClassIdx);

% Display the free space confidence.
figure
imagesc(freeSpaceConfidence)
title('Free Space Confidence Scores')
colorbar

% Create monoCamera for CamVid data.
sensor = camvidMonoCameraSensor();

% Define bird's-eye-view transformation parameters.
distAheadOfSensor = 20; % in meters, as previously specified in monoCamera height input
spaceToOneSide    = 3;  % look 3 meters to the right and left
bottomOffset      = 0;
outView = [bottomOffset, distAheadOfSensor, -spaceToOneSide, spaceToOneSide];

outImageSize = [NaN, 256]; % output image width in pixels; height is chosen automatically to preserve
units per pixel ratio

birdsEyeConfig = birdsEyeView(sensor,outView,outImageSize);

% Resize image and free space estimate to size of CamVid sensor.
imageSize = sensor.Intrinsics.ImageSize;
I = imresize(I,imageSize);
freeSpaceConfidence = imresize(freeSpaceConfidence,imageSize);

% Transform image and free space confidence scores into bird's-eye view.
imageBEV = transformImage(birdsEyeConfig,I);
freeSpaceBEV = transformImage(birdsEyeConfig,freeSpaceConfidence);

% Display image frame in bird's-eye view.
figure
imshow(imageBEV)

% Define dimensions and resolution of the occupancy grid.
```

```
gridX = distAheadOfSensor;
gridY = 2 * spaceToOneSide;
cellSize = 0.25; % in meters to match units used by CamVid sensor

% Create the occupancy grid from the free space estimate.
occupancyGrid = createOccupancyGridFromFreeSpaceEstimate(...
    freeSpaceBEV, birdsEyeConfig, gridX, gridY, cellSize);

% Create bird's-eye plot.
bep = birdsEyePlot('XLimits',[0 distAheadOfSensor],'YLimits', [-5 5]);

% Add occupancy grid to bird's-eye plot.
hold on
[numCellsY,numCellsX] = size(occupancyGrid);
X = linspace(0, gridX, numCellsX);
Y = linspace(-gridY/2, gridY/2, numCellsY);
h = pcolor(X,Y,occupancyGrid);
title('Occupancy Grid (probability)')
colorbar
delete(legend)

% Make the occupancy grid visualization transparent and remove grid lines.
h.FaceAlpha = 0.5;
h.LineStyle = 'none';

% Add coverage area to plot.
caPlotter = coverageAreaPlotter(bep, 'DisplayName', 'Coverage Area');

% Update it with a field of view of 35 degrees and a range of 60 meters
mountPosition = [0 0];
range = 15;
orientation = 0;
fieldOfView = 35;
plotCoverageArea(caPlotter, mountPosition, range, orientation, fieldOfView);
hold off

% Create the costmap.
costmap = vehicleCostmap(flipud(occupancyGrid), ...
    'CellSize',cellSize, ...
    'MapLocation',[0,-spaceToOneSide]);
costmap.CollisionChecker.InflationRadius = 0;

% Display the costmap.
figure
plot(costmap,'Inflation','off')
colormap(parula)
colorbar
title('Vehicle Costmap')

% Orient the costmap so that it lines up with the vehicle coordinate
% system, where the X-axis points in front of the ego vehicle and the
% Y-axis points to the left.
view(gca,-90,90)

% Create a set of locations in vehicle coordinates.
candidateLocations = [
    8 0.375
    10 0.375
    12 2
    14 0.375
    ];

% Check if locations are occupied.
isOccupied = checkOccupied(costmap,candidateLocations);

% Partition locations into free and occupied for visualization purposes.
occupiedLocations = candidateLocations(isOccupied,:);
freeLocations = candidateLocations(~isOccupied,:);
```

```
% Display free and occupied points on top of costmap.
hold on
markerSize = 100;
scatter(freeLocations(:,1),freeLocations(:,2),markerSize,'g','filled')
scatter(occupiedLocations(:,1),occupiedLocations(:,2),markerSize,'r','filled');
legend(["Free" "Occupied"])
hold off

function sensor = camvidMonoCameraSensor()
% Return a monoCamera camera configuration based on data from the CamVid
% data set[1].
%
% The cameraCalibrator app was used to calibrate the camera using the
% calibration images provided in CamVid:
%
% http://web4.cs.ucl.ac.uk/staff/g.brostow/MotionSegRecData/data/CalibrationSeq_and_Files_0010YU.zip
%
% Calibration pattern grid size is 28 mm.
%
% Camera pitch is computed from camera pose matrices [R t] stored here:
%
% http://web4.cs.ucl.ac.uk/staff/g.brostow/MotionSegRecData/data/EgoBoost_trax_matFiles.zip

% References
% ----------
% [1] Brostow, Gabriel J., Julien Fauqueur, and Roberto Cipolla. "Semantic Object
% Classes in Video: A high-definition ground truth database." _Pattern Recognition
% Letters_. Vol. 30, Issue 2, 2009, pp. 88-97.

calibrationData = load('camera_params_camvid.mat');

% Describe camera configuration.
focalLength    = calibrationData.cameraParams.FocalLength;
principalPoint = calibrationData.cameraParams.PrincipalPoint;
imageSize      = calibrationData.cameraParams.ImageSize;

% Camera height estimated based on camera setup pictured in [1].
height = 0.5;  % height in meters from the ground

% Camera pitch was computed using camera extrinsics provided in data set.
pitch = 0;  % pitch of the camera, towards the ground, in degrees

camIntrinsics = cameraIntrinsics(focalLength,principalPoint,imageSize);
sensor = monoCamera(camIntrinsics,height,'Pitch',pitch);
end
function occupancyGrid = createOccupancyGridFromFreeSpaceEstimate(...
    freeSpaceBEV,birdsEyeConfig,gridX,gridY,cellSize)
% Return an occupancy grid that contains the occupancy probability over
% a uniform 2-D grid.

% Number of cells in occupancy grid.
numCellsX = ceil(gridX / cellSize);
numCellsY = ceil(gridY / cellSize);

% Generate a set of (X,Y) points for each grid cell. These points are in
% the vehicle's coordinate system. Start by defining the edges of each grid
% cell.

% Define the edges of each grid cell in vehicle coordinates.
XEdges = linspace(0,gridX,numCellsX);
YEdges = linspace(-gridY/2,gridY/2,numCellsY);

% Next, specify the number of sample points to generate along each
% dimension within a grid cell. Use these to compute the step size in the
% X and Y direction. The step size will be used to shift the edge values of
% each grid to produce points that cover the entire area of a grid cell at
% the desired resolution.

% Sample 20 points from each grid cell. Sampling more points may produce
```

```
% smoother estimates at the cost of additional computation.
numSamplePoints = 20;

% Step size needed to sample number of desired points.
XStep = (XEdges(2)-XEdges(1)) / (numSamplePoints-1);
YStep = (YEdges(2)-YEdges(1)) / (numSamplePoints-1);

% Finally, slide the set of points across both dimensions of the grid
% cells. Sample the occupancy probability along the way using
% griddedInterpolant.

% Create griddedInterpolant for sampling occupancy probability. Use 1
% minus the free space confidence to represent the probability of occupancy.
occupancyProb = 1 - freeSpaceBEV;
sz = size(occupancyProb);
[y,x] = ndgrid(1:sz(1),1:sz(2));
F = griddedInterpolant(y,x,occupancyProb);

% Initialize the occupancy grid to zero.
occupancyGrid = zeros(numCellsY*numCellsX,1);

% Slide the set of points XEdges and YEdges across both dimensions of the
% grid cell.
for j = 1:numSamplePoints

    % Increment sample points in the X-direction
    X = XEdges + (j-1)*XStep;

    for i = 1:numSamplePoints

        % Increment sample points in the Y-direction
        Y = YEdges + (i-1)*YStep;

        % Generate a grid of sample points in bird's-eye-view vehicle coordinates
        [XGrid,YGrid] = meshgrid(X,Y);

        % Transform grid of sample points to image coordinates
        xy = vehicleToImage(birdsEyeConfig,[XGrid(:) YGrid(:)]);

        % Clip sample points to lie within image boundaries
        xy = max(xy,1);
        xq = min(xy(:,1),sz(2));
        yq = min(xy(:,2),sz(1));

        % Sample occupancy probabilities using griddedInterpolant and keep
        % a running sum.
        occupancyGrid = occupancyGrid + F(yq,xq);
    end

end

% Determine mean occupancy probability.
occupancyGrid = occupancyGrid / numSamplePoints^2;
occupancyGrid = reshape(occupancyGrid,numCellsY,numCellsX);
end
```

Full credit must be given to MathWorks for the example code used for the project. The code can be accessed at:

https://au.mathworks.com/help/driving/ug/create-occupancy-grid-using-monocular-camera-sensor.html

# Appendix H: Failed CNN Code for Training

```
# Std packages
import numpy as np
import pandas as pd
import time
import math

import matplotlib.pyplot as plt

# Torch and supporting.
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.nn import Conv2d
from torch.nn import MaxPool2d
from torch.nn import Linear
from torch.nn import ReLU
from torch.nn import Softmax
from torch.nn import Module
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader

# Torchvision
import torchvision
from torchvision import transforms
from torchvision.transforms import ToTensor

epochs = 10
lr = 0.001
batch_size = 64

"""
Define the model - CNN with six output classes being one of the six axis of freedom
"""
class netCNN(nn.Module):
    def __init__(self, lr, epochs, batch_size, num_classes=6):
        super(netCNN, self).__init__()
        # This section is used to initialise parameters for the CNN, allocation to GPU, etc.
        self.epochs = epochs
        self.lr = lr
        self.batch_size = batch_size
        self.num_classes = num_classes
        self.loss_history = []
        self.acc_history = []
        self.device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

    # Convolutional layers - 6 layers with 2 pooling layers at 3 and 6
        # First Convolution Layer with normalisation
        self.conv1 = nn.Conv2d(3, 32, 3)
        self.bn1 = nn.BatchNorm2d(32)
        # Second Convolution Layer with normalisation
        self.conv2 = nn.Conv2d(32,32,3)
        self.bn2 = nn.BatchNorm2d(32)
        # Third Convolution Layer with normalisation
        self.conv3 = nn.Conv2d(32,32,3)
        self.bn3 = nn.BatchNorm2d(32)
        # First Pooling Layer
        self.maxpool1 = nn.MaxPool2d(2)
        # Fourth Convoluton Layer with normalisation
        self.conv4 = nn.Conv2d(32,64,3)
        self.bn4 = nn.BatchNorm2d(64)
        # Fifth Convolution Layer with normalisation
        self.conv5 = nn.Conv2d(64,128,3)
        self.bn5 = nn.BatchNorm2d(128)
        # Sixth Convolution Layer with normalisation
        self.conv6 = nn.Conv2d(128,128,3)
        self.bn6 = nn.BatchNorm2d(128)
        # Second Pooling Layer
```

```
    self.maxpool2 = nn.MaxPool2d(2)

    # Calculate dimension of final pooling layer for input to FC layer
    input_dims = self.calc_input_dims()

# Fully Connected Layers
    # First fully connected layer
    self.fc1 = nn.Linear(input_dims, input_dims / 8)
    # Second fully connected layer
    self.fc2 = nn.Linear(input_dims / 8, input_dims / 32)
    # Third & Final fully connected layer
    self.fc3 = nn.Linear(input_dims / 32, self.num_classes)

# Define the method of optimisation - Adam generally considered to be the best for large networks as it is faster than SGD
    self.optimizer = optim.Adam(self.parameters(), lr = self.lr)

# Define the loss method between classes - CrossEntropyLoss better for more than 2 classes
    self.loss = nn.CrossEntropyLoss()
    self.to(self.device)
    self.get_data()

# Calculate the network size
def calc_input_dims(self):
    batch_data = torch.zeros((1, 3, 480, 480))

    batch_data = self.conv1(batch_data)
    batch_data = self.bn1(batch_data)

    batch_data = self.conv2(batch_data)
    batch_data = self.bn2(batch_data)

    batch_data = self.conv3(batch_data)
    batch_data = self.bn3(batch_data)

    batch_data = self.maxpool1(batch_data)

    batch_data = self.conv4(batch_data)
    batch_data = self.bn4(batch_data)

    batch_data = self.conv5(batch_data)
    batch_data = self.bn5(batch_data)

    batch_data = self.conv6(batch_data)
    batch_data = self.bn6(batch_data)

    batch_data = self.maxpool2(batch_data)

    return int(np.prod(batch_data.size()))

# This is the forward pass for the network
def forward(self, batch_data):
    batch_data = torch.tensor(batch_data).to(self.device)

    batch_data = self.conv1(batch_data)
    batch_data = self.bn1(batch_data)
    batch_data = F.relu(batch_data)

    batch_data = self.conv2(batch_data)
    batch_data = self.bn2(batch_data)
    batch_data = F.relu(batch_data)

    batch_data = self.conv3(batch_data)
    batch_data = self.maxpool1(batch_data)
    batch_data = F.relu(batch_data)

    batch_data = self.conv4(batch_data)
    batch_data = self.bn4(batch_data)
    batch_data = F.relu(batch_data)

    batch_data = self.conv5(batch_data)
```

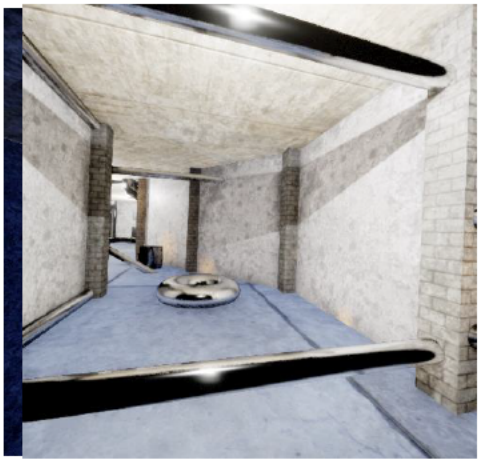
```
        batch_data = self.bn5(batch_data)
        batch_data = F.relu(batch_data)

        batch_data = self.conv6(batch_data)
        batch_data = self.bn6(batch_data)
        batch_data = F.relu(batch_data)

        batch_data = self.maxpool2(batch_data)

        batch_data = batch_data.view(batch_data.size()[0], -1)

        classes = self.fc1(batch_data)

        return classes

    # routine to bring in data --> needs to be changed to bring in data from a local source
    def get_data(self):
        train_data = FLIGHTS('name', train=True, download=True, transfrom=ToTensor())
        self.train_data_loader = torch.utils.data.DataLoader(train_data, batch_size=self.batch_size, shuffle=True, num_workers=16)

        test_data = FLIGHTS('name', train=False, download=True, transfrom=ToTensor())
        self.test_data_loader = torch.utils.data.DataLoader(test_data, batch_size=self.batch_size, shuffle=True, num_workers=16)

    # training routine
    def _train(self):
        self.train()
        for i in range(self.epochs):
            ep_loss = 0
            ep_acc = []
            for j, (input, label) in enumerate(self.train_data_loader):
                self.optimizer.zero_grad()
                label = label.to(self.device)
                prediction = self.forward(input)
                loss = self.loss(prediction, label)
                prediction = F.softmax(prediction, dim=1)
                classes = torch.argmax(prediction, dim=1)
                no_wrong = torch.where(classes != label, torch.tensor([1.].to(self.device), torch.tensor([0.]).to(self.device)))
                acc = 1 - torch.sum(no_wrong) / self.batch_size

                ep_acc.append(acc.item())
                self.acc_history.append(acc.item())
                ep_loss += loss item()
                loss.backward()
                self.optimzer.step()

            print('Finished epoch', i, 'total loss %.3f' % ep_loss, 'accuracy %.3f' % np.mean(ep_acc))

    # testing routine
    def _test(self):
        self.test()
        for j, (input, label) in enumerate(self.test_data_loader):
            self.optimizer.zero_grad()
            label = label.to(self.device)
            prediction = self.forward(input)
            loss = self.loss(prediction, label)
            prediction = F.softmax(prediction, dim=1)
            classes = torch.argmax(prediction, dim=1)
            no_wrong = torch.where(classes != label, torch.tensor([1.].to(self.device), torch.tensor([0.]).to(self.device)))
            acc = 1 - torch.sum(no_wrong) / self.batch_size

            ep_acc.append(acc.item())
            ep_loss += loss.item()


            print('Finished epoch', i, 'total loss %.3f' % ep_loss, 'accuracy %.3f' % np.mean(ep_acc))

# Runs the defined network on the selected dataset -> Initialise, train, test
if __name__ == "__main__":
    network = netCNN(lr=0.001, batch_size=128, epochs=25)
    network._train()
```

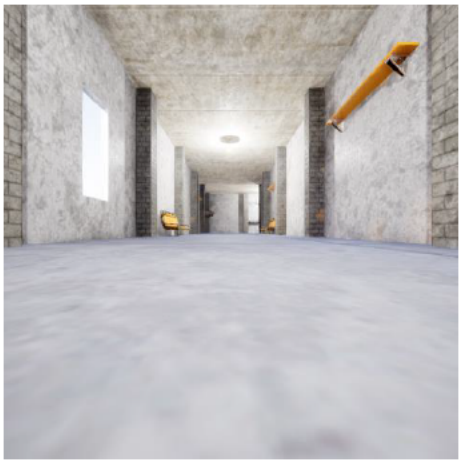
```
plt.plot(network.loss_history)
plt.show()
plt.plot(network.acc_history)
plt.show()
network._test()
```

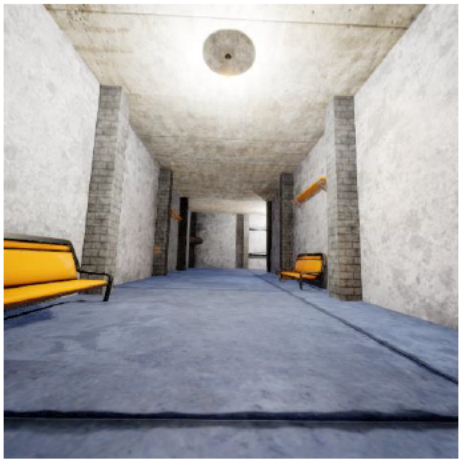
Full acknowledgement must be given to Sentdex & PythonProgramming.net as the code template provided by their tutorial served as the base code for this version of the CNN. The code can be found in the link found below:

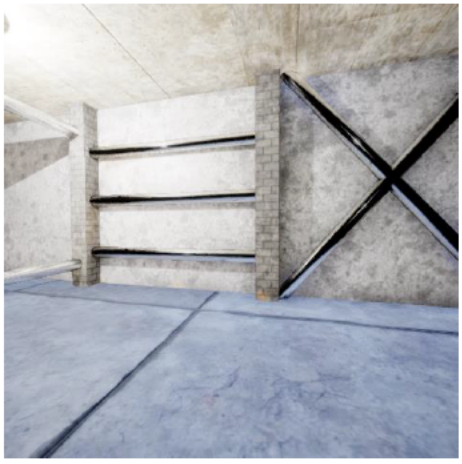
https://pythonprogramming.net/building-deep-learning-neural-network-pytorch/

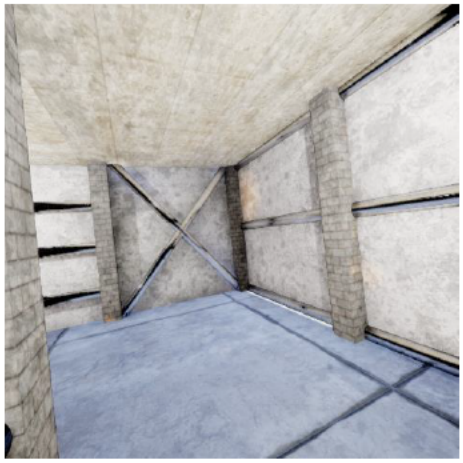
## Appendix I: Sample of Training Data

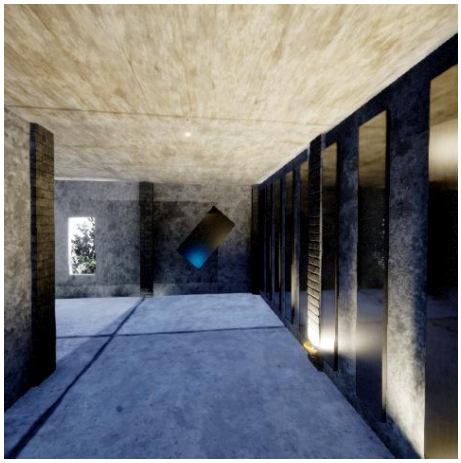


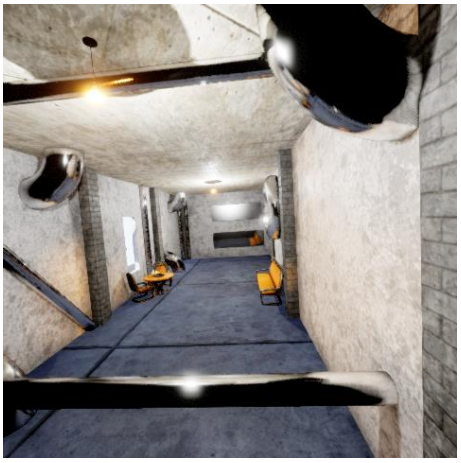


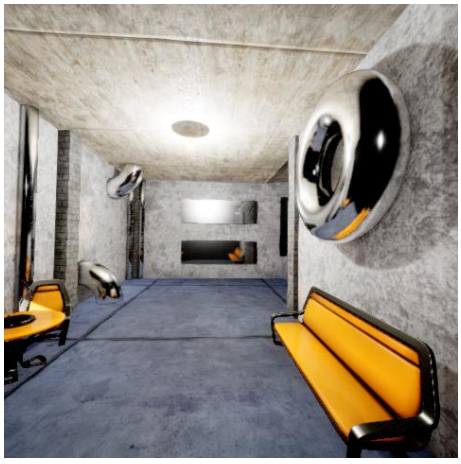


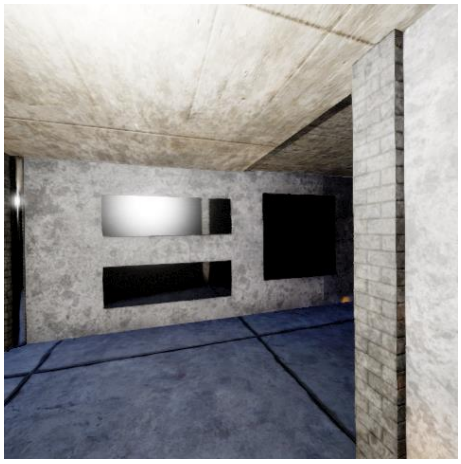


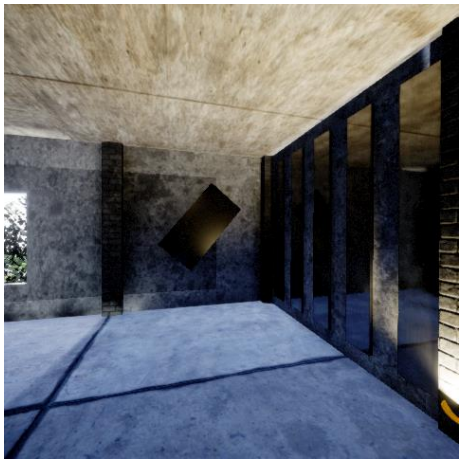


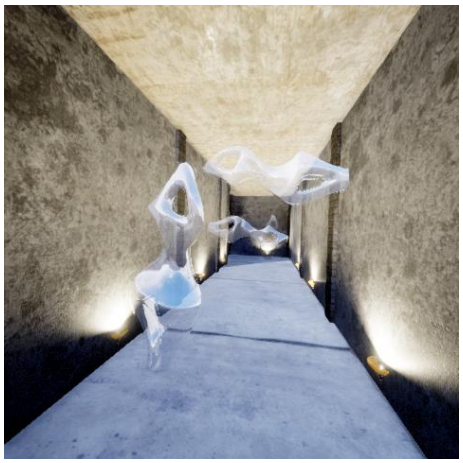


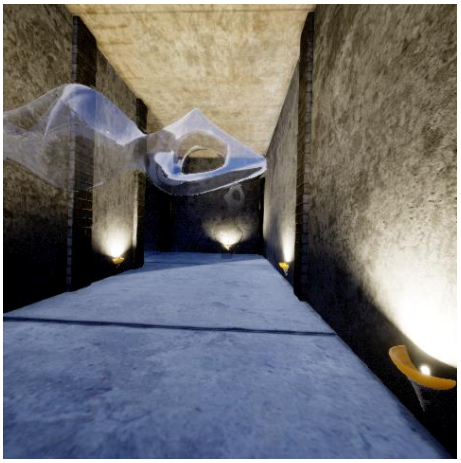


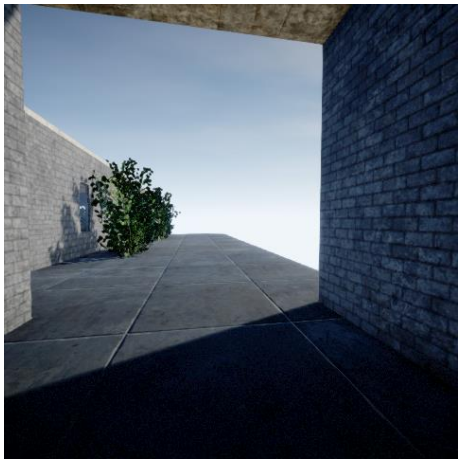


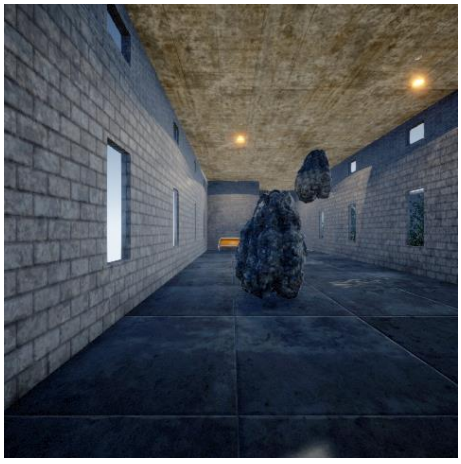


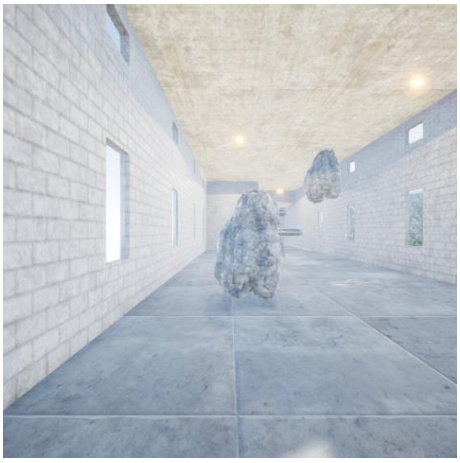












## Appendix J

# Code of Ethics

As engineering practitioners, we use our knowledge and skills for the benefit of the community to create engineering solutions for a sustainable future. In doing so, we strive to serve the community ahead of other personal or sectional interests.

Our Code of Ethics defines the values and principles that shape the decisions we make in engineering practice. The related Guidelines on Professional Conduct provide a framework for members of Engineers Australia to use when exercising their judgment in the practice of engineering and as members of Engineers Australia more broadly.

As members of Engineers Australia, we commit to practise in accordance with the Engineers Australia's General Regulations regarding competency, continuing professional development and the Code of Ethics. We accept that we will be held accountable for our conduct under Engineers Australia's disciplinary regulations.

**In the course of engineering practice we will:**

1

### Demonstrate integrity

1.1 Act on the basis of a well-informed conscience

1.2 Be honest and trustworthy

1.3 Respect the dignity of all persons.

2

### Practise competently

2.1 Maintain and develop knowledge and skills

2.2 Represent areas of competence objectively

2.3 Act on the basis of adequate knowledge.

3

### Exercise leadership

3.1 Uphold the reputation and trustworthiness of the practice of engineering

3.2 Support and encourage diversity

3.3 Make reasonable efforts to communicate honestly and effectively to all stakeholders, taking into account the reliance of others on engineering expertise.

4

### Promote sustainability

4.1 Engage responsibly with the community and other stakeholders

4.2 Practise engineering to foster the health, safety and wellbeing of the community and the environment

4.3 Balance the needs of the present with the needs of future generations.