2022-12-01

Intelligent Autonomous Inspections Using Deep Learning And Detection Markers

Alejandro Martinez Acosta
University of Texas at El Paso

INTELLIGENT AUTONOMOUS INSPECTIONS USING DEEP LEARNING AND DETECTION MARKERS

ALEJANDRO MARTINEZ ACOSTA

Doctoral Program in Mechanical Engineering

APPROVED:

Angel Flores-Abad, Chair, Ph.D.

Ahsan R. Choudury, Co-Chair, Ph.D.

Joel Quintana, Ph.D

David Murakami, Ph.D.

Stephen Crites, Ph.D.
Dean of the Graduate School

To my

MOTHER, FATHER and WIFE

with all my love

INTELLIGENT AUTONOMOUS INSPECTIONS USING DEEP LEARNING AND DETECTION MARKERS

by

ALEJANDRO MARTINEZ ACOSTA

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fullfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

Department of Aerospace and Mechanical Engineering

THE UNIVERSITY OF TEXAS AT EL PASO

December 2022

Acknowledgements

I would like to express my gratitude to my PhD. advisor Dr. Flores-Abad, who gave me the opportunity to work under his supervision on this dissertation project. I would like to thank him for the knowledge and guidance throughout my doctoral career.

Additionally, I would like to thank David Murakami PhD. for his guidance and feedback on this project as well as the opportunity to complete three internships at NASA AMES. Dr. Murakami was a crucial part of this project as he helped me focus on specific portions of this project. Our weekly meetings helped keep track of my progress and prioritize tasks.

I would also like to thank the Center for Space Exploration Technology Research (cSTER), for the further developing my engineering skills. Many important engineering and life lessons were taught to me by brilliant staff and faculty members of the glscster. Many thanks to all the people that gave me the chance to prove myself and gain confidence at the cSTER labs.

I want to also thank my brother Rodrigo Martinez and my sister Gabriela Martinez, that through their exemplar achievements and careers, have taught me to always pursue for greatness, devotion and passion for your career. I am deeply in debt for their support and love throughout our years growing up, an above all, for pushing me to always become a better person at all stages in my life.

I also owe this achievement to both of my parents Miguel Martinez and Yolanda Acosta. Without their unconditional love and support, this would not have been possible. To my father, I would like to express my eternal gratitude for the many life lessons that you taught me. For raising me to become the man I am right now. To my mother, thank you for your unconditional love, devotion, dedication and sacrifice. For all of the sacrifices you made to ensure that I will have a suitable and everlasting education, I cannot thank you enough.

Lastly, I would like to thank my wife and love of my life, Mireya Jimenez. Thank you for sticking by my side no matter how difficult this journey was. Your presence and love made all of this much more endurable and sustainable. Thank you for pushing me to always overcome difficulties with grace and gratitude. For your unconditional love and support on doing what I love. This is also for you!

Abstract

Inspection of industrial and scientific facilities is a crucial task that must be performed regularly. These inspections tasks ensure that the facility's structure is in safe operational conditions for humans. Furthermore, the safe operation of industrial machinery, is dependent on the conditions of the environment. For safety reasons, inspections for both structural integrity and equipment is often manually performed by operators or technicians. Naturally, this is often a tedious and laborious task. Additionally, buildings and structures frequently contain hard to reach or dangerous areas, which leads to the harm, injury or death of humans.

Autonomous robotic systems offer an attractive solution to the automation of inspections. This is due to their ability to reach remote and dangerous areas. Consequently, significant research efforts have been made for the use Unmanned Aerial Vehicle (UAV) and drones as flight inspection units. Compared to Unmanned Ground Vehicle (UGV), drones offer significantly better acrobatic and agility performance.

Needless to say, an autonomous drone must posses certain abilities to carry out inspections with little or no human intervention. Foremost, the drone must be able to position and orient itself towards inspection waypoints. Moreover, the drone must be able to use a map of the environment to identify and navigate to areas of interest on an inspection mission.

These abilities alone do not provide the drone the autonomy and efficiency required to perform quality inspections within required time frames. The inspection problem is often framed as a way to maximize the amount of volume a drone can cover through the use of coverage path planning. Therefore, drone systems spend most of the time inspecting areas or objects that produce little to no value for identifying defects. In addition, more time and drone flights must be performed to cover an entire area. Thus, this thesis proposes a novel approach to perform intelligent inspections using deep learning and Quick Response (QR) codes to identify, prioritize and segment objects of interest in the context of an inspection. Additionally, a novel navigation architecture that can find and prioritize collision-free trajectories is proposed. This architecture proposes the use of hierarchical state machines to capture complex reactive behavior found on a task driven robot. In conjunction, the proposed systems solve the inspection problem by identifying, prioritizing, labelling and navigating towards areas or objects of interest.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Mobile robots have been used for the inspection of industrial and scientific environments. Robots can potentially improve inspection tasks by making them safer and faster on tasks such as welding, painting and packaging [1]. Additionally, a robot can utilize a multitude of sensors such as cameras, range detectors, radar and depth sensors that yield a greater set of data, compared to just visual human inspection. The data gathered from these sensors provides accurate measurements for a multitude of environment variables that can assist in the process of automating human related tasks. Therefore, aerial mobile robots or drones, have recently gained popularity in the field of automated inspections due to their low-cost, speed, agility and inspection data output. In contrast, human inspection requires qualified personnel to visually inspect structural and/or equipment, which is time consuming and in some cases, even dangerous. Thus, the objective of an automated inspection task, is to detect structural or equipment damage or defects[1].

However, drone robots still lack the autonomy to determine whether there are defects or not while performing a flight mission. Current approaches heavily rely in the use of Ground Control Station (GCS), to plan, execute and monitor drone flight missions. Other approaches simply require an operator to conduct the mission manually through the use of a remote controller. Consequently, drone inspections are manual, slow and energy inefficient, as several fly missions are required to fully inspect an area.

This research aims to add a layer of automation and intelligence using deep learning neural networks and Quick Response (QR) codes as navigation markers. In addition, a novel drone navigation architecture with priority oriented navigation goals is presented in this thesis. This navigation architecture has the autonomy to choose between a low priority inspection plan, and high priority object or QR detection marker by interrupting and re-planning the original navigation trajectory, with a new higher priority trajectory.

This chapter will introduce the research problem, followed by aims, objectives and research significance. Additionally, several background research sections are presented at the beginning of this chapter. These sections provide the background study and concepts used to develop the ideas presented in this thesis. Finally, a section is dedicated to outline the chapters included in this thesis document.

[1]A damaged or defect is defined as visual signs of wear, tear or environment exposure that could potentially compromise the functionality and safety of the inspected item

1.1 Research Problem

Performing inspections using drones is certainly not a new research topic. Drone systems have been proposed for the inspection of public and private infrastructure such as bridges, power plants, sewers and wind turbines [2]. Inspection drones have also been used in areas such as cartography, agriculture, volcanology and photogrammetry [3]. Nonetheless, there is still a great interest on increasing the autonomy of drones, specifically on running Artifical Intelligence (AI) algorithms directly on the drone flight control computer. Increasing the level of autonomy on a drone would reduce human intervention, while at the same time increasing the decision making process of the drone. Moreover, increasing the level of autonomy would result in energy efficient inspections, as the number of flights will be reduced by limiting the inspection problem to observing and prioritizing detected areas, objects or markers.

Several efforts have been made to include deep learning to drones, with the aim to perform image recognition and segmentation [4, 5]. Traffic [6], agriculture [7] and natural disaster [8] surveillance, landing marker detection [9] and object marker detection [10] are some of the applications that make use of deep learning on drones.

Nevertheless, these applications are fairly limited on what they can accomplish. As an example, deep learning for object recognition or segmentation are only used as defect markers for post-processing tasks, as it is the case for agriculture surveillance. Moreover, these defect markers do not affect the flight mission of the drone, as they are simply ignored for navigation purposes. This is troublesome for applications such as natural disaster surveillance or automated inspections, where further actions need to be taken after a detection has been confirmed. For instance, in the application of natural disaster surveillance, assistance should be provided to human subjects in distress or in a dangerous situations by performing a set of actions or relaying information about the status of the situation.

Ultimately, this leads to inaction and a lack of autonomy, as the drone will not be capable of making smart and reactive decisions based on the output of its sensors and AI algorithms. This is a problem because some cases require extra actions to be taken. In the context of inspections, detecting defects on a building or equipment might require further actions such as navigating to the detected anomaly, recording a video, obtaining images, or capturing point-cloud data. This is especially important since drone flight time is limited, so capturing and detecting data becomes a crucial task.

Therefore, in this work, a concrete scenario is presented in section 7.6, where an initial inspection plan is generated on one of the walls of the National Full-Scale Aerodynamics Complex (NFAC) facility located at NASA AMES. While performing the initial inspection mission, a QR code marker is detected, which signals the drone to interrupt the initial inspection trajectory for a new higher priority inspection trajectory task. The drone then navigates to this new location where the QR code marker was detected. The drone performs actions such as recording images, video and point-cloud measurements at the marker's location. Lastly, the drone resumes its initial inspection mission and waits for another marker detection.

1.2 Research Aims and Objectives

Given the lack of research and development of drone autonomy in the context of inspections, this study aims to build and study an autonomous drone that can perform actions based on marker detections using deep learning and other image based codes. To complete this study, the following items must be accomplished:

1. Study and develop a drone navigation stack that can perform path planning using a map[2] (Chapter 4)
2. Identify a suitable deep learning neural network model and a QR system for automated detections (Chapter 5)
3. Construct and study the framework needed to bridge the gap between object recognition and drone navigation, which will add a new level of autonomy to the proposed inspection drone system. (Chapter 3)
4. Integrate actions that will improve and enhance the inspection quality (Chapter 6)

Ultimately, this thesis seeks to find an answer to the following specific question:

How can deep learning in conjunction with path planning be used to increase the level of autonomy of a drone for an inspection task?

1.3 Research Significance

This study will bring in new ideas on how to enhance the autonomy of a drone using path planning and deep learning neural networks by constructing a new framework that uses image recognition as markers for path planning. Additionally, this study also contributes to the field of autonomous inspection using drones by presenting a study case on a scientific environment.

This will address the lack of autonomy found on both commercial and research drones for inspection tasks. Furthermore, the presented work will fill the technical gap needed to achieve this autonomy by providing the software tools needed to complete this study. This will result in a reduction of human personnel needed to

[2]For a definition of a map in the context of this work see section 1.4.1

inspect structures, reduced facility downtime, an increase of periodic and reliable inspections, and a greater volume of data such as video, images and point-cloud that will assist in the early detection of damages.

1.4 Mobile Robot Navigation Architecture

An autonomous mobile robot must be able to navigate its environment on both static and dynamic objects[3]. Furthermore, it must obtain data from sensors and interpret it in a way that can be used to both localize and position the robot. Thus, a navigation architecture integrates components that can accomplish these tasks. For the majority of mobile robots, a navigation architecture requires two main components: Path planning and collision avoidance. Path planning tries to find a trajectory given a map. Its main objective is to change the robot from an initial state $p_{initial}$ to a goal state p_{goal}. In contrast, collision avoidance uses real-time sensor information to perform fast and dynamic avoidance of obstacles.

Another important aspect of navigation is the discretization of a continuous environment. This must always be the case before performing path planning, and it can have a great impact on the performance of the navigation system. For example, high resolution maps greatly increase the memory and storage requirements, which subsequently affects the computational complexity required for navigation, and the real-time response of the robotic system.

The following sections present the background study for mobile robot navigation architectures, with a focus on drone systems. On the first section, environment discretization with octrees and octomap is presented. This is followed by a brief introduction to path planning, along with the key research and techniques relevant to this thesis.

1.4.1 Octrees and Octomap

The first step in any navigation task is to transform a continuous environment into its discrete counterpart [11]. This can be accomplished using ranging sensor data and storing it into a spatial data structure. Some examples of spatial data structures are grid-maps, quad-trees, oc-trees and k-d-trees.

Although grid-maps are a popular choice in ground mobile robots, for drone applications this approach is impractical due to the memory size needed to store medium to large maps in $\mathbb{R}^3$. This fact is especially severe for embedded computers with constrained memory resources. For this reason, tree data structures are preferred over grid-maps, specifically for mobile robots that are not constrained in $\mathbb{R}^2$. Octrees can consol-

[3]This study defines static objects as those that cannot or are infrequently positioned into a different location and dynamic objects those that frequently change positions

idate large volumes of occupied or unoccupied space into a single voxel. Therefore, for drone applications, using octrees present an efficient way of creating and storing maps.

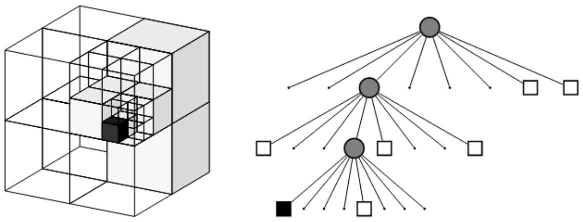


Figure 1.1: Example of a octree data structure and its tree representation [12]

An octree is a tree data structure that has exactly 8 node children. An octree is used along with sensor readings to discretize an environment in $\mathbb{R}^3$, by recursively partitioning space into 8 octants. Figure 1.1 shows an example of the tree structure used to partition space efficiently. Notice how some partitions are larger than others, this fact is the reason octrees are more efficient at discretizing environments for robotic applications in $\mathbb{R}^3$, since an octree can cluster portions of a map into bigger voxels.

Octomap is an efficient probabilistic 3D mapping software framework that uses an octree data structure to generate 3-dimensional maps [12]. The octomap framework is capable of representing occupied space through the following probabilistic model:

$$P(n|z_{1:t}) = [1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \frac{1 - P(n|z_{1:t-1}}{P(n|z_{1:t-1})} \frac{P(n)}{1 - P(n)}]^{-1} \tag{1.1}$$

Where, z_t is a current measurement and $P(n)$, $P(n|z_{1:t-1})$ is a prior and previous estimate respectively. Finally, $P(n|z_t)$ represents the probability of voxel n to be occupied given sensor measurement z_t.

Additionally, the octomap framework uses ray-casting from the sensor's field of view to occupied voxels to detect free or unoccupied space. This is a useful feature for path planning, since a configuration space with occupied and unoccupied partitions is required. Additionally, multiple resolutions from the same octomap can be queried. This is another useful feature, as there is no need to create multiple resolution scans of the same environment. Figure 1.2 show that the same octomap can be used at multiple resolutions.

1.4.2 Path Planning

Path planning seeks to find a collision-free trajectory from an initial location $p_{initial}$ to a target location p_{goal} within a reasonable amount of computational time, and with the shortest possible path. As previously

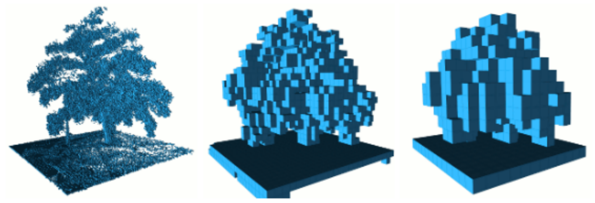


Figure 1.2: Octomap at different resolution levels [12]

stated, the first step for path planning is to discretize or obtain a map of a continuous environment. Using this map, a path planner can construct a graph considering cells on C_{free}, while simultaneously avoiding any cells on $C_{occupied}$. The constructed graph can be searched using path-finding algorithms such as depth-first, breadth-first search, dijkstra's or A* search algorithms to obtain a valid trajectory from $p_{initial}$ to p_{goal}.

Apart from graph searching, potential field methods for path planning are also found on the literature. In contrast to graph based methods, potential methods use gradient field vectors to find collision-free trajectories, much in the same way gravitational or electrical potential gradients are used to find body or electrical signals trajectories in space. However, these methods will not be covered in this thesis, so they are not part of the final study.

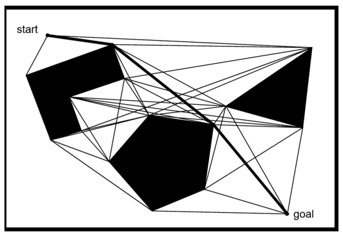
The next couple of sections introduce key ideas and strategies for graph construction and searching.

1.4.3 Graph Construction

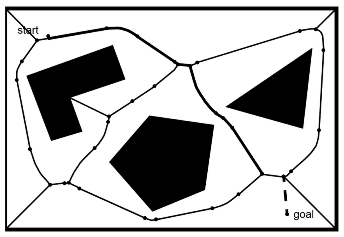
Graph construction produces a graph $G = \{V, E\}$, where V is the set of unoccupied partitions or vertices on C_{free} and E is the set of edges that connects all elements in V to its neighbors [13]. Clearly, the first step to construct a graph G, is to find all the elements $V \in C_{free}$. For completeness, every element $V \in C_{free}$ should be considered, however, in practice this is often computationally expensive or impractical. For this reason, there are two main graph construction strategies: structured graph construction and randomized graph construction.

A structured graph construction leverages the topology of the discretized environment to search for $V \in C_{free}$. For instance, on a binary occupancy grid-map, elements in V can be found by searching all elements in C_{free}, which are represented by a boolean value of 0. Some other examples of structured graph searches are visibility, cell decomposition, voronoi diagrams, and lattice graphs. Figure 1.3 shows the different strategies methods for graph construction.

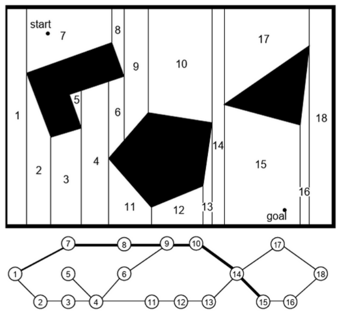
However, for higher dimensional spaces, finding all elements $V \in C_{free}$ can become impractical. Therefore, for this case, randomized graph searches on C_{free} are used. A randomized graph search randomly samples elements $V \in C_{free}$, so as to reduce the number of elements V on a graph. Rapidly Exploring



(a) visibility graph



(b) voronoi graph



(c) cell decomposition graph

Figure 1.3: Different graph construction strategies for path planning [11]

Random Trees (RRT) is one example of a randomized graph search strategy.

1.4.4 Graph Search

Once a graph is constructed, graph search algorithms can be used to find a trajectory between $p_{initial}$ and p_{goal}. A graph search algorithm attempts to find an optimal path, which for most robotic applications, refers to finding the shortest path to reach p_{goal}. Some examples of graph search algorithms are breadth-first search, depth-first search, dijkstra's and A* [14].

Furthermore, incremental graph searching and planning can be found on robotic applications. These search algorithms use previous search results to refine subsequent search queries. Moreover, incremental path planning constantly updates both the map and graph, which ultimately, results in faster searches. Popular incremental search algorithms are D* and Life Prolonged A* (LPA*).

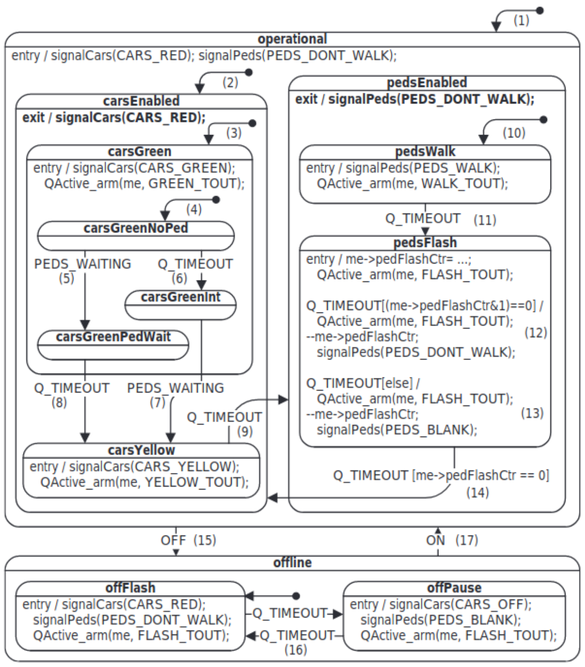
Although all search algorithms produce optimal results, in this thesis an A* search was selected. For the A* algorithm, see Appendix A.

1.5 Hierarchical State Machines

Reactive event-driven software systems are thoroughly used in robotic applications, as it concisely expresses the reaction nature of many of the robot's components. For instance, a robot must react to the different sensor stimuli through a set of pre-determined actions. Consequently, this behavior is often described through the use of Finite State Machine (FSM). However, expressing complex systems through the use of a FSM, causes a phenomenon known as state explosion [15], where many of the state machine's behaviors require an untractable amount of repeated states and transitions. Therefore, several extensions to traditional FSMs have been proposed by Harel [16], with the following requirements:

1. Cluster states together into hierarchical super states
2. Independence or orthogonality between states
3. General transitions
4. Refinement of states

These extensions allow traditional FSMs to cluster functionality into a higher-level states that can all react to common behavior. Hence, code abstraction can be used to hide complex behavior between states. Figure 1.4 shows an example how a pedestrian light control [15] leverages hierarchical state machines to design a complex reactive event-driven system. The following sections will briefly explain important concepts and background information related to hierarchical states, events, transitions, conditional guards and orthogonal states.



(1)
operational
entry / signalCars(CARS_RED); signalPeds(PEDS_DONT_WALK);
(2)
carsEnabled
exit / signalCars(CARS_RED);
(3)
carsGreen
entry / signalCars(CARS_GREEN);
QActive_arm(me, GREEN_TOUT);
(4)
carsGreenNoPed
PEDS_WAITING
(5)
Q_TIMEOUT
(6)
carsGreenInt
carsGreenPedWait
Q_TIMEOUT
(8)
PEDS_WAITING
(7)
Q_TIMEOUT
(9)
carsYellow
entry / signalCars(CARS_YELLOW);
QActive_arm(me, YELLOW_TOUT);
pedsEnabled
exit / signalPeds(PEDS_DONT_WALK);
(10)
pedsWalk
entry / signalPeds(PEDS_WALK);
QActive_arm(me, WALK_TOUT);
Q_TIMEOUT (11)
pedsFlash
entry / me->pedFlashCtr= ...;
QActive_arm(me, FLASH_TOUT);
Q_TIMEOUT[(me->pedFlashCtr&1)==0] /
QActive_arm(me, FLASH_TOUT);
--me->pedFlashCtr;
signalPeds(PEDS_DONT_WALK);
(12)
Q_TIMEOUT[else] /
QActive_arm(me, FLASH_TOUT);
--me->pedFlashCtr;
signalPeds(PEDS_BLANK);
(13)
Q_TIMEOUT [me->pedFlashCtr == 0]
(14)
OFF (15)
ON (17)
offline
offFlash
entry / signalCars(CARS_RED);
signalPeds(PEDS_DONT_WALK);
QActive_arm(me, FLASH_TOUT);
Q_TIMEOUT
Q_TIMEOUT
(16)
offPause
entry / signalCars(CARS_OFF);
signalPeds(PEDS_BLANK);
QActive_arm(me, FLASH_TOUT);

Figure 1.4: A complex pedestrian control system [15]

1.5.1 Hierarchical States

A hierarchical state is a higher-level state that surrounds one or more lower-level states. Hierarchical states or super-states, factor out common behavior from lower-sub states. Thus, sub-states can relay events and transitions to super-states, which ultimately avoids state and event repetition.

In addition, nesting states into super and sub states, restricts the visibility of a system to a specific level, which can be extremely useful for debugging. If needed, a system can be zoomed in or zoomed out to reveal some of its components. For example, in figure 1.4, two main super states **OPERATIONAL** and **OFFLINE** display the first level of abstraction for a pedestrian light controlled system. Moreover, it can also be observed, that these super states contain several other sub-states that show different levels of abstraction.

1.5.2 Events and Transitions

An event is a temporal or spatial trigger that causes a transition between states. This is often represented graphically as an arrow with the event name on it as shown in figure 1.4. Usually after an event transition, **ENTRY** and **EXIT** actions take place. These actions can setup and cleanup state transitions smoothly.

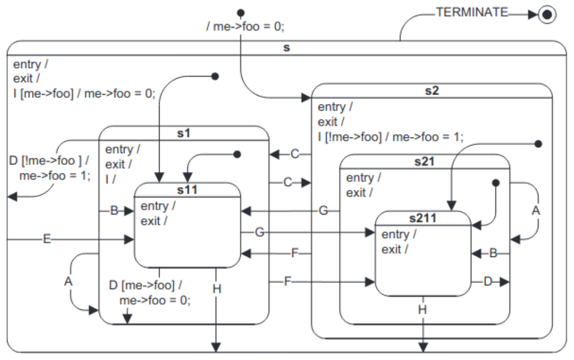


Figure 1.5: Set of transitions and events between states at different levels [15]

For hierarchical state machines, in figure 1.5 notice how events can be directly propagated to four different state levels, $s1 \rightarrow s2$, $s2 \rightarrow s11$, $s21 \rightarrow s11$ and $s11 \rightarrow s211$. Indeed, this is a unique feature of hierarchical state machines, however, transitions at different hierarchies, will again degrade into a traditional FSM; so, a design with fewer transitions between levels is preferred.

1.5.3 Conditional Guards

For some state transitions, boolean conditional statements are needed when there is a fork between two or more state transitions. In the case of extended state machines, this results in an if conditional statement that is graphically represented as a diamond as shown in figure 1.6. This conditional statement is what is known as a conditional guard, and it is used to select transitions between states based on events.

Guard conditions are sometimes unavoidable, and are often part of the design requirement of a state machine. However, too many guard guard conditions, can reduce the quality of the state machine, as it will

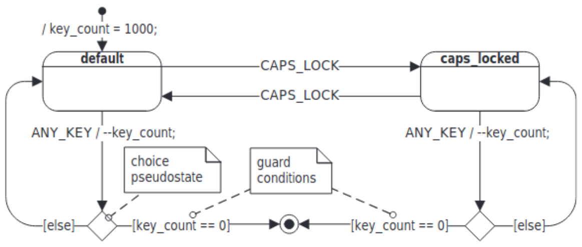


Figure 1.6: Example of a guard condition for a state machine [15]

become harder to maintain and extend.

1.5.4 Orthogonal States

An orthogonal state can be described as the logical **AND** decomposition of hierarchical states [15]. Similarly, a clustered hierarchical state can be thought of an logical **OR** decomposition. Figure 1.7 shows a state machine for a keyboard [15] with two orthogonal regions; **MAIN_KEYPAD** and **NUMERIC_KEYPAD**. In this example, the keyboard state machine is initially in two different states, **DEFAULT** AND **NUMBERS**, which represent the **AND** decomposition previously mentioned. Additionally, notice that the **MAIN_KEYPAD** orthogonal state can be at either the **DEFAULT** OR **CAPS_LOCKED** state. Finally, orthogonal state's events do not cause transitions on other orthogonal regions, which is also why orthogonal states show state and event independence.

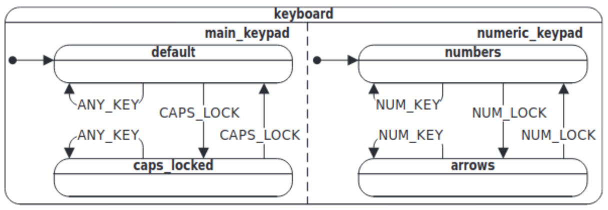


Figure 1.7: Simple keyboard state machine with two orthogonal regions [15]

1.6 Deep Learning and Neural Networks

Attaching some sort of AI to a robot system has been the ultimate goal for the majority of robotic applications. Human-like behavior for robots could greatly increase the safety and utility of a great deal of human-specific jobs. Furthermore, AI intends to increase the level of autonomy of a robot by supplying the decision making mechanisms typically found on a human or biological organism. This translates to an increased level of perception that assists on the selection of adequate actions. Altogether, AI modifies a robot's behavior through a higher level of perception and decision making that is based on sensory inputs.

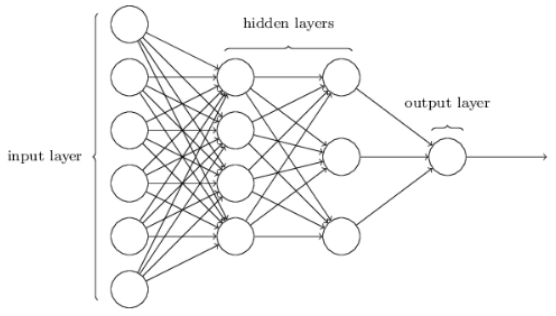


Figure 1.8: Neural network with multiple hidden layers[17]

Recently, there has been a great deal of advances on a subset field of Machine Learning (which is subsequently a subset of AI) known as Neural Networks. These type of AI algorithms seek to mimic the behavior of the human brain using a collection of interconnected artificial neurons known as perceptrons. Neural networks can contain several hidden layers between the input and output layers as shown in figure 1.8 [17]. This type of network is known as a Deep Learning Neural Network, and it can solve all sorts of interesting machine learning problems [18]. In this work, an object recognition neural network was used.

The following sections provide the relevant background information related to deep learning and neural networks. The first section briefly explains neural networks with some mathematical background. This section is then followed by a short introduction on gradient descent and backpropagation. Finally, this section concludes with background information on deep learning and convolutional neural networks, both of which compose most of modern neural networks applications.

1.6.1 Neural Networks

1.6.1.1 Perceptron

Neural networks are made of multiple layers of perceptrons. A perceptron is an idealized mathematical model of a biological neuron that takes several input values and produces a binary output based on the weighted sum of the inputs [17]. Figure 1.9 shows a perceptron model with three inputs x_1, x_2, x_3 and one output y.

Generally, the output y of a perceptron with inputs $x_1, x_2, ..., x_n$, weights $w_1, w_2, ..., w_n$ and threshold value b is given by the following functions:

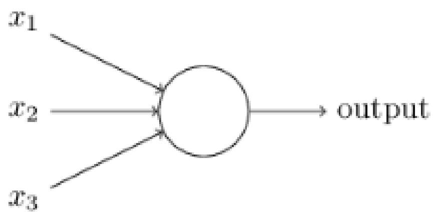


Figure 1.9: Visual representation of a perceptron model[17]

$$y = \begin{Bmatrix} 0, & \text{if } \sum_i^n x_i w_i \leq b \\ 1, & \text{if } \sum_i^n x_i w_i > b \end{Bmatrix} \tag{1.2}$$

Equation 1.3 can be compactly expressed in vector notation by $\mathbf{x} = [x_1, x_2, ..., x_n]$ and $\mathbf{w}^\top = [w_1, w_2, ..., w_n]$. Subsequently, the threshold b can be move to the right hand side of the equation to produce the following equations:

$$y = \begin{Bmatrix} 0, & \text{if } \mathbf{x} \cdot \mathbf{w}^\top + b \leq 0 \\ 1, & \text{if } \mathbf{x} \cdot \mathbf{w}^\top + b > 0 \end{Bmatrix} \tag{1.3}$$

1.6.1.2 Activation Functions

Nevertheless, fully interconnected perceptron neural networks cannot solve complex problems due to the binary nature of its output. This is due to the fact that small variations on weight values can cause large variations on the neuron's output. As a consequence of this, training a perceptron network, proves to be a complicated task as it is difficult to find the appropriate weights that produce the desired output.

Hence, smooth and continuous activation functions are usually preferred over threshold output values. An optional activation function, is the sigmoid function $\sigma(z)$ shown in figure 1.10, which takes as input the dot product $z = \mathbf{x} \cdot \mathbf{w}^\top$. The sigmoid function is defined by the following equation:

$$\sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-(\mathbf{x} \cdot \mathbf{w}^\top)}} \tag{1.4}$$

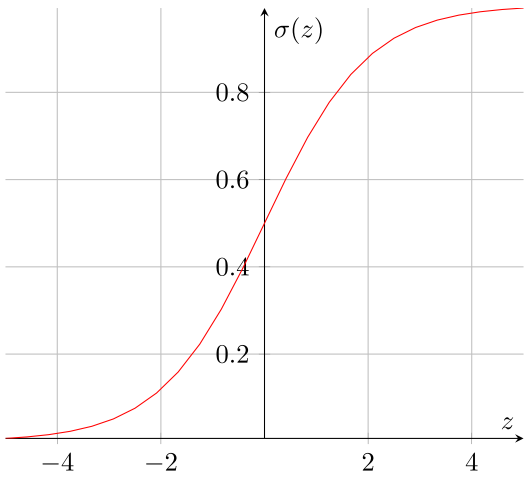


Figure 1.10: Sigmoid function graph

A more popular choice for an activation function is the piecewise linear function known as the Rectified Linear Unit (ReLU). The ReLU activation function has become the de-facto standard because it shows better training performance compared to a sigmoid function. Other useful activation functions found on neural networks are the Tanh, Hard Tanh and Sign functions, though with the exception of the hard tanh, these are not widely used on modern networks [19].

1.6.1.3 Neural Network Layers

Similarly to the input and output neuron layers, a hidden neuron layer n can be expressed as a vector $\mathbf{a^{(n)}} = [a_1^{(n)}, a_2^{(n), \ldots, a_m^n}]$. Hidden layers are interconnected between the input, output and hidden layers through a set of weights. Figure 1.11 shows the input layer $\mathbf{x}$ connected to the first layer $\mathbf{a^{(1)}}$ through a set of weights $\mathbf{w^{(1)}} = [w_{1,1}, w_{1,2}, ..., w_{1,n}]$, and a bias vector b.

On closer inspection of neuron $a_1^{(1)}$, it can be seen that its output $a_1^{(2)}$ can be given by,

$$a_1^{(2)} = \sigma(x_1 w_{1,1} + x_2 w_{1,2} + \ldots + x_n w_{1,n} + b_1^{(1)}) = \sigma(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \cdot \begin{bmatrix} w_{1,1} & w_{1,2} & \ldots & w_{1,n} \end{bmatrix} + b_1^{(1)}) \quad (1.5)$$

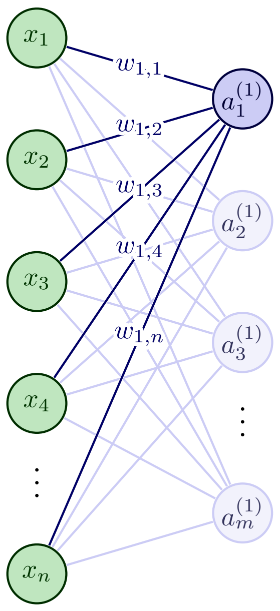


Figure 1.11: A single input layer connected to a hidden layer

Or in a more compact notation,

$$a_1^{(2)} = \sigma(\mathbf{x} \cdot \mathbf{w^{(1)}} + b_1^{(1)}) \tag{1.6}$$

It then follows that the all neuron outputs for this layer can be represented in matrix notation as

$$\begin{bmatrix} a_1^{(2)} \\ a_1^{(2)} \\ \vdots \\ a_m^{(2)} \end{bmatrix} = \sigma(\begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & \dots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \dots & w_{m,n} \end{bmatrix} \begin{bmatrix} a_1^{(1)} \\ a_1^{(1)} \\ \vdots \\ a_m^{(1)} \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_1^{(1)} \\ \vdots \\ b_m^{(1)} \end{bmatrix}) \tag{1.7}$$

Similarly as for a single neuron, this can be re-written more compactly as,

$$\mathbf{a^{(2)}} = \sigma(\mathbf{W^{(1)}}\mathbf{a^{(1)}} + \mathbf{b^{(1)}}) \tag{1.8}$$

In general, these equations relate an input layer $\mathbf{a^{(n-1)}}$ to an output layer $\mathbf{a^{n}}$, through a weight matrix $\mathbf{W^{(n)}}$ and bias vector $\mathbf{b^{n}}$

1.6.1.4 Cost Functions

A cost function tries to minimize the error between the output value of the neural network $\hat{\mathbf{y}}$ and the true value of the output $\mathbf{y}$ given the set of inputs $\mathbf{x}$ for a data set of size n. To quantify the error over a presented dataset, a quadratic or Mean Squared Error(MSE) function is used [17]. The Mean Squared Error is represented by the following formula:

$$C(\mathbf{W}, \mathbf{b}) = \sum_{k=1}^{n} (\mathbf{y_k} - \hat{\mathbf{y}}_\mathbf{k})^2 = \sum_{k=1}^{n} (\mathbf{y_k} - \sigma(\mathbf{W}\mathbf{x_k} + \mathbf{b}))^2 \tag{1.9}$$

In practice, several other cost functions are used to quantify the networks error. A few examples are cross entropy, Hinge and Huber cost functions.

1.6.1.5 Gradient Descent

A neural network can be trained to produce a desired output $\mathbf{y}$ with the correct selection of weights $\mathbf{W}$, bias $\mathbf{b}$ and input $\mathbf{x}$. However, it will be impractical to assume that these weights can be found experimentally, due to the large number of weights and biases. Therefore, to automate the process of finding the correct weights, and thus, train a network, there must exist an operation that minimizes the output of the cost function $C(\mathbf{W}, \mathbf{b})$. Through careful observation, it can be noted that incremental weight changes must be performed to reduce the output of the cost function 1.9. This can be mathematically expressed through the use of a gradient descent algorithm [17],

$$\mathbf{W_{k+1}} = \mathbf{W_k} - \eta \nabla C \tag{1.10}$$

$$\mathbf{b_{k+1}} = \mathbf{b_k} - \eta \nabla C \tag{1.11}$$

This iterative process attempts to find a global minima on the multi-variable cost function 1.9 through the process of choosing a new set of slightly modified set of weights and biases. Finally, the parameter η is known as the learning rate, and it plays a crucial role on the effectiveness of a trained neural model.

It can also be noted that there is a large number of weight value corrections that must be computed on every iteration. Therefore, for practical networks, random batches of inputs are selected on every training cycle. This is method is known as stochastic gradient descent and it is largely used on modern neural network architectures.

1.6.1.6 Backpropagation

The previous section demonstrated that weights and biases can be adjusted so that a cost function $C(\mathbf{W}, \mathbf{b})$ can be minimized. Equations 1.10 and 1.11 required the computation of the gradient of the cost function $\eta \nabla C$. Subsequently, to find the gradient, the partial derivatives of $\frac{\partial C}{\partial W}$ and $\frac{\partial C}{\partial b}$ must be computed over all weight and bias values. Computing $\eta \nabla C$, is therefore, inherently expensive, as it requires several arithmetic operations per weight.

The backpropagation algorithm was introduced as a way to speed up the process of computing the gradient of the cost function ∇C. The core of the backpropagation algorithm is solve the partial derivatives $\frac{\partial C}{\partial W}$ and $\frac{\partial C}{\partial b}$ for every weight and bias on the network [17], through the use of the cost function $C(\mathbf{W}, \mathbf{b})$. The backpropagation takes the local error at the left-most or output layer, and then it uses this error to modify the weights at the previous layer. This is then repeated throughout all layers in the network. The four fundamental equations for backpropagation are [17]:

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{1.12}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^L) \tag{1.13}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{1.14}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{1.15}$$

1.6.1.7 What is deep learning?

Modern neural networks are capable of decomposing complex problems using deeply nested neuron layers. Through the use of several layers, a network model can learn to identify specific features on a dataset. For example, for the image of a human face, a deep neural network can learn to identify distinctive features that make up a face [17]. These can be thought of as a set of questions that the neural network asks; *is there a nose?*, *is there a mouth? And what about eyes?*.The scope of these questions get narrower as the data progresses through the network. Using the same human face analogy, a deeper layer could be attempting to answer the following questions; *Is there an eyebrow?*, *is there an iris?*, *what about a pupil?*. Eventually, patterns down to individual pixels could be recognized by a deeply nested network. Deep learning, thus, is a neural network with multiple layers that is able to recognize patterns at multiple levels.

However, multiple layered networks present a challenge for large datasets such as images. The number of weights parameters increases exponentially, which causes the training process to become more difficult. As a result, development of new techniques such as Convolutional Neural Networks have been the result of pushing the boundaries of state-of-the-art deep learning algorithms.

1.6.2 Convolutional Neural Networks

Traditional deep neural networks take an input vector $\mathbf{x}$ as a set of uncorrelated values [17]. For instance, an image with a resolution of 28 by 28 pixels can be decomposed as a vector of length of $28x28 = 784$, which correspond to an input layer of 784 neurons. This traditional approach will give equal importance to every single pixel input, with the same equal importance on its weights and bias values. As a result of this, this network architecture will need to identify per pixel patterns that determine features for a true output value. Moreover, this network does not consider the spatial structure of the data. For this reason, Convolutional Neural Networks were introduced as an alternative to traditional neural networks.

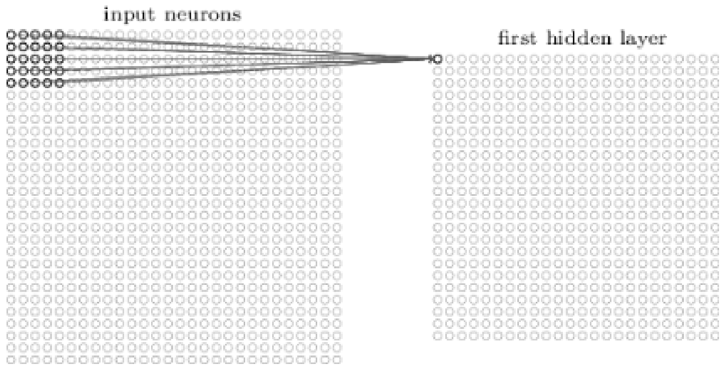


Figure 1.12: Convolutional neural network with local receptive fields[17]

A Convolutional Neural Network uses local receptive fields, shared weights and pooling to detect spatial features on a set of input data [17]. These networks are well-suited for image classification and segmentation, as they can detect local features. Furthermore, convolutional networks can be trained faster than traditional neural networks due to their lower number of weight parameters. Figure 1.12 shows the basic principle of what a convolutional network is and how it extracts local features through the use of its receptive fields.

1.7 Thesis Outline

Chapter one presents the background study, research problem, questions, objectives and significance. This chapter establishes the research topic and why it is important in the context of autonomous inspections.

Chapter two will present the literature review of path planning for drones followed by a review of deep learning neural network applications on autonomous drones. This chapter identifies key innovations used on autonomous inspections.

Chapter three discusses the methodology and simulations used in the development of the work presented on the following chapters. This chapters serves as an introduction to the tools and ideas for the software development of this thesis.

Chapter four presents the methodology used to develop a innovative drone navigation stack that can avoid both static and dynamic obstacles. In addition, a priority oriented global path planner is also presented in this chapter.

Chapter five will introduce the methodology used to develop an object recognition and localization system using deep learning neural network and QR codes.

Chapter six presents the methodology developed to create inspection waypoints on octomaps as well as on object and QR detections.

Chapter seven shows the conclusions and a case study that demonstrates an inspection mission using the proposed methods and software.

Chapter 2: Literature Review

2.1 Drone Inspection

UAV automated inspection systems have recently gained research popularity due to the abundance of hardware and software tools to perform safe and repeatable flights. Plenty of drone platforms have been proposed to solve a great deal of inspection tasks. For instance, Banic et al. [20] proposed a UAV vision system for railway infrastructure inspection and monitoring using a single derivative Canny Edge detection and a K-nearest neighbor algorithm to detect and classify rail track damage. A similar study by Mao et al. [21] developed a drone detection system using Histogram of Oriented Gradients (HOG) and a Support Vector Machine (SVM) for the inspection of power transmission lines. The SVM model was assisted by a Principal Component Analysis (PCA) algorithm to increase the processing speed at the cost of accuracy. This study determined that HOG and PCA assisted SVM can reach an accuracy of 85.2% for the the detection of damaged power lines. In both of these cases, in can be observed that the computer vision algorithms are not part of the flight strategy of the drone, and are only used as post-processing tools for detection. Moreover, these are algorithms are highly specialized for their intended application, which means that these methods are not portable to other inspection tasks.

Other inspection methods still rely on a human operator to flight and inspect an area. Such is the case for the study done by Marecelo Teixeira et al. [22] where they proposed a drone for structural inspection that can be controlled through a set of head and motion gestures using the google glasses. Moreover, this drone directly streams the captured video to the glasses, so that the user can visually inspect a building. A similar example was proposed by Seo et al. [23] with a five stage process to safely and manually inspect a bridge. In their study, Seo et. al used a DJI drone to capture high resolution images that were visually evaluated by a local department of transportation. The study concluded that drone inspections can be performed, but weather conditions and operator flight experience can become an issue for routinely and reliable inspections. These study cases present one end of the spectrum for drone inspections, in which they do not present neither the navigation or detection autonomy for an inspection task. Both tasks rely on a human operator to navigate an environment and detect defects on an input sensor or image. Consequently, these drone platforms are extremely limited from an autonomy standpoint and do not offer a solution to the problem of automated inspections.

Lastly, Irizarry et al. [24] explored the usability of drones as inspection tools for safety, by heuristic evaluations of user interfaces. The aim of this study was to evaluate the feasibility of drone inspections for construction sites through the use of mobile phones and tablets. At the end of their study, Irizarry et al. concluded that the ideal safety inspection drone should have autonomous navigation, voice interaction, environmental applicability, high resolution cameras, a multitasking application, and a collaborative user interface. These set of requirements were found to be lacking on the previous proposed drone systems, and as it will be shown, it can also be demonstrated that most drones lack one or more feature requirements proposed on this study.

Table 2.1: Drone Inspection Systems Review

Author & Year	**Inspection Task**	**Autonomy Level**	**Deep Learning Enabled**
Zahid et. al 2020	Power Lines	Manual	Yes
Mc Aree et al. 2016	General Inspection	Semi-autonomous	No
Zhang et al. 2022	Power Lines	Manual	Yes
Mao et al. 2019	Power Line	Manual	No
Besada et al. 2018	General Inspection	Autonomous	No
Seo et al. 2018	Bridges	Manual	No
Banic et al. 2019	Railways	Manual	No
Ashour et al. 2016	Construction	Autonomous	No
Teixeira et al. 2014	General Inspection	Manual	No
Zefir at al. 2018	Solar Panels	Autonomous	No
Irizarry et al. 2012	Construction	Manual	No
Shivauddin et al. 2019	Wind turbines	Manual	Yes

Nevertheless, other drone inspection research efforts have concentrated on including deep learning algorithms to assist with the detection and classification of defects for several inspection applications. One example is the work of Siddiqui et. al [25], where a remotely controlled drone was used to capture images on high voltage transmission power lines. The images were processed using a deep learning neural network framework to detect and localize damaged power lines. Likewise, Shivauddin et. al [26] also used a remotely controlled drone to inspect wind turbines using a deep learning model. In their work, Shivauddin et. al used the wind turbine images to train multiple neural network architectures, which were able to achieve a Mean Average Precision (MAP) of 90.63%. Additionally, they showed that the precision of the models could be increased by augmenting the training data set. Another study by Zhang et. al [27] demonstrated a traditional HOG and deep neural network to detect damaged power lines with an accuracy of up 97%. Moreover, the authors were able to categorize 14 different power line defects using the mentioned methods.

Clearly, these studies show that deep learning and neural networks have practical applications in the field of automated drone inspections. However, the proposed methods still rely on manual drone flights around the inspection area. Other proposed systems do attempt to automate the inspection by using a GCS for mission planning and logging. An example of such system is presented in the work of Besada et al. [28] where a mission plan system was proposed as an extension to simple waypoint based GCS. Besada et al. proposed a Mission Definition System (MDS) with the goal of aerial infrastructure inspections using high-level mission definition primitives. Their system seeks to facilitate pre-flight mission planning, mission visualization and post-flight evaluation tools. A similar drone system that uses pre-flight mission software is showed in the work of Zefri et al. [29]. In their paper, an UAV was used to inspect photovoltaic installations through the use of thermal and photogrammetry imaging. Both sets of images were processed to generate visual orthomosaics that were used to detect defects on photovoltaic panels. Zefri et al. concluded that their study required imaging sensors with higher resolution due to the small scale of the defects encountered on the panels. A comparable drone system proposed by Ashour et al. [30] uses a GCS connected to a government database to perform routine inspections on constructions sites. The proposed drone platform visually inspects a construction site to detect anomalies that could potentially cause a safety hazard.

The previous proposed drone systems do demonstrate a level of autonomy, however, this is only through the use of a pre-determined flight mission that cannot be changed by the drone. Therefore, a drone will not be able to plan a different mission mid-flight or avoid dynamic collisions. A proposed solution by Mc Aree et. al [31] uses a semi-autonomous drone to inspect buildings with complex geometry features. The drone can assist an operator to maintain a fixed distance to a wall using LiDAR sensor readings. Using this feature, an operator can maneuver the drone around constrained or difficult to reach sections in a building.

2.2 Drone Path Planning

Most automated drone inspection systems work through the use of a GCS system that generates a pre-flight mission based on manually specified waypoints. These kind of systems offer limited navigation abilities for autonomous drones. Thus, path planning algorithms for autonomous navigation has recently attracted numerous researchers. Genetic algorithms have been proposed for path planning, where the goal is to iteratively find a suitable path determined by a certain criteria. Such is the case for the work of Shivgan and Dong [32] where a genetic algorithm was used to find an energy efficient trajectory between a set of waypoints. Shivgan and Dong re-formulated the path planning search as as traveling salesman problem, with the goal to minimize the energy consumption for a planned trajectory. Moreover, the genetic algorithm uses a fitness equation to choose the fittest trajectories between generations. In their results, they showed that a path generated by their genetic algorithm, consumes less energy than a greedy approach by constructing a trajectory between 10 random waypoints. A similar study done by Hayat et al. [33] used a coverage path planning algorithm for multi-UAV search and rescue missions. With the proposed system, drones can disseminate information regarding mission objectives and planning requirements. The multiple drone trajectories are evaluated using a genetic algorithm to choose and re-plan the optimal coverage path. Hayat et al. concluded that their approach improved the completion time for multi-UAV coverage path planning mission. Another coverage path planning and genetic algorithm UAV system is mentioned in [34]. This study employs a coverage probabilistic road-map to construct a graph which is subsequently used to reformulate a min-max set-covering vehicle routing problem. To solve the problem, a modified genetic biased key genetic algorithm was used to generate collision-free trajectories for complex structure geometries with paths that are up to 48 times shorter than other methods.

Graph search algorithms are also popular methods used to solve path planning problems. An example is the work of Zompas [35] where he proposed a path planning system using a visibility graph and A* given a previously obtained map with the octomap framework. In his results, Zompas showed that his visibility graph could be constructed in 7.94 seconds with a search time of 0.24 seconds using A*, which demonstrates a practical use case for online and real-time path planning. A similar path planning system that uses the octomap software is presented by Bergstrom [36], where he worked on a path planning system that constructs a graph using a cell decomposition method. Furthermore, Bergstrom applied weight penalties to graph vertices that were closer to a wall. With this method, Bergstrom uses an A* search algorithm to find a collision-free path, while also keeping a safe distance from walls. A final example is provided by Fangyu et al. [37] using a voxel grid system and a novel 3D propagating approximate Euclidean distance transformation to plan trajectories in known environments. In their work, they proposed an A* searching algorithm to find the safest short path and the safest least cost path, both of which ensure a minimal distance to an obstacle.

As mentioned on 1.4.4, random based graph searches can be useful when large or higher dimensional configuration spaces are used. For instance, Oleynikova et al. [38] proposed a collision avoidance algorithm using a high-degree polynomial made out of several segments with the assistance of a RRT* planning algorithm. The RRT* algorithm was used to quickly re-plan rough trajectory estimations that could be latter refined using their continuous polynomial algorithm. Both algorithms can generate a new trajectory plan quickly, which is ideal to perform collision avoidance at higher frequencies. In contrast, Zheng et al. [39] explored the possibility of performing path planning directly on point clouds without conversion to a traditional mapping format. The proposed path planning system uses kd-trees and RRT to perform collision avoidance on a down sampled point cloud.

In [40], a stochastic model to characterize the path traversal time using the transportation network was proposed. In this work, the authors also consider the battery life of the drone so that the flight time can be maximized according to the feasibility of a produced path. Another unique method explained by Qin at al. [41] uses both a UAV and UGV to map an environment without the use of a Global Positioning System

Table 2.2: Drone Path Planning Review

Author & Year	**Path Planning Method**	**Mapping System**
Qin et. al 2020	Batch Informed Trees (BIT*-H)	Octomap
Oleynikova et al. 2016	Continuos Time Trajectory Spline + RTT*	Octomap
Zompas 2016	A* + visibility graph	Octomap
Shivgan et al. 2020	Genetic algorithm	N/A
Hayat et al. 2019	Coverage Path Planning + Genetic algorithm	N/A
Jing et al. 2020	Genetic algorithm	Octomap
Bergestrom 2018	A* + cell decomposition	Octomap
Zheng et al. 2020	RRT + k-d tree	Raw Point Cloud
Huang et al. 2021	Stochastic model	Transportation Network
Fangyu at al. 2018	A* + Distance Transformation	Voxel Grid

(GPS). Both platforms work jointly to produce a map using the octomap software. Moreover, the UAV and UGV use a frontier algorithm to explored unmapped Sections of the environment. For this task, a Batch Informed Tree (BIT*-H) was used to for path planning.

Chapter 3: Methodology and Simulations

Drone flight missions can be time consuming and in some cases even dangerous, especially at the software prototyping stage. Therefore, for software development and testing, a drone dynamics and visualization simulation was used. In addition, flight control software was used to maintain position set-points of the aircraft, as well as to communicate with other external software components using the well established Mavlink protocol [42]. With the simulation software bundle, external software modules were developed to perform basic navigation, path planning, object and QR detection, and waypoint generation. Moreover, software unit testing tools were used to carry out repeatable drone flight mission scenarios.

In this chapter, the software simulation components are briefly introduced and explained. Moreover, the methodology for the software design and testing is also presented in this chapter. Lastly, this chapter establishes the tools and methods used to developed the software system components presented in the following chapters.

3.1 Simulation Environment

This work used a simulation environment to visualize and test software modules. The simulation environment is composed of the following software elements:

1. Drone dynamics and visualization (section 3.1.1)
2. Drone sensors and algorithms visualization (section 3.1.3)
3. Drone Software In The Loop (SITL) flight controller (section 3.1.2)
4. Drone software communication protocol (section 3.1.4)

In conjunction, these components can simulate a flight mission on a variety of user defined environments by computing a dynamical model of a drone and then visualizing the output using a Graphical User Interface (GUI). The dynamic model is controlled by the SITL flight controller, which is the same software that will be deployed on the drone's hardware. Moreover, a different software tool is used to visualize the drone's sensor and algorithms output form the developed software modules.

Lastly, the developed software uses a communication protocol to command the SITL flight control to take a specific action. Consequently, this sets the inputs to the drone dynamic model. On a final note, simulation parameters such as wind, friction and gravity can de defined for an environment. Evidently, several case scenarios can be performed on the same environment. Figure 3.1 shows the software simulation components used to test the work presented in this thesis.

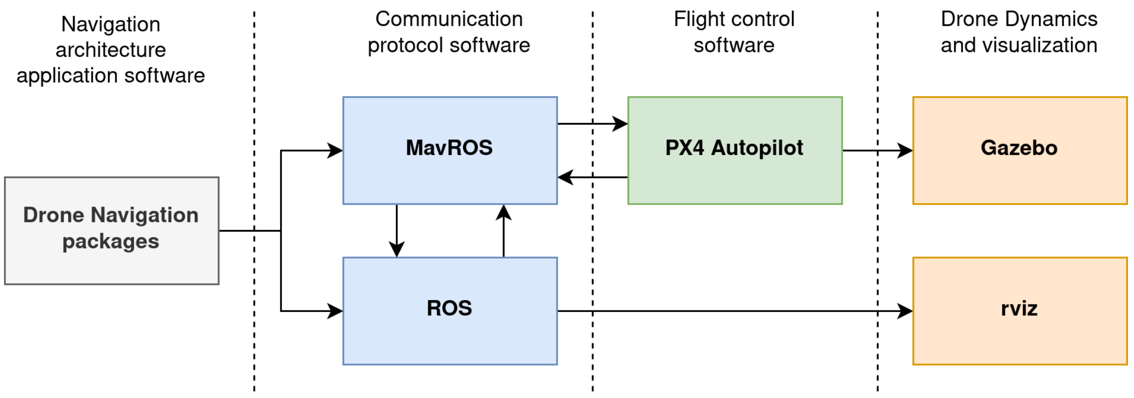

Navigation architecture application software
Communication protocol software
Flight control software
Drone Dynamics and visualization
Drone Navigation packages
MavROS
ROS
PX4 Autopilot
Gazebo
rviz

Figure 3.1: Simulation software components diagram

The next sections introduce and briefly explain each of the software tools used to create and deploy a drone flight mission, as well as the tools used to develop the software packages presented in latter chapters.

3.1.1 Gazebo

Gazebo is a simulation and visualization software tool for dynamic robotic models. Gazebo's Application Programming Interface (API) allows developers to design and test robotic models with the help of a physics simulation library and a GUI. In addition, gazebo uses a plugin system that accepts simulation models for a great variety of known and established mechanical and sensor models. Through the use of model and world files, a simulation environment can be defined using a mesh, collision and physical models.

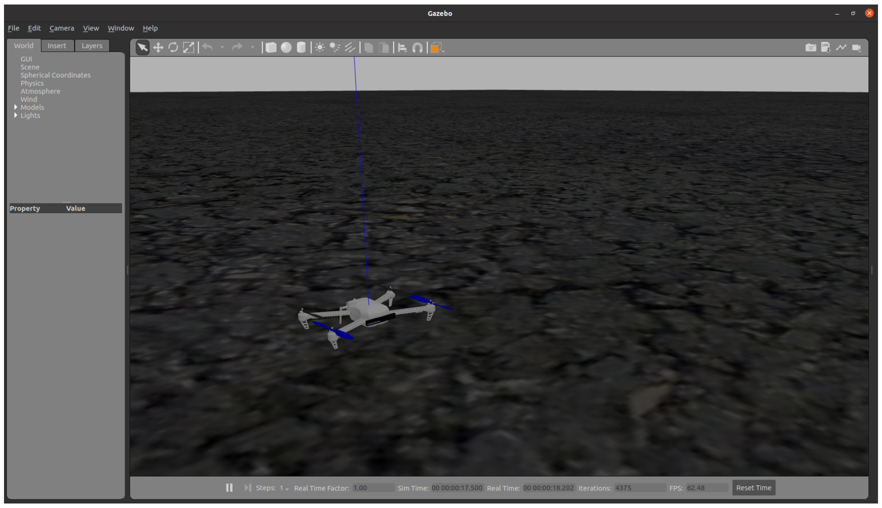


Figure 3.2: Gazebo robot simulation environment for drone simulations

For this work, a set of pre-defined models, worlds and plugin files [43] were used to simulate a dynamic drone model on several environments. The drone simulation model uses a plugin system for all of its components. The plugins define the behavior and physical properties of components such as the Inertial Measurement Unit (IMU) and the GPS. Evidently, with the plugin system, the drone software model components can be added or removed as needed. Finally, the robot can be visualized using a mesh and texture models. This is useful to visualize and debug the robot's behavior in real-time. Figure 3.2 shows the GUI for the gazebo robot simulation software suite.

3.1.2 PX4 Autopilot Software

For low-level hardware control, the PX4 autopilot software was used[1]. This software runs directly on the flight control computer and it is used to stabilize the attitude, position and velocity of the aircraft. Alternatively, the PX4 autopilot software can be deployed as a SITL on other hosts machines through the use of simulated sensor models.

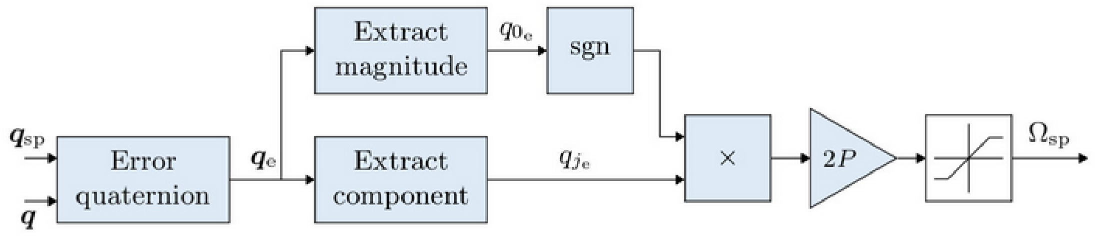


Figure 3.3: Attitude control diagram for the PX4 autopilot software [43]

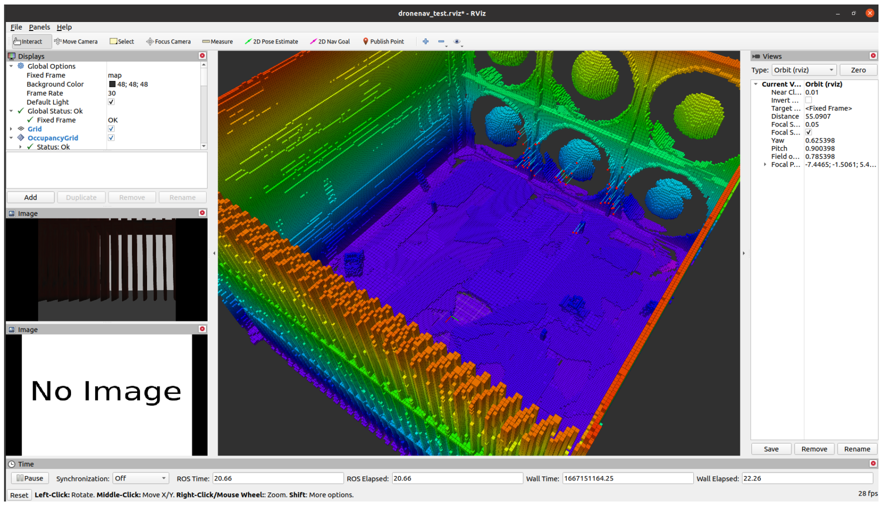
Indeed, this computer must follow hard timing deadlines for control, therefore, real-time performance is needed. To achieve this, the autopilot software first fuses a set of sensor measurements using the Extended Kalman Filter (EKF). The fused measurements are used to control the attitude of the aircraft using the architecture shown in figure 3.3. Additionally, the PX4 software handles low-level communication with an off-board computer using the Mavlink protocol. Using the Mavlink protocol, the off-board computer can command the flight computer to perform tasks such as taking off, landing and moving to certain locations.

3.1.3 rviz

The rviz software tool is used to visualize sensor and user defined primitive markers. Data from multiple sensors such as depth, RGB cameras, LiDAR scanners, encoders, and others, can be displayed selectively using topic messages. Additionally, primitive markers can be defined through the use of a list of 3-dimensional

[1]The PX4 project page can be found at `https://px4.io/`

points. These markers display data from developed software and algorithms, and serve as a debugging and visualization tool. For example, the navigation waypoints and computed path trajectory can both be displayed using markers as shown in figures 4.3 and 4.5. Figure 3.4 demonstrates the GUI used to select the sensor and marker topic messages.



```
Global Options
  Fixed Frame         map
  Background Color    48; 48; 48
  Frame Rate          30
  Default Light       ✓
✓ Global Status: Ok
  ✓ Fixed Frame       OK
Grid                  ✓
OccupancyGrid         ✓
  ✓ Status: Ok
```

```
Type: Orbit (rviz)
Current V...   Orbit (rviz)
  Near Cl...   0.01
  Invert ...
  Target ...   <Fixed Frame>
  Distance     55.0907
  Focal S...   0.05
  Focal S...   ✓
  Yaw          0.625398
  Pitch        0.900398
  Field o...   0.785398
  Focal P...   -7.4465; -1.5061; 5.4...
```

```
Synchronization: Off   ROS Time: 20.66   ROS Elapsed: 20.66   Wall Time: 1667151164.25   Wall Elapsed: 22.26
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.   28 fps
```

Figure 3.4: rviz was used to visualize software output and sensor data

3.1.4 Mavlink and Mavros

The Mavlink protocol is used for low-level communication between system components, GCS and off-board computers. The protocol defines the low-level binary encoding used to transfer data packages between devices. The protocol is lightweight, with embedded systems as the intended host devices. Mavlink is also a flexible protocol that can be expanded or modified with user defined messages. Lastly, the Mavlink protocol uses a publish/subscribe software architecture using messages and topics. This architecture allows for reactive message transfers that will be broadcasted to software components or off-board computers that need the message data.

The Mavlink protocol was latter ported to be used along with ROS (see section 3.1.5), which it then became known as MavROS. MavROS uses a similar publish/subscribe software model to communicate with a off-board flight control computer, but unlike the Mavlink protocol, additional control software can be developed to modify the behavior of the drone.

3.1.5 ROS

ROS is a set of software tools used to develop and test complex robotic systems with multiple sensors and actuators. Similar to the Mavlink protocol shown in section 3.1.4, ROS uses a publish/subscribe software model. The publisher and subscriber nodes use topics to post and receive messages. The software model used by ROS captures the reactive and modular nature of most robotic platforms. For instance, with ROS, sensor drivers can be developed as publisher nodes that broadcast a message with sensor data. One or more subscriber nodes can listen to this message, and will react once a new message has been received.

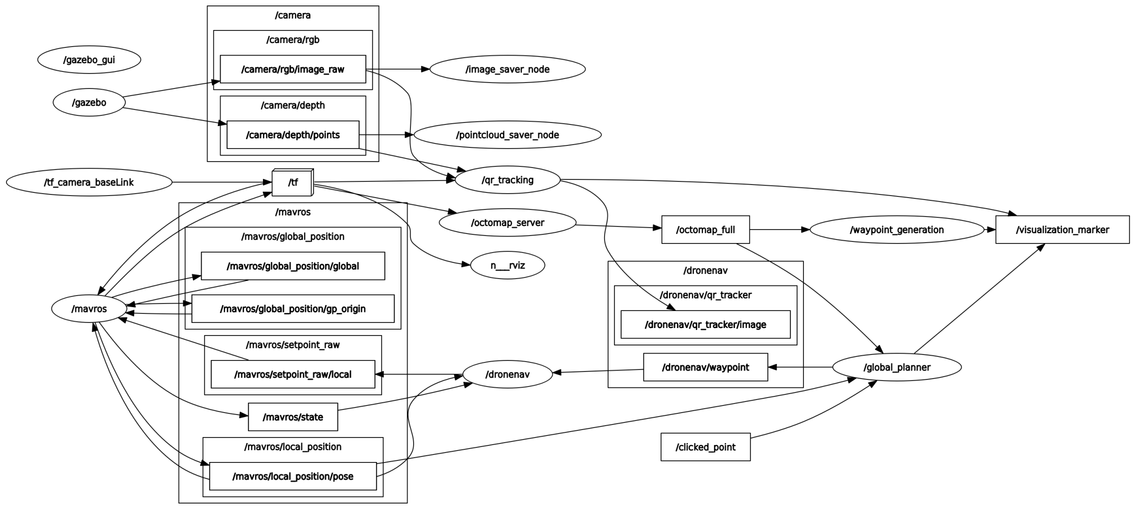


Figure 3.5: ROS node graph for the proposed drone navigation architecture and marker detection

The proposed drone system uses six ROS packages to divide the functionality into modular and maintainable software modules. Each package is composed of one or more nodes that publish and subscribe to different message topics. For example,the drone uses a message structure as shown in table 3.1 to publish waypoint data that positions and orients the drone. The combination of topics and messages produces complex, event driven and modular robotic software platforms with multiple interactions between its components. ROS has a way to visualize these interactions through a graph network diagram. Figure 3.5 shows the nodes and their interactions for the inspection drone system.

Table 3.1: Topic message used on the dronenav package

Waypoint message		
Field	**Data type**	**Description**
action	uint8	Action flag for this waypoint
flags	uint8	Navigation flags for the drone and path planner
position	Point	Waypoint position in x, y, z coordinates
normal	Point	Normal vector of this waypoint
yaw	float64	Yaw angle of this waypoint

3.2 Software Methodology

ROS software tools allow distinct robot components to be separated into modular units called nodes. However, ROS does not specify how nodes should communicate or what messages should be used. Furthermore, packaging and organizing nodes into common functionality is not a trivial task. Consequently, nodes that share common functionality or perform similar tasks, should be packaged together. Nevertheless, identifying common functionality is an iterative task that was performed throughout the software development process.

Additionally, each ROS package should be designed according to specific methodologies and philosophies in mind to fulfill usage and test requirements for practical use and deployment. The methodologies used in this study for the software development process were:

1. Each package should be fully modular and independent of the inner workings of other packages.
2. Each package should be responsible of one and only one task.
3. Each package should have a test suite for one or more test cases.
4. A package should be parametrized so that it can be adapted to several use cases and user customization.
5. Packages should only communicate with each other through the use of pre-defined messages

The following sections present the software architecture design used to create the packages and nodes needed for the drone inspection platform, as well as the testing platform used to check the functionality and validity of the programs.

3.2.1 Software Architecture

As mentioned in section 3.1.5 six ROS packages were developed to modularize distinct software components needed for the drone inspection system. The packages along with a brief description is presented in table 3.2

Table 3.2: Drone Inspection ROS software packages

Package	**Description**
dronenav	Core drone navigation package
dronenav_actions	Drone inspection actions package
dronenav_msgs	Drone system message definitions
dronenav_tests	Drone system integration and tests
global_planner	Path planning for octomaps package
object_recognition	Object recognition and tracking package
qr_tracking	QR code detection and tracking package

Each package consists of one or more ROS nodes that communicate with other package nodes. Custom drone inspection messages are defined on the *dronenav_msgs* and are shared for all other packages. Furthermore, ROS nodes can contain a number of *Publisher*, *Subscriber*, *Services*, and *Actions*. For example, the *dronenav* package defines the a ROS node with a **Landing** *Service* server that can be called from a terminal. The same *dronenav* node also exposes the **Waypoint** *Publish* topic that can be used to queue navigation waypoints. Lastly, ROS actions are used for the *dronenav_actions* package nodes to command and receive feedback on the progress of an action (see section 6.4).

To launch and specify parameters for all nodes, an XML launch file was used. This file specifies what and how the nodes should be launched for specific packages through the definition of parameters and pre-launch tasks. Once nodes are launched, topics, services and actions are exposed to a terminal, which can subsequently be used to post and call messages that will modify the drone's behavior reactively.

3.2.2 Software Testing

Software testing is an important part of quality software design. Additionally, testing assists in writing code examples and documentation. For that reason, software unit testing was a crucial component in the development process.

To test packages, the *rostest* and *gtest* software testing framework were used. Using *rostest*, a test suite fixture can be defined, where the setup and cleanup of a test case is done. For the case of a ROS package, this means that *Publishers*, *Subscribers* and *Services* infrastructure is defined before running a test case. Once completed, messages can be posted to topics that will cause the drone to perform a specific action. The validity of the execution can then be tested using **ASSERTION** or **EXPECTED** values. One example of such case is presented in listing 3.1, where a test is performed to check whether the drone was commanded to a particular position and orientation.

Listing 3.1: Node software unit test for the dronenav package

```
TEST_F(DronenavTestFixture, dronenav_test)
{
  /*Takeoff request*/
  dronenav_msgs::Takeoff takeoff_srv;
  /*Request takeoff and test if request was successfully sent*/
  takeoff_client.call(takeoff_srv);
  ASSERT_TRUE(takeoff_srv.response.success);

  /*Wait until we have take off and in position*/
  wait_for_state(dronenav_msgs::Status::HOVERING_STATE, 10.0);
  EXPECT_STREQ(status.state.c_str(), dronenav_msgs::Status::HOVERING_STATE.c_str());

  /*Send a waypoint*/
  waypoint_pub.publish(waypoint);

  /*Check if drone is in the correct position*/
  EXPECT_NEAR(status.current_position.x, waypoint[0].position.x, 0.1);
  EXPECT_NEAR(status.current_position.y, waypoint[0].position.y, 0.1);
  EXPECT_NEAR(status.current_position.z, waypoint[0].position.z, 0.1);
  EXPECT_NEAR(status.current_yaw, waypoint[0].yaw, 0.1);

  /*Land request*/
  dronenav_msgs::Land land_srv;
  /*Request land and test if request was successfully sent*/
  land_client.call(land_srv);
  ASSERT_TRUE(land_srv.response.success);

  /*Wait until we have landed*/
  wait_for_state(dronenav_msgs::Status::LANDED_TOUCHDOWN_STATE, 10.0);
  EXPECT_STREQ(status.state.c_str(),
    dronenav_msgs::Status::LANDED_TOUCHDOWN_STATE.c_str());
}
```

Chapter 4: Drone Navigation Architecture

A drone navigation architecture is the integration of distinct systems with the goal of orienting and positioning a drone without colliding with static or dynamic obstacles. Furthermore, a navigation architecture seeks to provide a higher level trajectory strategy so that the drone can complete a specific mission. Indeed, for an inspection mission, the goal is to detect and record defects, which is often solved by a coverage path mission.

However, as shown in 2, most inspection missions are pre-planned at the pre-flight stage, using a set of fixed waypoints. This is problematic from a navigation standpoint for two reasons. First, the pre-planned mission is often static and not flexible enough to react to specific flight events, such as defect detections and/or collision events. Second, the drone cannot use the environment to plan a new trajectory strategy based on external events. Both of these issues are not addressed in the literature and on most inspection drones. Moreover, the proposed drone navigation architectures are fairly limited to just path planning, and do not look to add complex behavior aside from finding collision-free trajectories

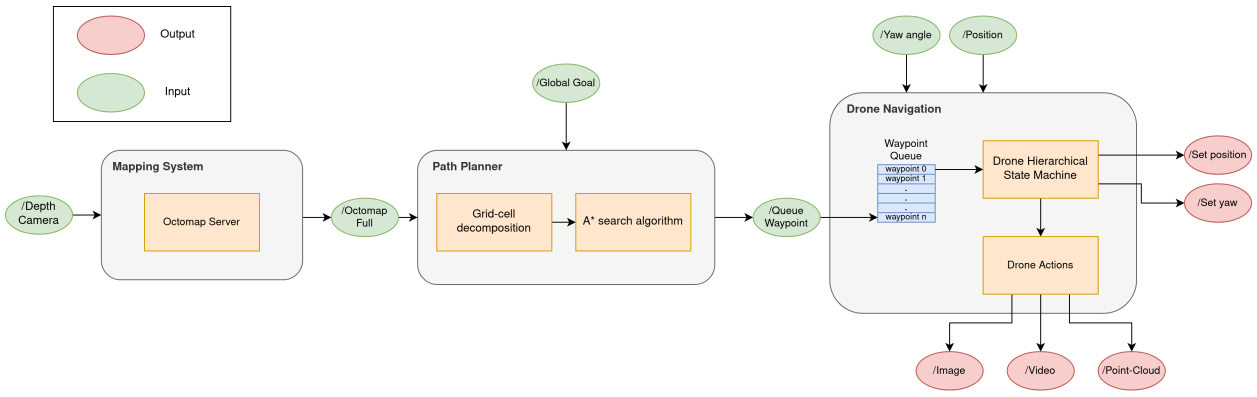


Figure 4.1: Drone navigation system

Therefore, this chapter proposes a novel navigation architecture that focuses on re-planning paths caused by external events such as defect detections captured by an imaging sensor. The proposed navigation architecture uses the octomap framework to generate a map that can be used for path planning. The path planner then uses this map to construct and search a graph for a valid path. Finally, a hierarchical state machine handles the events and states for both the drone positioning and orientation system, as well as the path planner trajectory output system. Additionally, the planner system features a path selection and prioritization event that is used for re-planning.

The next outline demonstrates the methodology used to map an environment, construct a priority based path planning system and a hierarchical state machine that controls complex behaviors and interactions between all of the mapping and path planning systems.

4.1 Mapping System

The mapping system selected for this project was the octomap mapping software [12], which uses an octree data structure and a probabilistic model function to update voxel occupancy values. To create a map using octomap, a sensor capable of measuring range data is needed. For this study, a simulated depth camera with the parameters shown in table 4.1 was used for mapping.

Table 4.1: Simulated Depth Camera Specifications

Parameter	**Value**	**Unit**
Depth Range	6.0	$meters$
RGB Resolution	640x480	$pixels$
Depth Resolution	640x480	$points$
Field of View	1.02974	$meters$
Frame Rate	20	$\frac{frames}{second}$

The simulated depth camera generates a point cloud sensor message with the individual 3D world coordinates of each pixel. These set of measurements can then be used to construct an octomap using a mapping server. The recorded octomap can be stored into disk after enough of the map has been recorded, which can be later restored in a future mapping session.

4.1.1 Octomap

For the purposes of navigation, an octomap can give the required partition data to identify C_{free}. The octomap can be searched to find all elements in C_{free} that can be considered for path planning. However, not all voxels on an octomap have the same size, due to the structure of an octree. This presents a challenge from a graph construction standpoint, since it is difficult to index neighboring voxels. Nonetheless, the octomap software has a method to quickly and efficiently query voxels around a bounding box, which is useful to find and connect neighboring elements to a given voxel.

A manual simulated flight was performed to generate the octomap. Additionally, pre-computed waypoints were used to automatically fly and record environments using the octomap mapping server. However, the pre-computed waypoints were used only to map large structures. Figure 4.2 shows an octomap generated from a 3D model of the NFAC wind tunnel.

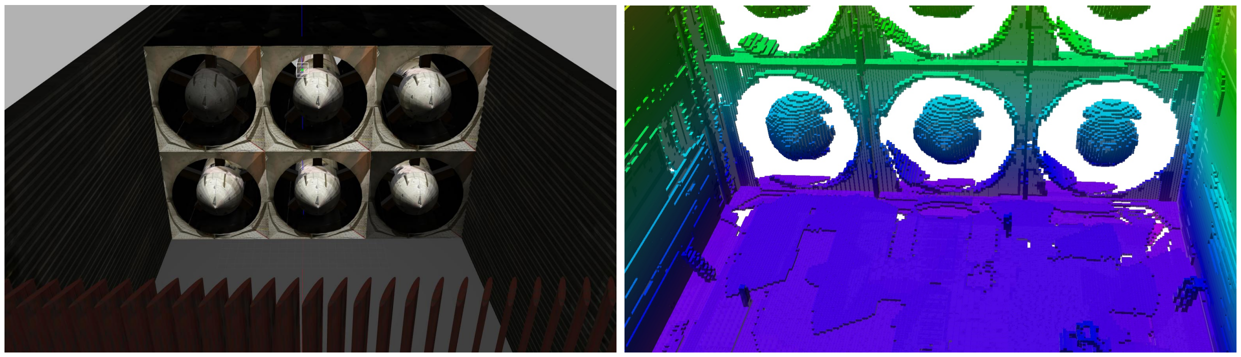

(a) NFAC 3D model (b) Octomap generated from NFAC 3D model

Figure 4.2: 3D model and octomap generated generated from a simulated environment

4.2 Path Planning

The octomap can be used to generate collision-free trajectories using a path planning algorithm. In this study, a two step path planning is proposed, by first constructing a a graph data structure using a cell decomposition algorithm, followed by the well-known A* graph search algorithm. Furthermore, the graph construction algorithm can connect the initial and target positions for generated paths from the current drone's position to a desired location.

4.2.1 Graph Construction

Constructing a graph data structure is the first step in most path planning algorithms. A graph data structure can be represented in two forms: Adjacency list or Adjacency Matrix. For this work, an adjacency list data structure was used to represent a graph.

The adjacency list was constructed by first considering the voxels in C_{free}, which make up the vertices of the graph. For the edges of the graph, a bounding box is created around every vertex. Using this box, voxels in C_{free} can be queried fast and efficiently. These voxels are then connected to vertex where the bounding box was applied. Figure 4.3 and 4.4 show the generated vertices and edges visually for the NFAC octomap.

However, the proposed method is impractical when considering every voxel on the map , due to the number of voxels that will have to be considered. For this reason, the graph construction algorithm considers voxels that are at a lower or specific size threshold. For an octomap, this requires a limit on the tree depth level, which results on limiting the size of voxels that can be considered. Moreover, the voxels used for path planning are only those found between the start and goal positions, by means of another larger bounding box. Finally, vertex neighbors are limited both in size and number. Doing this, ensures fine granularity

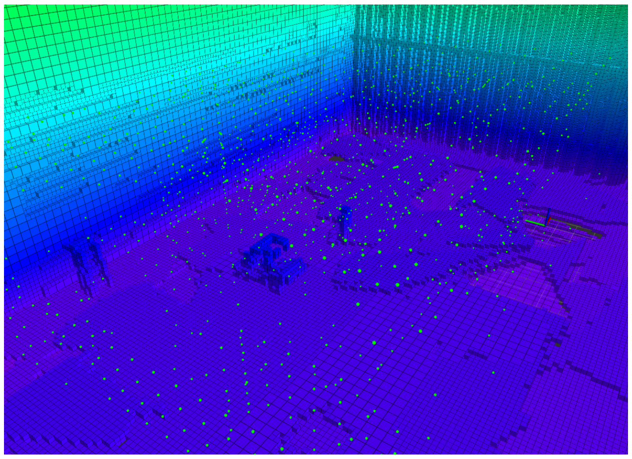


Figure 4.3: Graph vertices shown as green spheres

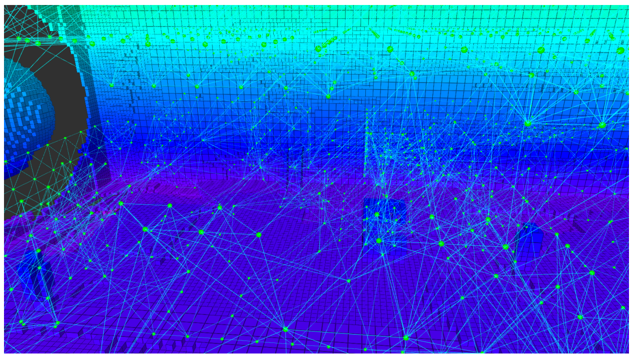


Figure 4.4: Graph vertices and edges represented with blue line

between vertices that avoid invalid edge connections that will go through obstacles.

4.2.2 Graph Search using A*

Once the graph is constructed, a path can be found through the use of a graph searching algorithm. In this work, an A* search algorithm was used to find a path between a start and goal vertices. The produced path then becomes a set of waypoints with both position and yaw rotation if desired. Figure 4.5 demonstrates the output path between the drone's current position and a given goal position.

The planner queues the waypoints to the drone, and subsequently waits for the drone to finish navigating the path. The planner then dequeues the next target position and computes a new path. This is repeated until the planner has found (or in some cases not found) paths for all goal positions.

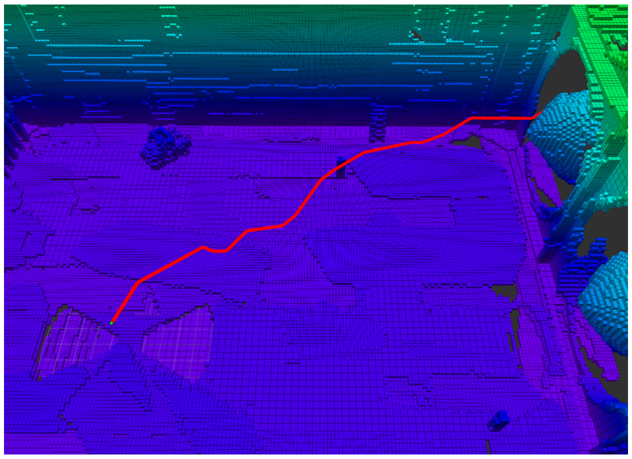


Figure 4.5: Path trajectory between start and goal vertices

4.2.3 Path Planning Priority

The proposed path planning method is capable of re-planning a path if an event occurs during the flight. If this event is considered of higher priority than the current path, a new path can be generated that preempts the previous path. This effect is immediate and causes the drone to navigate the new higher priority path. Once navigated, the path planner re-plans a new path using the previously preempted goal, which guarantees the navigation of an interrupted goal. This method allows the drone to re-plan a collision free path according to external events based on fault or damage detections.

To accomplish this behavior, a double ended queue data structure was used to append low-priority goals at the back of the queue, and prepend high-priority goals at the top of the queue. This behavior ensures that the planner generates paths for both an initial plan and reactive events. Figure 4.8 shows a graphical representation of the double ended queue data structure and how it prepends and appends high and low priority navigation goals.

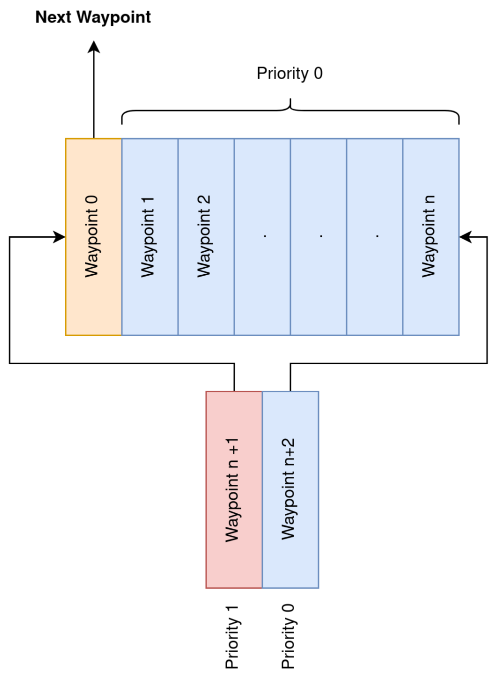


Figure 4.6: Path priority double ended queue

4.3 Drone Hierarchical State Machine

The proposed drone requires handling states based on external events such as QR a code or object detection. Furthermore, the drone has several internal states that must be handled in the right sequence. One such example are the transitions between positioning, orienting and performing an action, all of which must happen in the right order. Traditional FSMs can be used to represent this behavior programmatically.

However, as explained in section 1.5.1, for complex systems, a traditional state machine implementation becomes untractable and unmanageable. This is due to the number of states and event redundancies that are needed to go from a shared state to another. For instance, a landing event can be triggered at any time while the drone is flying. For a traditional state machine, events for every flying state will require a transition to a landing state. Therefore, for hierarchical state machines, states can be clustered into super states that will have one transition between another clustered state. This mechanism saves redundancies between states and transitions. This advantage can be clearly shown in figure 1.4 where the landing and flying states cluster other states that are components of a lower level drone functionality.

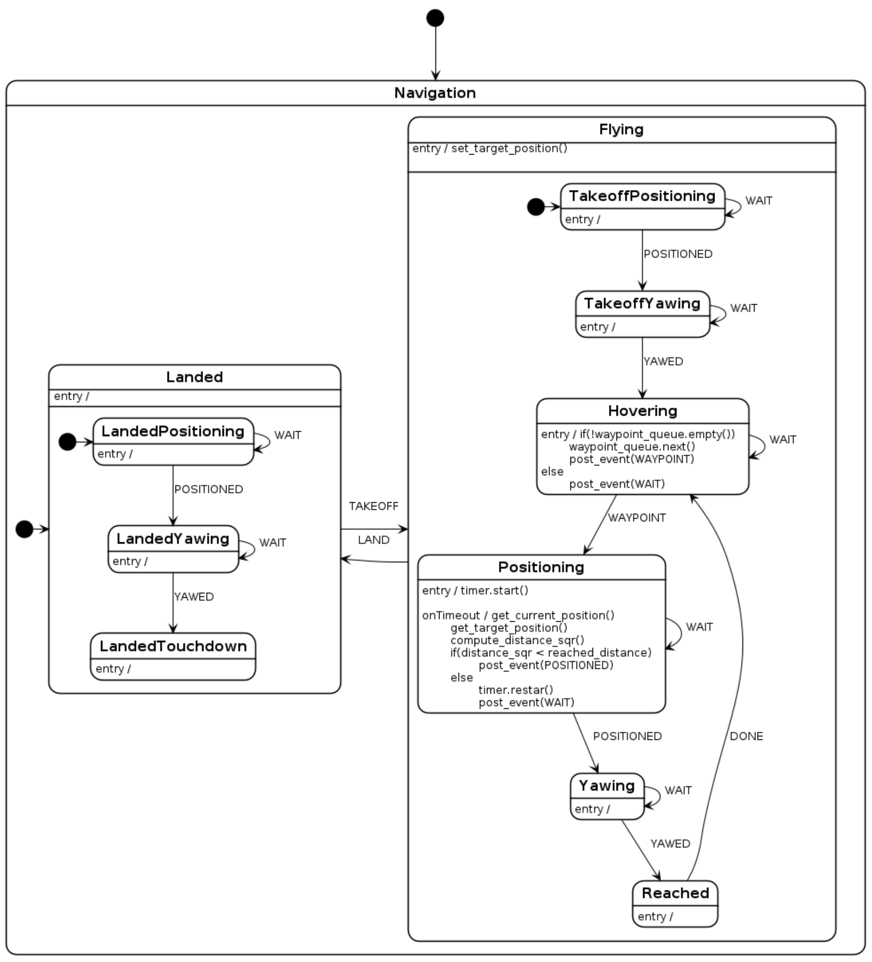


Figure 4.7: Proposed hierarchical state machine to control a drone

4.3.1 Drone Hierarchical States and Events

There are two super states on the presented state machine: **LANDED** and **FLYING**. These two super states are clustered together into a super state named **NAVIGATION**, which represents the main state machine for the drone. The **LANDED** super state contains several sub-states named **LANDED_POSITIONING**, **LANDED_YAWING** and **LANDED_TOUCHDOWN**. When the initial drone is commanded to land, the drone will go through these states sequentially until it lands on the ground. Similarly, for the **FLYING** super-state, an initial **TAKEOFF_POSITIONING** and **TAKEOFF_YAWING** states are triggered when the drone is commanded to takeoff. Once the takeoff events are completed, the drone goes into normal operation and waits for a waypoint at the **HOVERING** state. When a waypoint is received, the drone first goes to the **POSITIONING** state followed by the **YAWING** state, and finally performing an action at the **REACHED** state. The drone's state machine then goes back to the **HOVERING** state, where it dequeues the next waypoint and triggers the same sequence of events if there is a waypoint waiting on the queue.

4.3.2 Path Planning State Machine

The path planner expresses a complex system that needs to determine when a path has been navigated, and whether a higher priority goal has been received. To handle both tasks, a state machine as shown in figure 4.8 was designed. As with the drone state machine, a higher-state transition needed to express the path preemption behavior, which required two clustered states and a single event transition. Additionally, this state machine will compute paths sequentially as goals are queued.

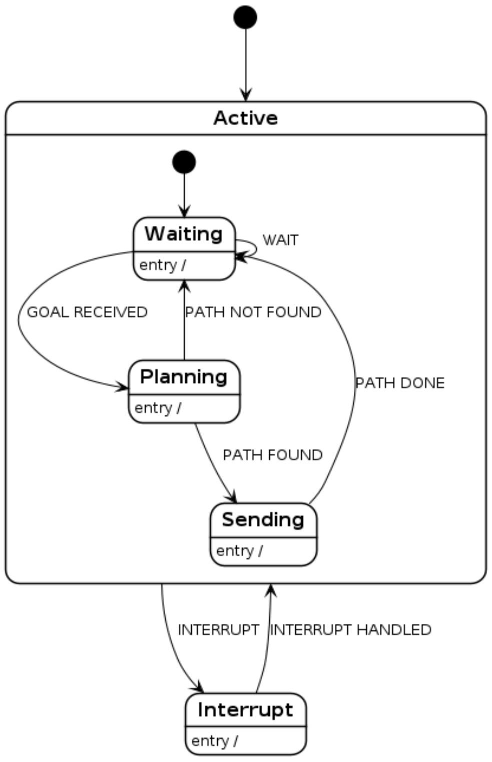


Figure 4.8: Path planning state machine to handle paths

4.3.3 Path Planning States and Events

There are 2 modes of operation for the proposed path planning system: **ACTIVE** and **INTERRUPT**. The **ACTIVE** state contains the states for the normal operation of the path planner, which consists of

WAITING, **PlANNING** and **SENDING**. The planner first waits for a new goal at the **WAITING** state. When a goal is queued, the planner transitions to the **PLANNING** state, where the graph is constructed and searched to find a trajectory. If a trajectory is not found, the planner goes back to the **WAITING** state, where it checks the queue again. Otherwise, the planner goes to the **SENDING** state, where the path is sent to the drone. The **SENDING** state then waits for the drone to finish traversing this path before going back to the **WAITING** state and repeating the same process.

The second mode of operation is the **INTERRUPT** state. In this state, the planner first sends a command to flush the waypoint queue of the drone, followed by a re-planning using the higher priority path. When the planner finishes navigating the higher priority path, it goes back to the **ACTIVE** state, which places the planner back into normal operation.

Chapter 5: Deep Learning and Markers for Intelligent Inspections

Deep learning has applications in image detection and segmentation, one of which is infrastructure detection damage. Neural networks are trained to detect, label and localize damages on a given image. On the field of machine learning, novel neural networks are proposed frequently. MobileNet neural networks [44] stand out as an attractive solution for online, real-time object detection for mobile platforms. Indeed, this allows mobile robot platforms to deploy object recognition neural networks that are fast, accurate and energy efficient. Additionally, the MobileNet networks have been implemented using the Single Shot MultiBox Detector (SSD) [45] neural network architecture, which further increases the accuracy and speed of detections.

However, deep learning algorithms are limited to detections and localization using the image pixel coordinates. As a result of this, image detection analysis is often done offline at the post-flight stage. This is problematic because the drone can not know where the detection happened with respect to a map frame. This limits the drone's ability to localize an object and generate a navigation plan for further inspection. In addition, critical inspection and navigation information is ignored, as the drone will not be able to react and navigate to a detection event. Evidently, detection events should be treated with high priority, and trigger additional inspection actions and events.

Therefore, this chapter presents a novel marker detection and localization system using deep learning neural networks and QR codes. The system uses a SSD MobileNet neural network and QR code software to detect, segment, transform and localize a detection box in a 3-dimensional map space. This system is capable of localizing and generating navigation waypoints (see section 6.2) by using the pixel coordinates and point-cloud data of a depth camera to extract and segment the 3-dimensional coordinates of an object or QR code. The segmented points are used to create a 3-dimensional bounding box that surrounds the detected object or QR code. The 3-dimensional coordinates are then transformed from the camera's frame to the map's inertial frame using a roll, pitch, yaw matrix transform. Using the transformed coordinate points, navigation waypoints can be created using the transformed 3-dimensional detection boxes. Lastly, a tracking system was implemented to uniquely identify detections and minimize false positive detections.

This chapter introduces the SSD MobileNet V1 neural network, how it was trained and deployed on a simulation environment. This is followed by several sections that go over the point-cloud coordinates, segmentation and transformation process to locate an object and a QR code with respect to an inertial frame. The chapter is concluded by the QR code and object tracking system.

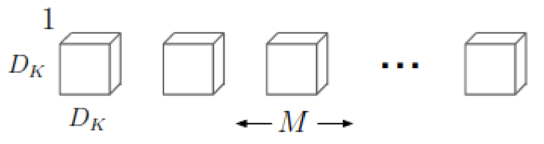
5.1 SSD MobileNet V1

MobileNet is a new class of neural networks that are fast and efficient on embedded systems. This family of networks use depthwise separable convolutional networks instead of traditional convolutional networks that are found on deep learning models. A depthwise separable convolutional network greatly reduces the number of weight parameters, which subsequently decreases the time and computational effort needed to train a network. Additionally, once deployed, the network model can infer input images at a higher speed and less computational effort than a traditional architecture with convolutional neural networks.

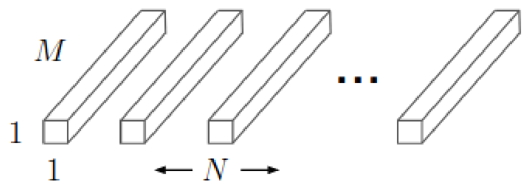
However, this architecture trades speed and efficiency at the cost of accuracy. To improve the accuracy, the MobileNet architecture was integrated into a SSD architecture that improved the accuracy of the neural network without significant latency penalties.

5.1.1 MobileNet

A MobileNet neural network is small and efficient architecture for real-time performance on mobile and embedded devices. The MobileNet architecture achieves this by using depthwise separable convolutional neural networks. In contrast to a traditional convolutional neural network, depthwise convolutional networks required fewer computations to filter and combine input datasets. Depthwise convolutional neural networks filter input channels separately, followed by a a 1x1 convolution (known as point-wise convolution) to combine the outputs of the depthwise convolutional layer. Figure 5.1 demonstrates the depthwise channel convolutional layers and the 1x1 point-wise full convolutional layer.



(a) Depthwise separable convolutional networks [44]



(b) Point-wise convolutional neural network [44]

Figure 5.1: Depthwise separable convolutional neural networks [44]

5.1.2 Single Shot MultiBox Detector (SSD)

Single Shot MultiBox Detector (SSD) is a deep learning neural network architecture that improves the accuracy of previous real-time detectors. SSD accomplishes this by eliminating bounding box proposals and feature resampling. Moreover, SSD proposes convolutional neural networks at different aspect ratios to predict object categories. As a result, SSD improves the accuracy of real-time detection systems, and as stated in the original article [45], the mAP increases from 64.3% to 74.3% for MobileNet networks. Figure 5.2 shows a diagram of a basic SSD neural network architecture using conventional convolutional neural networks of different dimensions.

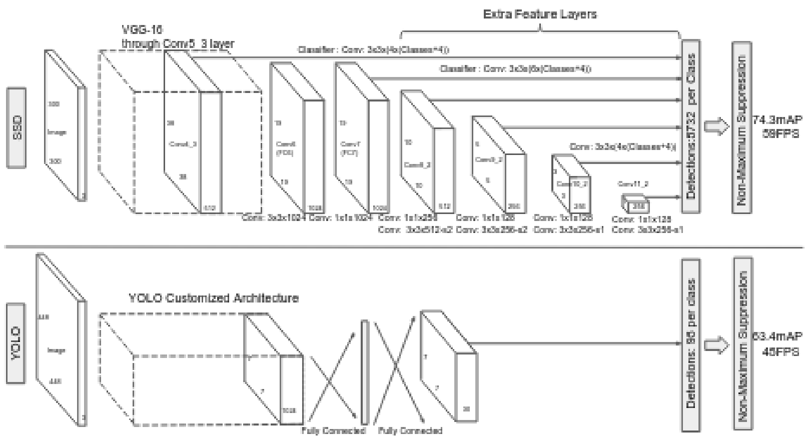

Extra Feature Layers
VGG-16 through Conv5_3 layer
SSD
Classifier : Conv: 3x3x(4x(Classes+4))
Classifier : Conv: 3x3x(6x(Classes+4))
Conv: 3x3x(4x(Classes+4))
Detections:8732 per Class
Non-Maximum Suppression
74.3mAP 59FPS
Conv: 3x3x1024
Conv: 1x1x1024
Conv: 1x1x256 Conv: 3x3x512-s2
Conv: 1x1x128 Conv: 3x3x256-s2
Conv: 1x1x128 Conv: 3x3x256-s1
Conv: 1x1x128 Conv: 3x3x256-s1
YOLO
YOLO Customized Architecture
Fully Connected
Fully Connected
Detections: 98 per class
Non-Maximum Suppression
63.4mAP 45FPS

Figure 5.2: SSD Network Architecture [45]

5.1.3 Training and Deployment

The SSD MobileNet V1 model was trained using Microsoft's Common Object in Context (COCO) dataset [46], which contains photos of 91 objects that can be found frequently in a variety of contexts. According to [44] a trained SSD 300 framework with the MobileNet model architecture is able to reach a mAP of 19.3%. At the same time, the number of operations was greatly reduced for the SSD 300 network model.

For this study, a pre-trained tensorflow SSD MobileNet neural network was used to infer object detections from a simulated RGB camera. To load and run the model, the Deep Learning OpenCV framework was used to infer bounding boxes around possible detected objects. The object class and probability is also provided. Figure 5.3 shows the image detection output using OpenCV and the pre-trained SSD MobileNet model.

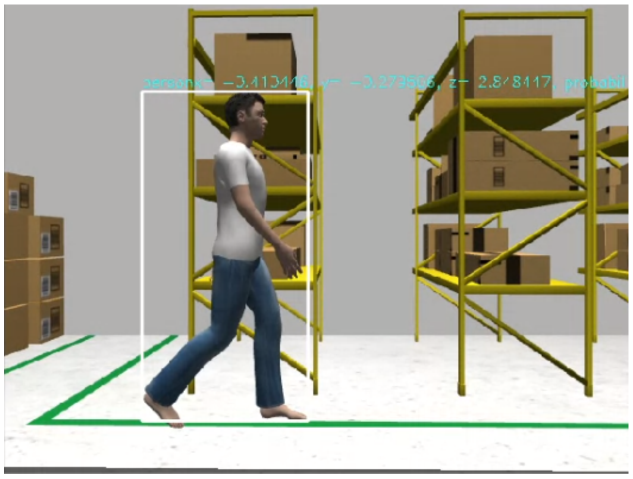


Figure 5.3: Object detection using the Deep Learning OpenCV framework and a pre-trained SSD MobileNet model

5.2 QR code detection and Localization

QR code detection is a reliable, ubiquitous and fast method to extract information from an image. Therefore, an alternative approach using QR code detection was developed to create automated inspection markers. The markers will serve as a way to direct the drone to a section of interest, where some actions must be performed. To detect QR codes, the zbar software library[1] was used [47]. The same camera used in section 5.1.3 was used to capture RGB images that were later converted to a grayscale image for QR code detection. Similarly to section 5.3, the QR detection pixel coordinates were transformed to 3-dimensional coordinates using the same point-cloud indexing process shown in section 5.3.1. With the 3-dimensional coordinates, the center point of the detection box can be found using the following equations:

$$X = \frac{1}{n}\sum_{k=0}^{n} x_i \tag{5.1}$$

$$Y = \frac{1}{n}\sum_{k=0}^{n} y_i \tag{5.2}$$

Where X and Y are the center coordinates of the detection box, x_i and y_i are the detection coordinates, and n is the number of detection coordinates. Finally, the QR code type and data can be extracted and displayed on an image. Figure 5.4 shows the QR code detection displayed on the original RGB input image.

[1]zbar library can be found at this link `https://zbar.sourceforge.net/`

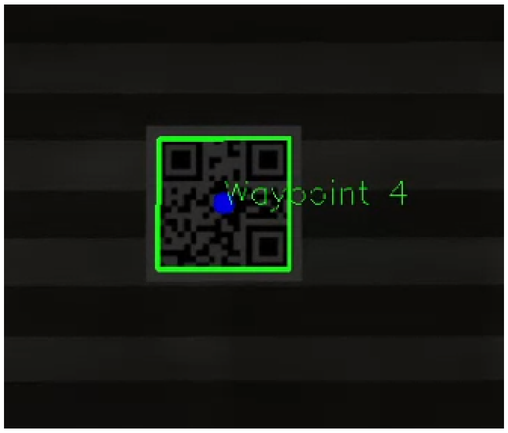


Figure 5.4: QR code detection with data displayed directly on the center of the code

5.3 Object Localization

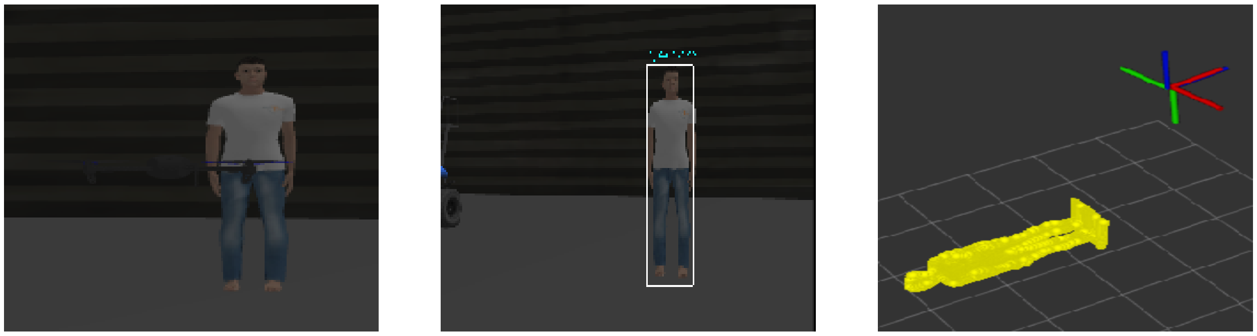
The object and QR detection systems output bounding boxes with pixel point coordinates on a conventional $X - Y$ image frame. This is not particularly useful for navigation, since the 3-dimensional world coordinates of the object cannot be extracted directly from the image detection output. Therefore, the following sections present a novel method to transform pixel detection coordinates, to 3-dimensional coordinates that can be used for localizing an object with respect to an inertial frame. The proposed method segments a point-cloud measurement from the box detection pixel coordinates and image width parameter (see section 5.3.1 for more details). The farthest and closest points are scanned on the segmented point-cloud. These two points are then used to construct a 3-dimensional bounding box, which subsequently is used to estimate the object's location on the camera's frame. Lastly, the estimated coordinates are transformed from the camera's frame to the map's frame of reference, giving the final coordinates to localize an object for navigation. The final estimated coordinates can then be used to create a navigation waypoint.

5.3.1 Object Point Cloud Segmentation

Point-clouds contains 3-dimensional range data from the camera's position to obstacles found in an environment. Point-cloud data is often used for mapping or collision avoidance tasks. Nevertheless, in this work, point-clouds and pixel detection coordinates are used to extract and segment portions of a larger point-cloud measurements. Using the image width parameter and the pixel coordinates, an index is computed to find the corresponding 3-dimensional point coordinate located inside a list of a larger point-cloud measurement. Equation 5.3 demonstrates the equation used to compute a point-cloud index k given the i and j image coordinates, and the image width w.

$$k = i + j * w \tag{5.3}$$

After finding the 3-dimensional point coordinates, a segmentation process is performed which consist of storing the points inside the detection coordinates into a separate and smaller point-cloud. This subset point-cloud represents the points that make up the detected object, and thus, can be used to locate an object in $\mathbb{R}^3$. Figure 5.5 shows the output of a point-cloud segmentation process using the found 3-dimensional point detection coordinates. Notice that the segmented point-cloud is located with respect to the camera's frame of reference, section 5.3.3 explains in detail how to transform and translate the detection 3-dimensional coordinates from the camera's to the map's frame.


(a) Drone facing the detected object as seen from the map

(b) Camera detection output image

(c) Point cloud segmentation represented as yellow dots

Figure 5.5: Object detection and point cloud segmentation process

5.3.2 Object Bounding Box from Point Cloud Segmentation

Although the segmented cloud gives 3-dimensional points that make up an object, it is still unclear which of the points should be used to estimate the location of the object. Additionally, the depth information about the object is now known, but it is not included in the original detection coordinates used on section 5.3.1. Hence, a 3-dimensional bounding box was created by first finding the farthest and closest points on the segmented point-cloud, and then using these two points to create bounding box as shown in figure 5.6.

The center point can be found by using the maximum and minimum point coordinate values as shown in equations 5.4-5.6. The computed coordinates are assumed to be the object's position in the camera's frame.

$$x = \frac{x_{max} + x_{min}}{2} \tag{5.4}$$

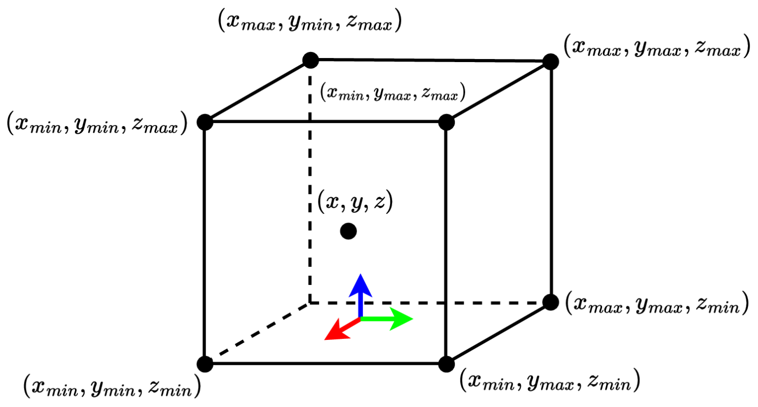

$(x_{max}, y_{min}, z_{max})$
$(x_{max}, y_{max}, z_{max})$
$(x_{min}, y_{max}, z_{max})$
$(x_{min}, y_{min}, z_{max})$
(x, y, z)
$(x_{max}, y_{max}, z_{min})$
$(x_{min}, y_{min}, z_{min})$
$(x_{min}, y_{max}, z_{min})$

Figure 5.6: Bounding box for object recognition with coordinates used to localize object (minimum is closest and maximum is farthest)

$$y = \frac{y_{max} + y_{min}}{2} \tag{5.5}$$

$$z = \frac{z_{max} + z_{min}}{2} \tag{5.6}$$

Ultimately, this process maps 2-dimensional detection pixel image coordinates, to 3-dimensional coordinates for a bounding box and its center point, that uniquely identifies a detection event. Consequently, the bounding box can be used to generate navigation waypoints useful for reactive inspection events (see section 6.2).

5.3.3 Object and QR code frame transformation

In sections 5.2 and 5.3.2 3-dimensional coordinates were extracted using point-cloud indexing and segmentation. However, these coordinates are with respect to the camera's frame of reference, and hence, cannot be used to locate an object or QR code on a map. For that reason, a matrix transformation followed by a vector translation needs to be performed between the camera's frame and the map's inertial frame. Moreover, the camera is attached the drone's frame, so evidently, an intermediate transformation between the camera's frame and the drone's frame has to be performed first. Figure 5.7 demonstrates the relationship between the drone, camera and map frames.

The camera's position and orientation with respect to the drone's frame of reference are:

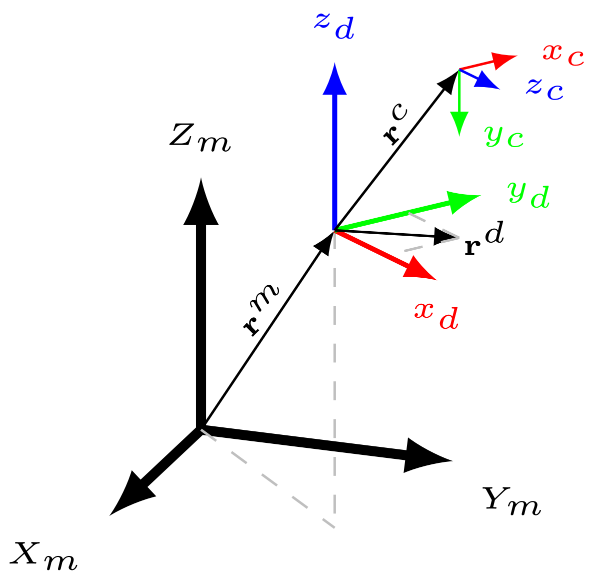


Figure 5.7: Map, drone and camera frames

$$\mathbf{r}^c = \begin{bmatrix} 0.1 \\ 0.0 \\ 0.0 \end{bmatrix} \tag{5.7}$$

$$\theta^c = \begin{bmatrix} -\pi \\ 0.0 \\ \pi \end{bmatrix} \tag{5.8}$$

Where $\mathbf{r}^c$ is the translation vector and θ^c the orientation vector with pitch, roll and yaw values. Vector values in 5.7 and 5.8 show that this is a static transform that requires the pitch and roll angles to be rotated $-\pi$ and π radians respectively. Furthermore, the frame must be translated -0.1 meters about the x-axis of the drone frame of reference. Finally, a transform between the drone frame d and the camera frame c must be performed. The equation that relates a vector in the camera frame $\mathbf{p}^c$ to a vector in the drone frame $\mathbf{p}^d$ is

$$\mathbf{p}^d = \mathbf{R}_c^d \mathbf{p}^c + \mathbf{r}^d \tag{5.9}$$

Where $\mathbf{R}_c^d$ and $\mathbf{r}^d$ is the transform matrix and translation vector between the c and d frames. Similarly a vector in the map's frame $\mathbf{p}^m$ can be related to a vector in the drone's frame $\mathbf{p}^d$ using 5.10

$$\mathbf{p}^m = \mathbf{R}_d^m \mathbf{p}^d + \mathbf{r}^m \tag{5.10}$$

Where $\mathbf{R}_d^m$ and $\mathbf{r}^m$ is the transform matrix and translation vector between the d and m frames.Using equation 5.9 and 5.10, an equation to transform and translate vectors in $\mathbf{p}^c$ and $\mathbf{p}^m$ can be found,

$$\mathbf{p}^m = \mathbf{R}_c^m \mathbf{p}^c + \mathbf{R}_d^m \mathbf{r}^d + \mathbf{r}^m \tag{5.11}$$

$$\mathbf{R}_c^m = \mathbf{R}_d^m \mathbf{R}_c^d \tag{5.12}$$

Lastly, the transform $\mathbf{R}_c^m$ and vector translations $\mathbf{p}^d$ and $\mathbf{p}^c$ relate a vector from the camera frame to the map frame. It should be noted that equation 5.12 uses the ZYX euler convention and it is represented as,

$$\mathbf{R} = \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\gamma) & -sin(\gamma) \\ 0 & sin(\gamma) & cos(\gamma) \end{bmatrix} \tag{5.13}$$

5.3.4 Object and QR code detection tracking

Using the methodology of the previous sections, the proposed system can localize QR codes or objects on a map. However, the detections do not have unique identifiers that will avoid the storage of multiple similar detections. Moreover, the detection system does not have a way to deal with false positive events, as it will consider detections that appear in one frame instead of multiple frames. For those reasons, a simple tracking algorithm based on [48] was implemented to create unique identifiers based on new and previous detection events that will prevent the tracking of duplicate and false positive detections. The tracking system starts by computing a cost matrix using the euclidean distance from current and previous detections known as tracks. The assignment problem is solved using the hungarian algorithm [2] and the previously computed cost matrix. Lastly, the creation of new trackers is determined by the number of frames it has been detected and the track quality determined by the euclidean distance of new detections. The following sections detail the process to track and update detections.

5.3.4.1 Cost matrix

The cost matrix was computed using the euclidean distance between current and previous detections. Matrix 5.14 demonstrates the basic structure of the cost matrix using tracked and current detections. The tracked

[2]Software library used can be found at `https://github.com/mcximing/hungarian-algorithm-cpp`

detections correspond to the columns of the matrix as $t_0, t_1, \ldots, t_n$, with the current detections $d_0, d_1, \ldots, d_n$ shown as the rows in the matrix. The index i, j corresponds to the euclidean distance $r_{i,j}$ between track i and detection j.

$$\mathbf{C} = \begin{array}{c c} \begin{array}{ccccc} t_0 & t_1 & t_2 & \ldots & t_n \end{array} & \\ \begin{pmatrix} r_{0,0} & r_{0,1} & r_{0,2} & \ldots & r_{0,n} \\ r_{1,0} & r_{1,1} & r_{1,2} & \ldots & r_{1,n} \\ r_{2,0} & r_{2,1} & r_{2,2} & \ldots & r_{2,n} \\ \vdots & \vdots & \vdots & \ddots & r_{0,n} \\ r_{n,0} & r_{n,1} & r_{n,2} & \ldots & r_{n,n} \end{pmatrix} & \begin{array}{c} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_n \end{array} \end{array} \tag{5.14}$$

5.3.4.2 Detection to track assignment

The objective of the detection to track assignment is to minimize the cost matrix element sum. For instance, in the case of matrix 5.14, every detection row d_i must choose the lowest value in all tracked columns t_j. Moreover, two different detections cannot be assigned to the same track, as this will not create new tracks. A hungarian algorithm implementation was used to solve the detection to track assignment problem. Once solved, the algorithm will output a list of index assignments i, j such that element sum of the matrix $\mathbf{C}$ is minimized. Detections that were not assigned to any tracked detections will have an index value of -1.

$$\mathbf{C} = \begin{array}{c c} \begin{array}{cccc} t_0 & t_1 & t_2 & t_3 \end{array} & \\ \begin{pmatrix} \underline{0.12} & 1.30 & 3.58 & 10.3 \\ 20.45 & \underline{1.20} & 30.4 & 20.4 \\ 1.02 & 3.2 & 0.45 & \underline{0.02} \\ 1.02 & 10.42 & \underline{0.70} & 2.34 \end{pmatrix} & \begin{array}{c} d_0 \\ d_1 \\ d_2 \\ d_3 \end{array} \end{array} \tag{5.15}$$

Matrix 5.15 demonstrates an example cost matrix with the computed assignments underlined. The assignments are $(0, 0), (1, 1)(2, 3), (3, 2)$ and the total cost of this assignment is $c = 2.04$.

5.3.4.3 Creation and deletion of detections

Detections that were not assigned to tracks are candidates for the creation of a new track. However, potential new tracks go under a supervisory period to avoid the indiscriminate addition of false positive detections. The supervisory period consists of counting the number of frames in which the detection has appeared. If the count exceeds a threshold, a track is created and assigned a unique ID. Otherwise, if the potential track has not been observed for a number of frames, it gets deleted form the list.

Additionally, poor detection to track assignments are ignored, as this will cause inadequate updates to the current tracks. A poor detection is characterized by the euclidean distance between a detection and a track. Detections that are beyond a threshold value $r_{i,j} > \gamma$ are considered to be low quality, and should not be used to update current tracks. With this method, track noise is reduced and will help to maintain good tracking updates for the position and labels of the current tracks.

Chapter 6: Generation of Waypoints and Actions for Drone Inspection

An autonomous inspection drone requires navigation waypoints to perform path planning and collect suitable inspection data. Most inspection systems use coverage path planning [49] to generate waypoints on an environment. Coverage path planning attempts to cover an entire surface or volume geometry with a trajectory pattern. With this pattern, waypoints can be generated for a path planning system. During the navigation phase, the drone can perform automated actions that will aid in the process of inspection.

Nevertheless, coverage path planning approaches are energy and time inefficient. The drone will spend most of its flight time navigating towards areas that are empty or of little value from an inspection stand point. Consequently, more flights are required to guarantee the detection of damage and defects. Moreover, coverage path planning uses primitive geometries to generate inspection trajectories without the use of a map. Evidently, most buildings poses sections with complex geometries and structures that will not be inspected using coverage path planning algorithms. Ultimately, this results in low quality inspections that do not yield meaningful data for the detection and characterization of damages and defects.

For those reasons, this chapter presents a novel method to generate waypoints using an octomap and detection bounding boxes from object recognition and QR codes. For an octomap, voxel surface normals are used to generate waypoints by extruding the normal vector by a parameter scalar value. Similarly, bounding box surfaces from objects and QR codes are used to first compute vector normals from the box surfaces, and then extrude these normal vectors in the same way as the octomap voxel normals. Additionally, waypoints embed actions that are triggered once they been navigated. Thus, a waypoint contains information about the location of an area of interest, and the action that must be performed once the location has been reached.

The next sections explain in detail waypoint generation for a map, object detection and QR codes. Finally, a section is dedicated to detail the inspection actions that can be performed by the drone.

6.1 Waypoint Generation from Octomaps

An octomap was used in section 4.2 to generate collision free trajectories by scanning unoccupied voxels. Similarly, an octomap can be used to scan occupied space. Although counterintuitive at first, occupied space contains crucial data for an inspection. Occupied voxels determine the areas where building sections or structures can be found. Moreover, a voxel can supply information about the direction in which the drone must be facing to capture data. For instance, occupied voxels on a mapped surface can yield information about the position and direction of an area of interest through its surface vector normals. The vector normals can be extruded to an offset distance from the voxels by simple scalar multiplication. For instance, consider the following voxel with a vector position $\overrightarrow{p}$ and a surface normal vector $\overrightarrow{n}$ aligned with the x-axis of the map,

$$\overrightarrow{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{6.1}$$

$$\overrightarrow{n} = \begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix} \tag{6.2}$$

Using the vector $\overrightarrow{p}$, $\overrightarrow{n}$ and a scalar parameter β, a new vector $\overrightarrow{w}$ can be constructed. The vector $\overrightarrow{w}$ contains the location to which the drone must navigate to capture inspection data. The vector $\overrightarrow{w}$ is represented by the following equation,

$$\overrightarrow{w} = \overrightarrow{p} + \beta\overrightarrow{n} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \beta \begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix} = \begin{bmatrix} x + \beta \\ y \\ x \end{bmatrix} \tag{6.3}$$

Additionally, the drone must be facing directly opposite to the normal vector. Hence the yaw angle for the drone can be computed using the following equation:

$$\theta = \cos^{-1}(\overrightarrow{n} \cdot -\overrightarrow{n}) \tag{6.4}$$

Figure 6.1 shows the generated waypoints on an octomap using the voxels position, surface normal vectors and a scalar parameter as shown in equation

However, as it was the case for the graph construction in section 4.2.1, considering every occupied voxel will be impractical for medium and large maps. Therefore, number of scanned occupied voxels for waypoint generation is down-sampled to a specified user parameter value.

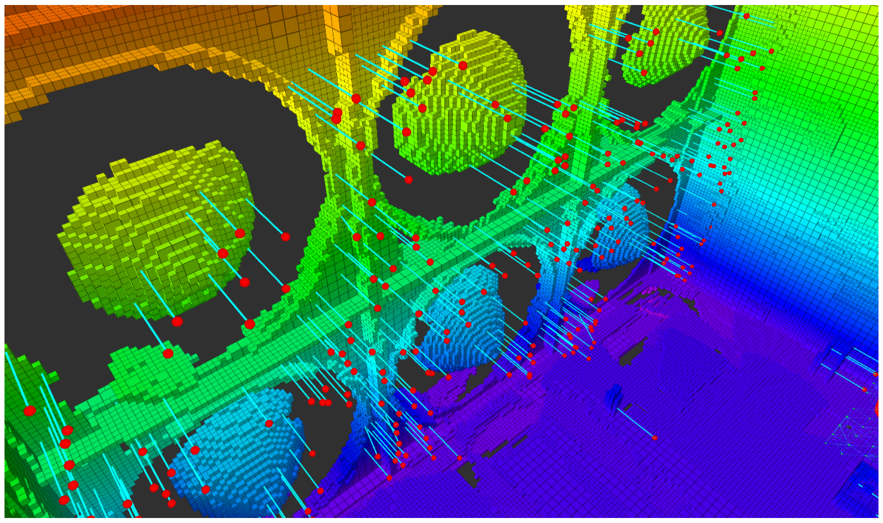


Figure 6.1: Waypoints generated from voxel surface normals using the octomap software

6.1.1 Viewport

The occupied voxels on an octomap contain face surface normals that can be used to generate waypoints as shown in section 6.1. However, there is still ambiguity as to which surface normal should be used. For instance, a voxel contains six surfaces, each with its own normal vector. In some cases, considering all six normal vectors will generate waypoints that are impossible or impractical to reach for a drone. Furthermore, for the case of an indoor inspection, vector normals must face to the inside of the building. Evidently, a sense of direction must be given to test the direction of the normal vectors. This test selection process can be performed mathematically through the use of a viewport vector $\overrightarrow{v_{view}}$ and the following equation,

$$\alpha = \overrightarrow{n} \cdot \overrightarrow{v_{view}} \tag{6.5}$$

Where $\overrightarrow{n}$ is the normal vector under consideration and α is the result of the dot product operation that can be interpreted as how "equal" both vectors are. If the dot product of the two vectors is zero, then both vectors are orthogonal to each other, which means that normal vector under consideration is not facing the viewport vector. Therefore, this normal vector must not be considered to construct a waypoint. Likewise, for a result of $\alpha = -1$, the normal vector under consideration is pointing opposite to viewport vector, and evidently, should not be considered. These results can be summarized in equation 6.6

$$y = \begin{cases} true, & \text{if} \quad \alpha = \vec{n} \cdot \overrightarrow{v_{view}} \geq \gamma \\ false, & \text{if} \quad \alpha = \vec{n} \cdot \overrightarrow{v_{view}} < \gamma \end{cases} \tag{6.6}$$

Were γ is a parameter threshold used to determine how "equal" both vectors should be if they want to be considered for a waypoint. For $y = true$ a waypoint must be generated from this surface normal, otherwise if $y = false$ this surface normal should be ignored.

6.2 Waypoints from object detection bounding box

While map waypoints are useful, there are instances where detected objects or areas require immediate attention. Sections 5.3.2 and 5.3.4 discussed the detection and tracking of objects and QR codes through the use of bounding boxes. However, the methods discussed in these sections estimate the 3-dimensional coordinates at the center of the detected object. Generating a waypoint from these coordinates will certainly place the drone at an unreasonable and dangerous position to the object. For that reason, the bounding box construction method shown in section 5.3.2 is used to compute normal vectors along the surfaces of the box. For instance, using one face of the bounding box, vectors $\overrightarrow{AB}$ and $\overrightarrow{AC}$ can be constructed from the vertices of this face. Figure 6.2 shows the face of a box with vectors $\overrightarrow{AB}$ and $\overrightarrow{AC}$

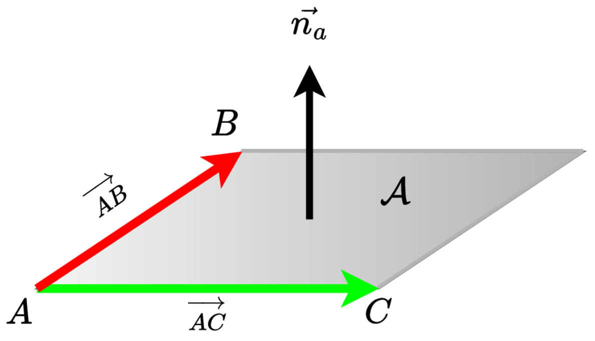


Figure 6.2: Surface normal from on of the faces of the detection box

Using these vectors a normal vector $\vec{n_a}$ on surface $\mathcal{A}$ can be found using the following equations,

$$\vec{v_1} = \overrightarrow{AB} \tag{6.7}$$

$$\vec{v_2} = \overrightarrow{AC} \tag{6.8}$$

$$\vec{n_a} = \vec{v_1} \times \vec{v_2} \tag{6.9}$$

Similar to section 6.1, the computed normal vectors can be used to orient and position the drone using equations 6.3 and 6.4, which will ensure that the drone will be at a safe distance from the detected object or area. Figure 6.4 demonstrates the computed surface normals on a constructed bounding box detection. Observe that the distance from the object can be controlled by the scalar parameter β as discussed in section 6.1

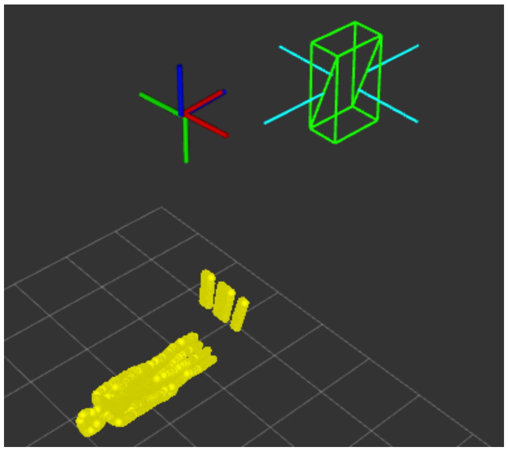


Figure 6.3: Detected object bounding box normal vectors used to inspection waypoints

6.3 Waypoints from QR code detection surface

The QR detection system discussed in section 5.2 can also be used to generate navigation waypoints. In contrast to an object, the QR detection output provides only a surface detection instead of a bounding box. This fact reduces the number of normal vectors to one. Using equations 6.7, 6.8 and 6.9 a normal vector to the detection surface can be computed. Similarly, the scalar parameter β controls the distance of the waypoint from the QR detection surface. Figure 6.4 demonstrates a normal vector computed from a QR detection surface.

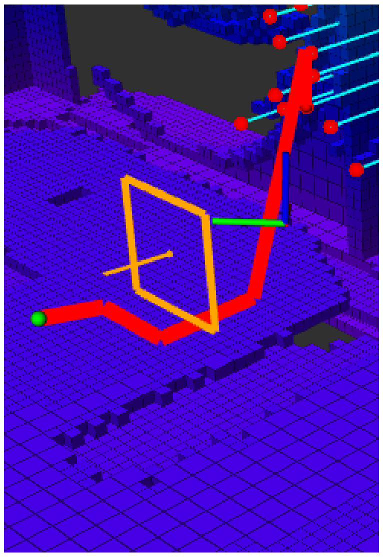


Figure 6.4: Detected QR code with computed surface normal, which is subsequently used to create a navigation waypoint

6.4 Inspection Actions

An automated inspection drone must be able to perform actions suitable to detect and characterize defects and damaged areas or objects. Section 2 showed that most drone inspection systems use imaging sensors to capture photos of an area at pre-determined positions. However, these actions are determined at the pre-flight stages and are not subject to change once the drone is performing a mission. Moreover, recording images is the only action available to these drones, which limits the ability to detect defects on an inspection mission by using only one type of sensor data.

Therefore, for the proposed drone inspection system, three distinctly actions are available: image recording, video recording and point cloud recording. These actions make use of the depth camera capabilities shown in section 4.1. Additionally, the action type is directly embedded on the waypoint, so when reached, the action takes place at the location and orientation of the waypoint. This means that different actions can be triggered at different times and locations, which greatly increases the volume and data sources relevant for an inspection. Section 7.5 demonstrates the recorded point-cloud and images from an inspection mission.

Chapter 7: Results and Case Studies

This chapter presents the results and case study for the inspection of the NFAC located at the NASA AMES research facility. The NFAC is the largest wind tunnel facility in the world [50]. This building was selected as part of a joint effort with the Fluid Mechanics Lab (FML) to automate the inspection of the NFAC wind tunnel. The NFAC wind tunnel presents a unique inspection challenge because of the sheer scale of the building. Additionally, the NFAC has remote and hard to reach areas that will be difficult to inspect manually. For these reasons, a drone platform was selected to automate the inspection of the NFAC. The NFAC was divided into two sections due to practical constraints on mesh model sizes for simulations.

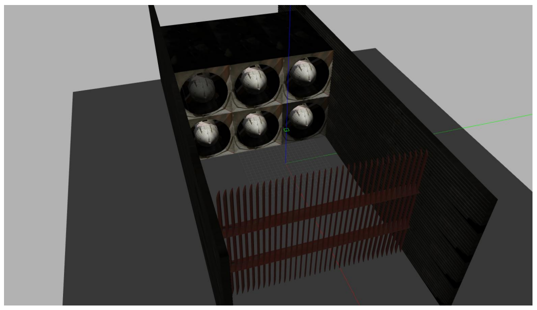
To perform simulations on the NFAC, a mesh model was constructed using approximated dimensions. This model was then placed into the simulation environnement discussed in chapter 3. With the mesh model in place, an octomap is created manually by flying the drone using a remote controller. Following the octomap recording, path planning and waypoint generation can be performed using the methods discussed in 4.2 and 6.1 respectively. Finally, object and QR detection can be used to generate tracked detection bounding boxes and surfaces waypoint generation and inspection as presented in section 6.2 and 5.2. The waypoints determine the actions that must be taken at the end of navigation trajectory. Images, video recordings and point-cloud data are produced at the end of the inspection.

The following sections present the results for the systems and methods shown in chapters 4- 6. Environment models and mapping, path planning, and object/QR detection results are presented in the first sections. Lastly, a study case inspection mission of the NFAC is presented on the last section. This acts as a demonstration of the proposed methods and systems and its utility and innovation for automated drone inspections.

7.1 Environment Models and Mapping

A mesh model was created to setup a simulation environment. Using the mesh model, an octomap was created by manually flying around the mesh model. This resulted on a voxel map with an underlying octree data structure. Figures 7.1 and 7.2 demonstrate the mesh models and maps generated by the octomap software framework.

The octomaps were characterized by their memory size, number of free and occupied voxels, and the scan latency of occupied and unoccupied voxels. Both are important metrics for online drone inspection mapping and path planning. Table 7.1 demonstrates the results for the two sections of the NFAC facility.


(a) NFAC section 0

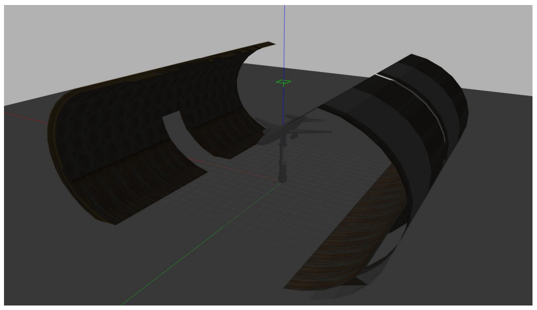
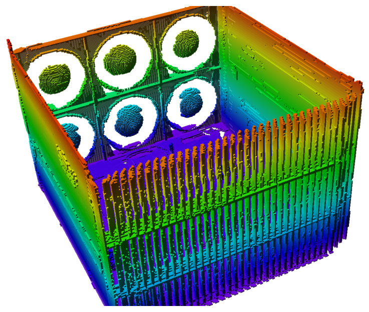

(b) NFAC section 1

Figure 7.1: Mesh models of simulated environments


(a) NFAC section 0 octomap

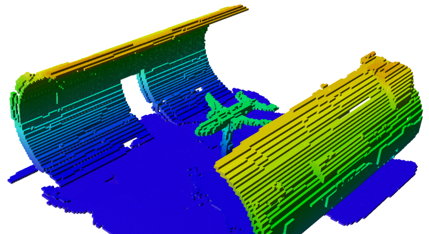

(b) NFAC section 1 octomap

Figure 7.2: Recorded octomaps for mesh models

Table 7.1: Octomap Results

Octomap	Memory size	Free Voxels	Occupied Voxels	Scan Latency
NFAC section 0	292.6kB	211272	483094	364 ms
NFAC section 1	77.1kB	115084	49966	77 ms

7.2 Path Planning Results

The path planning system must find a valid collision-free trajectory within a practical amount of time. To accomplish low latency, the path planner must limit the number of vertices and edges that are considered to construct a graph, regardless of the distance between the start and goal positions. Consequently, the graph search algorithm will require fewer node graph searches, which ultimately results in fast path generations.

To test the graph size, and path trajectory computation time[1], 100 path finding searches were performed using random start and goal positions. Figure 7.3 shows the edge and vertex count plot in relation to the distance of the start and end positions. Similarly, figure 7.4 shows the computational time to construct and search a trajectory path. Form the figures it can be observed that the the number of vertices and edges is bounded, which subsequently bounds the computational time to construct and search a graph. Moreover, the computational time mostly remains mostly constant at the sample average of *294 ms*. This demonstrates that fast path trajectory computations are possible even for large maps using the method in section 4.2. Similarly, for graph construction, the vertex and edge count remains close to the sample average of *502* and *2043* respectively.

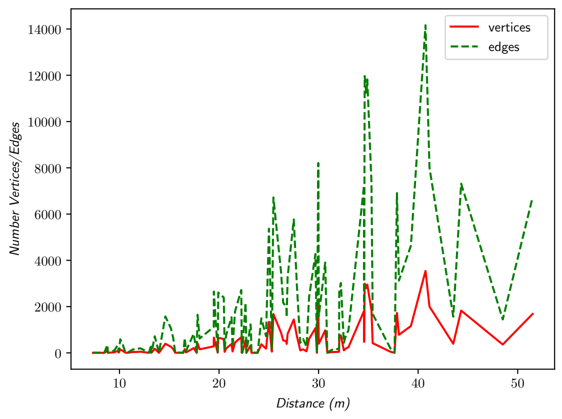


Figure 7.3: Graph vertices and edges count as a function of distance between start and goal positions

[1]All tests were performed using an 11th Gen Intel Core i7-1165G7 2.8GHz CPU, with 23.3 GiB of memory

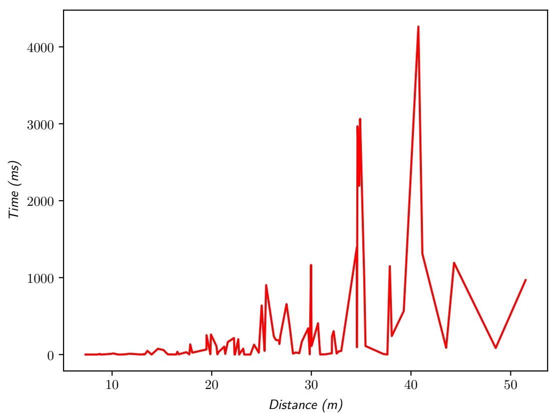


Figure 7.4: Graph build time as a function of distance between start and goal positions

7.3 Object and QR Detection and tracking Results

The QR and object detection system was proposed as a way of generating waypoints for navigation and inspection. Evidently, the quality of the inspection data is tightly coupled to the accuracy of the detection system. Therefore, to test the accuracy and precision of the QR and object detection system, an experiment was performed on the NFAC simulation. The experiment consisted of navigating, detecting and recording the coordinates of four QR codes with unique data and positions. Using the recorded coordinates of the detections, figure 7.5 was plotted.

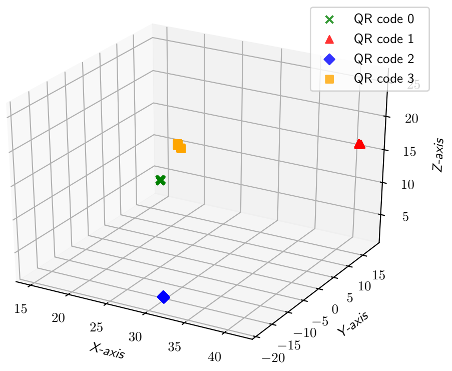


Figure 7.5: NFAC QR code detection positions

Figure 7.6 shows the coordinate locations for detections on QR code 0. From the image it can be observed that the detections were highly concentrated around the mean center point $\bar{\mathbf{r}} = [15.039 \quad 18.14 \quad 3.40]^\top$. Compared to the true value position of $\mathbf{r} = [15.1 \quad 18.0 \quad 4.1]^\top$, this represents a percent error of $\mathbf{r}_{error} = [0.26\% \quad 0.22\% \quad 6.65\%]^\top$. The average deviations from the mean represent the precision of the detection measurements and were recorded to be $\mathbf{r}_{dev} = [\pm 0.029 \quad \pm 0.008 \quad \pm 0.068]^\top$

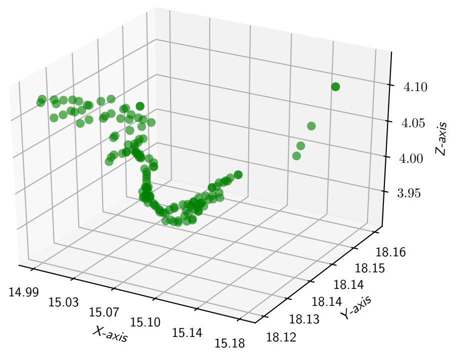


Figure 7.6: Detections for QR code 0

Lastly, figure 7.7 show the approximated probability mass functions for the measurements on all 3 axis. The figure demonstrates measurements concentrated around the mean with low variance, which exemplifies the high accuracy and precision of the detection and tracking system.

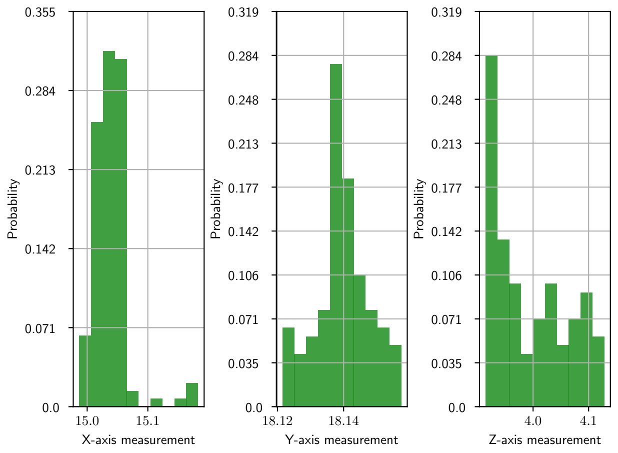


Figure 7.7: QR code 0 detection approximated probability density mass functions for x, y, z values

7.4 Waypoint Generation Results

Waypoints can be generated using an octomap using the method described in section 6.1. A uniformly distributed set of waypoints is a desired outcome of this method, as this will likely increase the inspection coverage on an selected area. Therefore, to test waypoint distribution, generated waypoints for the NFAC mesh model were plotted in figure 7.8 to observe the distribution of the $x - z$ and $y - z$ planes of the walls. Moreover, the 2D histograms for the waypoint distribution of the walls were plotted on figure 7.9. The 2D histogram demonstrates the waypoint distribution count of 10x10 equally spaced plane regions. From both figures, it can be observed that that the distribution of waypoints varies on different surfaces. This might be due to a number of factors such as low voxel density, a high down-sample factor, inadequate map segmentation and incorrect viewport direction.

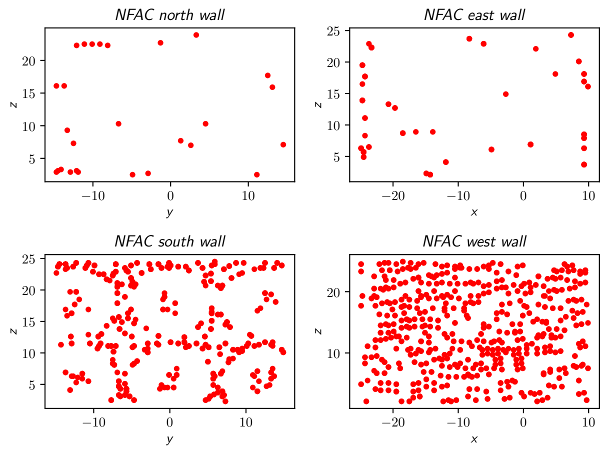


Figure 7.8: Waypoint distribution for the NFAC building

Computational time was also measured for the generation of waypoints of the NFAC octomap. Table 7.2 shows the results for the computational time and the number of waypoints generated for the four walls.

Table 7.2: NFAC waypoint generation time and waypoints found results

NFAC wall	Time	Waypoints found
North wall	1061.9 ms	27
East wall	507.604 ms	112
South wall	514.409 ms	731
West wall	499.927 ms	898

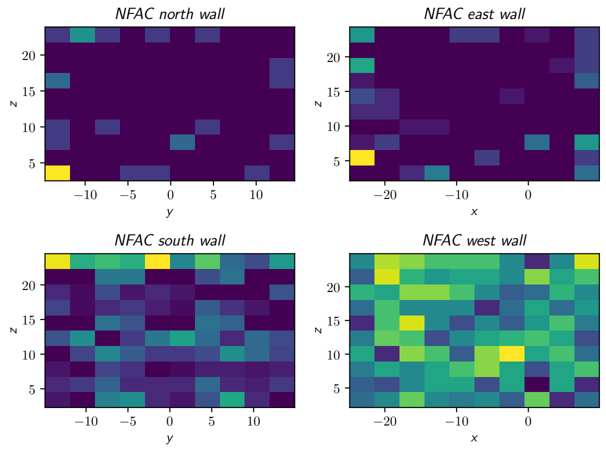
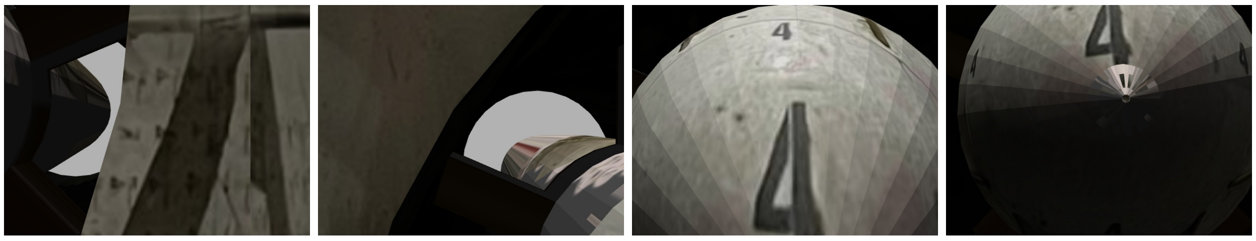


Figure 7.9: Waypoint distribution 2D histogram

7.5 Inspection Data Results

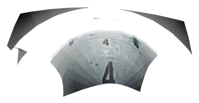
The proposed drone inspection system is capable of recording images, point-clouds and videos at specific waypoints. The waypoints found in section 7.4 are used to run a simulated test case to navigate and record both images and point-cloud data form the NFAC south wall (Figure 7.9 (c)). Figure 7.10 shows the images taken from the simulated inspection mission. The images show part of the wind tunnel's fans. Additionally, figures 7.11 and 7.12 show the point-cloud data images for the propellers from different viewing angles.

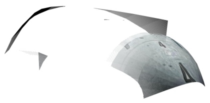


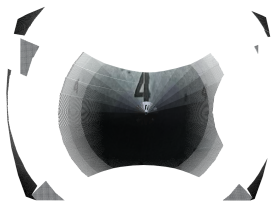
(a) NFAC image 0 (b) NFAC image 1 (c) NFAC image 2 (d) NFAC image 3

Figure 7.10: Inspection images taken from the NFAC model

The demonstrated images and point-cloud data can be used for post-flight image or point-cloud processing, and/or human visual inspection. Additionally, the collected data can be used to certify or validate the inspection of an area or object according to the building's safety standard.


(a) NFAC point-cloud 0


(b) NFAC point-cloud 1


(c) NFAC point-cloud 2

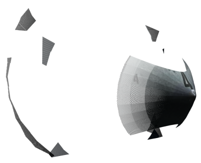
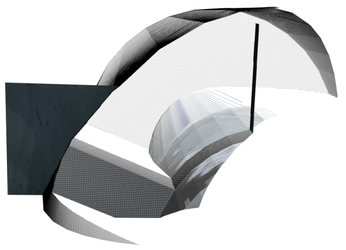

(d) NFAC point-cloud 0

Figure 7.11: Inspection point-cloud taken from the NFAC model, showing different perspectives


(a) NFAC fans point-cloud

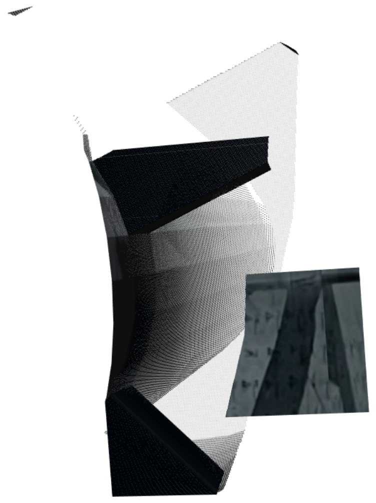

(b) NFAC fans point-cloud

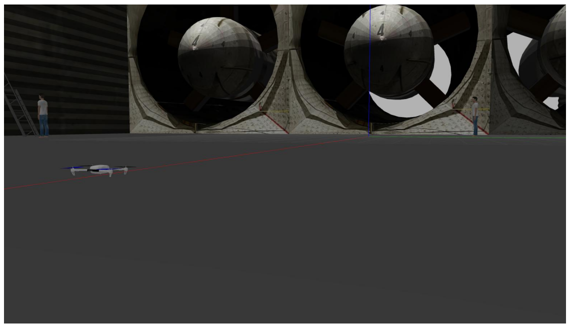
Figure 7.12: Inspection point-cloud for the inside of the NFAC fans

7.6 Case Study: NFAC

A simulated case study was performed on the NFAC, were the objective was to test and demonstrate the proposed system on a realistic and practical scenario. This study case demonstrates the ability for the system to perform an initial inspection plan by autonomously generating waypoints from a given octomap. The drone then detects a higher priority QR code that triggers an navigation interrupt that selects the new waypoint as the goal for the path planner. At the same time, the original inspection plan is queued again, so that the drone can continue the mission after inspecting the higher priority QR code detection. The drone will then complete the original inspection mission and produce image and point-cloud data. The following sections show the sequence of events for a realistic inspection mission using the proposed drone inspection system.

7.6.1 Drone Takeoff and Initial Inspection Plan

The test program initially requests the drone to takeoff and position itself at at the location $\mathbf{r} = [0.0 \quad 0.0 \quad 3.0]$. Figure 7.13 demonstrate the simulation images for the takeoff.


(a) Drone initial simulation state and position

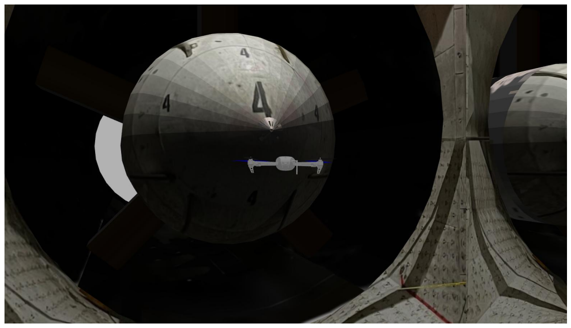

(b) Drone takeoff position and view on the simulation environment

Figure 7.13: Drone initial position and takeoff

After takeoff, the drone requests a service to generate waypoints on the south wall of the NFAC mesh model using a previously recorded octomap. This results in the generation of waypoints that will be used for an inspection mission. The inspection height for the wall was limited to $z = 10m$. Figure 7.14 shows the generated waypoints for the study case.

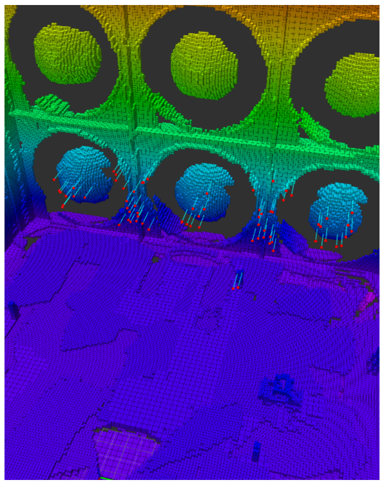
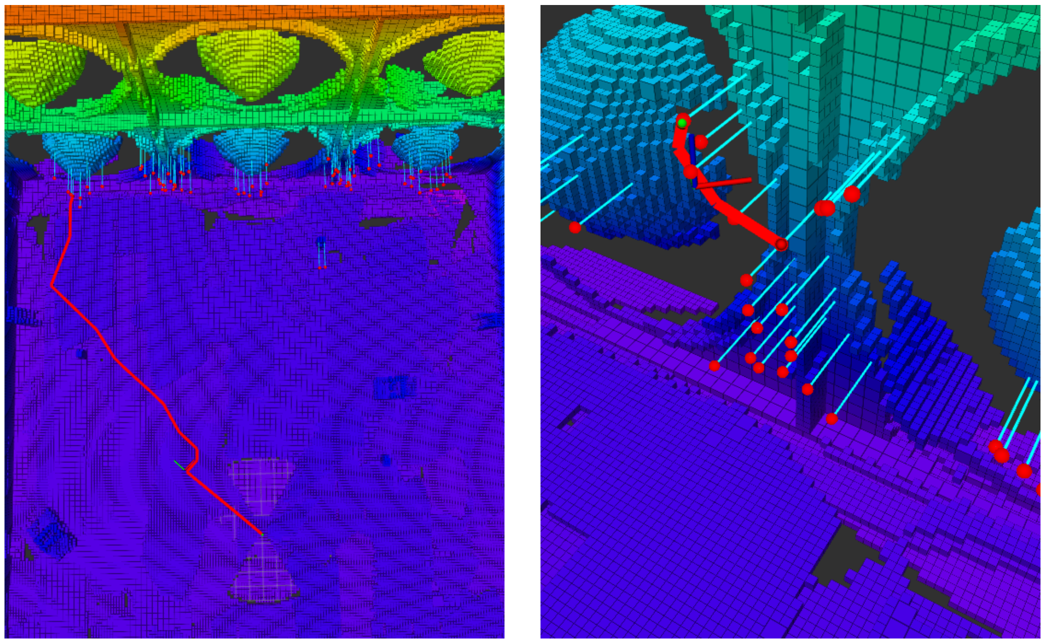


Figure 7.14: Inspection waypoints generated at the south wall of the NFAC model. The height is limited to $z = 10m$

7.6.2 Drone Path Planning

The inspection waypoints are then passed to the planner to find collision-free trajectories for the generated inspection waypoints. If a valid trajectory is found, the planner sends the navigation positions and headings to the drone. This results in a series of path planning trajectories between the current drone's position and an inspection waypoint goal. Figures 7.15(a) and 7.15(b) show two generated paths between waypoints



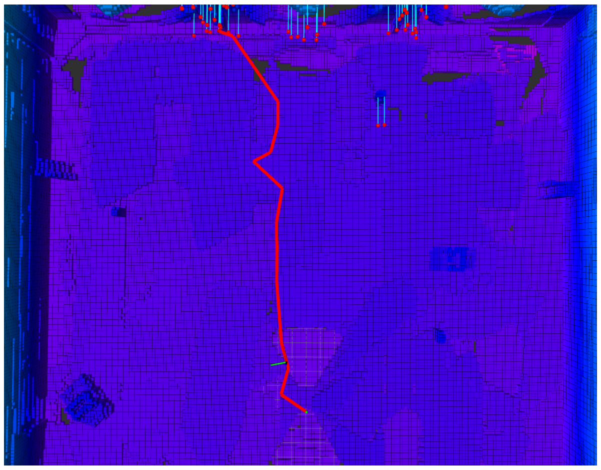
(a) Path planner inspection trajectory from initial takeoff position

(b) Path planner inspection trajectory from two waypoints

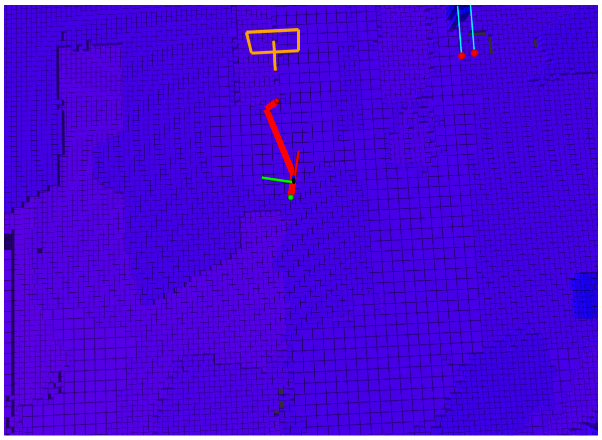
Figure 7.15: Path planner inspection trajectory from two waypoints

7.6.3 Drone QR Detection and Path Re-Planning

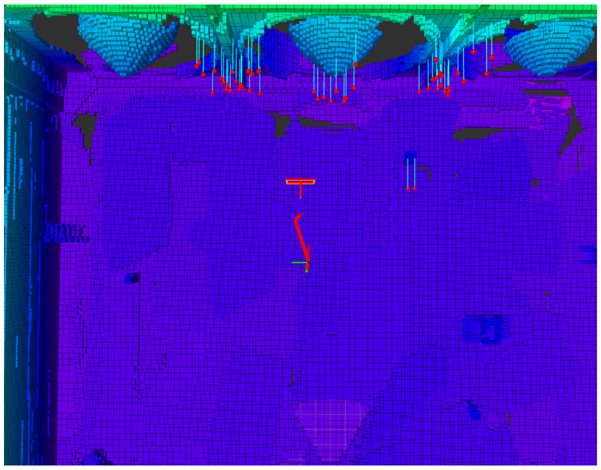
The drone QR detection and path re-planning is divided into four stages: Original path trajectory, detection, path re-planning based on generated waypoint detection, and return to original path. The first stage is the planned trajectory based on the original inspection waypoints. The second stage observes and reacts to a detection event by generating a new inspection waypoint as discussed in section 6.3. The third stage sends a new higher priority inspection waypoint for the computation of a navigation trajectory based on the detected QR code's position. The fourth and last stage re-plans the interrupted trajectory using the drone's current position, this takes the drone back to normal operation and to the original inspection mission. Figures 7.16 and 7.17 shows the detection and re-planning stages using visual markers and in real-time simulation images.



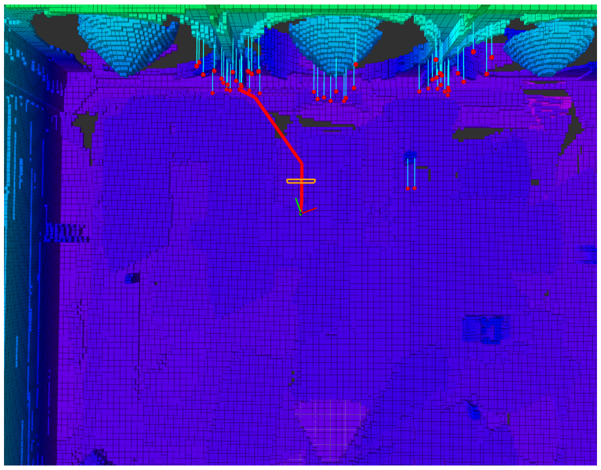
(a) Original trajectory to inspect NFAC wall



(b) Marker detection and waypoint generation form surface normal

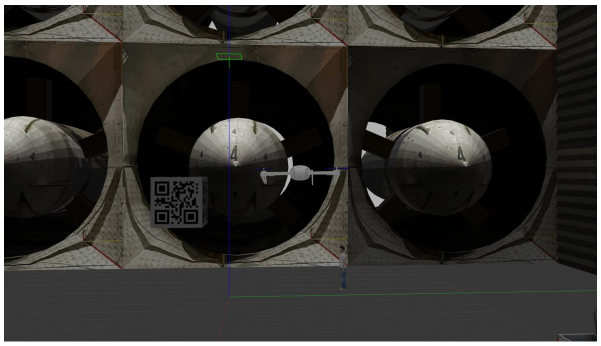


(c) Path re-planning based on high priority waypoint

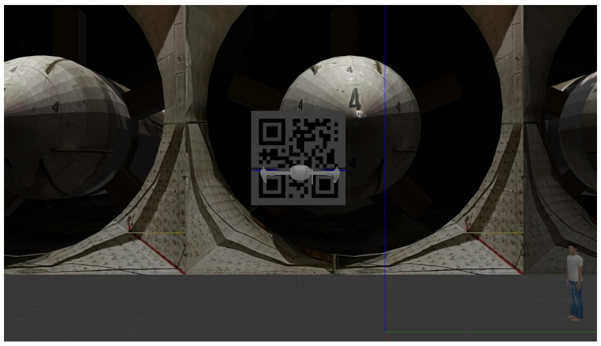


(d) Original inspection re-planning, back to normal operation

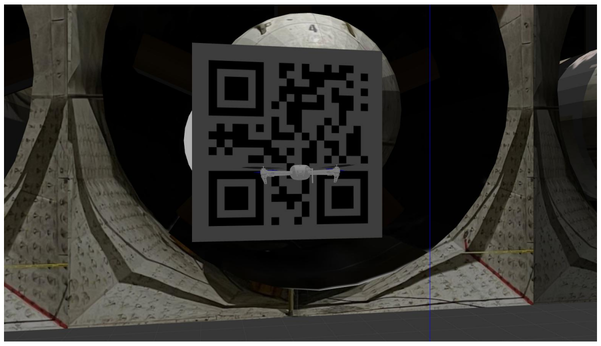
Figure 7.16: Marker detection and inspection using high priority navigation waypoints



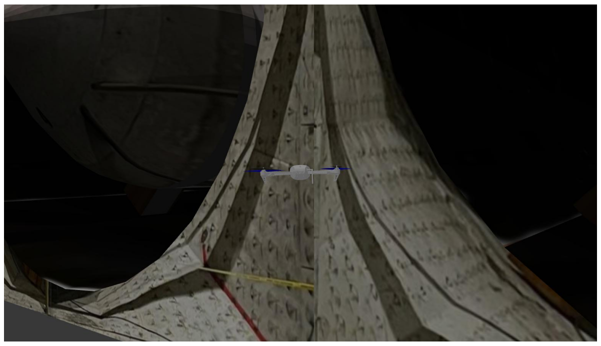
(a) Original trajectory to inspect NFAC wall



(b) Marker detection and waypoint generation form surface normal



(c) Path re-planning based on high priority waypoint



(d) Original inspection re-planning, back to normal operation

Figure 7.17: Marker detection and inspection using high priority navigation waypoints

7.6.4 Drone Landing and Inspection Results

Finally, once all detections and original waypoints are inspected, the drone requests a landing service. The drone can request to land at the current position or at the takeoff position. In this study case, a land request places the drone at the floor using the drone's current location.

The recorded images, point-cloud and videos will be available and stored on the drone's computer storage and will be similar to those shown in section 7.5

Chapter 8: Conclusion

This chapter present the conclusion of the conducted study and the previous chapters. It will also summarize the key findings and answers to the initial research question. Furthermore, this chapter also outlines the contributions and limitations of the study, as well as future research recommendations.

This study aimed to develop a drone system that could autonomously inspect industrial and scientific infrastructure. To that end, the results and work presented in chapters 4 through 7 show that such system is possible, and that autonomy for inspection can be achieved with the use of deep learning neural networks and QR codes. Moreover, this study shows that complex reactive navigation architectures can be programmed, extended and maintained using hierarchical state machines. The study also demonstrated a reliable method to generate inspection waypoints on octomaps, object detection and QR codes. In conjunction, these systems demonstrate a drone inspection platform with a higher level of autonomy. Additionally, this study demonstrated in chapter 2 that autonomy is an important characteristic of drone inspection systems, as current systems do not react or detect critical damages or defects on infrastructure.

The study conducted proposed a novel drone inspection system that enhances the level of autonomy for in-flight decision making. Key ideas, for the detection and navigation of damage or defects were presented in this study. First, by presenting a reactive drone navigation architecture that prioritizes detection inspection waypoints. Second, by object and QR detection, and third, by automating the generation of inspection waypoints using an octomap and detections bounding boxes and surfaces.

Moreover, this study it attempts to solve the lack of automation of most inspection drone systems. The presented solution attempts to solve the autonomy problem by increasing the perception of the drone through the use of deep learning object recognition and QR. Moreover, this increased perception level leads to complex actions and behaviors through the use of a hierarchical state machine.

Section 7.6 demonstrated a practical application of the proposed drone system. From that section, it can be observed that the drone system can react to QR detections with a new navigation plan and inspection actions. The data produced from the example inspection mission also demonstrates the utility of such platform, and applications for pos-flight data analysis of image, point-cloud and video data, as well as automated detection data.

However, this study did not cover collision avoidance, which is an important component of all robotic navigation platforms. The lack of collision avoidance limits the utility of this drone to only static environments. Additionally, the drone requires an octomap before a mission takes place. To record an octomap, this study proposed manual drone flights around the inspection area. Of course, this limits the autonomy of the system, which goes against the goal of this study. Lastly, no field flights were performed during the conducted study. Evidently, a practical use case can not be verified until a field test is done. However, flight control software did not report bugs or glitches during the simulated mission, which shows that deploying the software into the hardware platform will not present fatal or critical errors.

Therefore, future work requires field testing on a hardware platform, which will finally certify a practical use and utility of the drone inspection system. Additionally, more work needs to be done towards collision avoidance, as this will increase the applicability of the system to dynamics environments. Lastly, more research and development needs to be performed on exploration algorithms that will allow the drone to map and navigate unknown areas. Ultimately, this will add another layer of automation for mapping and exploration.

Bibliography

[1] Georg Graetz and Guy Michaels. “Robots at Work”. In: *The Review of Economics and Statistics* 100.5 (Dec. 2018), pp. 753–768. ISSN: 0034-6535, 1530-9142. DOI: 10.1162/rest_a_00754. URL: https://direct.mit.edu/rest/article/100/5/753-768/58489 (visited on 10/28/2022).

[2] Sophie Jordan et al. “State-of-the-art technologies for UAV inspections”. In: *IET Radar, Sonar & Navigation* 12.2 (Feb. 2018), pp. 151–164. ISSN: 1751-8792, 1751-8792. DOI: 10.1049/iet-rsn.2017.0251. URL: https://onlinelibrary.wiley.com/doi/10.1049/iet-rsn.2017.0251 (visited on 09/19/2022).

[3] Tarek Rakha and Alice Gorodetsky. “Review of Unmanned Aerial System (UAS) applications in the built environment: Towards automated building inspection procedures using drones”. In: *Automation in Construction* 93 (Sept. 2018), pp. 252–264. ISSN: 09265805. DOI: 10.1016/j.autcon.2018.05.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S0926580518300165 (visited on 09/19/2022).

[4] Widodo Budiharto et al. “Fast Object Detection for Quadcopter Drone Using Deep Learning”. In: *2018 3rd International Conference on Computer and Communication Systems (ICCCS)*. 2018 3rd International Conference on Computer and Communication Systems (ICCCS). Nagoya, Japan: IEEE, Apr. 2018, pp. 192–195. ISBN: 978-1-5386-6350-9. DOI: 10.1109/CCOMS.2018.8463284. URL: https://ieeexplore.ieee.org/document/8463284/ (visited on 10/03/2022).

[5] Saheba Bhatnagar, Laurence Gill, and Bidisha Ghosh. “Drone Image Segmentation Using Machine and Deep Learning for Mapping Raised Bog Vegetation Communities”. In: *Remote Sensing* 12.16 (Aug. 12, 2020), p. 2602. ISSN: 2072-4292. DOI: 10.3390/rs12162602. URL: https://www.mdpi.com/2072-4292/12/16/2602 (visited on 10/03/2022).

[6] Igor Bisio et al. “A Systematic Review of Drone Based Road Traffic Monitoring System”. In: *IEEE Access* 10 (2022). Conference Name: IEEE Access, pp. 101537–101555. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3207282.

[7] UM Rao Mogili and B B V L Deepak. “Review on Application of Drone Systems in Precision Agriculture”. In: *Procedia Computer Science* 133 (2018), pp. 502–509. ISSN: 18770509. DOI: 10.1016/j.procs.2018.07.063. URL: https://linkinghub.elsevier.com/retrieve/pii/S1877050918310081 (visited on 10/03/2022).

[8] Balmukund Mishra et al. “Drone-surveillance for search and rescue in natural disaster”. In: *Computer Communications* 156 (Apr. 2020), pp. 1–10. ISSN: 01403664. DOI: 10.1016/j.comcom.2020.03.

012. URL: https://linkinghub.elsevier.com/retrieve/pii/S0140366419318602 (visited on 10/03/2022).

[9] Malik Demirhan and Chinthaka Premachandra. "Development of an Automated Camera-Based Drone Landing System". In: *IEEE Access* 8 (2020). Conference Name: IEEE Access, pp. 202111–202121. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3034948.

[10] Jin Kim et al. "Autonomous flight system using marker recognition on drone". In: *2015 21st Korea-Japan Joint Workshop on Frontiers of Computer Vision (FCV)*. 2015 21st Korea-Japan Joint Workshop on Frontiers of Computer Vision (FCV). Jan. 2015, pp. 1–4. DOI: 10.1109/FCV.2015.7103712.

[11] Roland Siegwart et al. *Introduction to Autonomous Mobile Robots, Second Edition*. Cambridge, UNITED STATES: MIT Press, 2011. URL: http://ebookcentral.proquest.com/lib/utep/detail.action?docID=3339191 (visited on 09/13/2022).

[12] Armin Hornung et al. "OctoMap: an efficient probabilistic 3D mapping framework based on octrees". In: *Autonomous Robots* 34.3 (Apr. 2013), pp. 189–206. ISSN: 0929-5593, 1573-7527. DOI: 10.1007/s10514-012-9321-0. URL: http://link.springer.com/10.1007/s10514-012-9321-0 (visited on 03/14/2022).

[13] Jan Faigl. "Grid and Graph based Path Planning Methods". In: (), p. 87.

[14] Karthik Karur et al. "A Survey of Path Planning Algorithms for Mobile Robots". In: *Vehicles* 3.3 (Aug. 4, 2021), pp. 448–468. ISSN: 2624-8921. DOI: 10.3390/vehicles3030027. URL: https://www.mdpi.com/2624-8921/3/3/27 (visited on 10/04/2022).

[15] Miro Samek. *Practical UML statecharts in C/C++: event-driven programming for embedded systems*. 2nd ed. OCLC: ocn214307694. Amsterdam ; Boston: Newnes/Elsevier, 2009. 712 pp. ISBN: 978-0-7506-8706-5.

[16] David Harel. "Statecharts: A visual Formalism For Complex Systems*". In: *Science of Computer Programming* 8 (1987), pp. 231–274.

[17] Michael Nielsen. "Neural Networks and Deep Learning". In: (), p. 225.

[18] Shi Dong, Ping Wang, and Khushnood Abbas. "A survey on deep learning and its applications". In: *Computer Science Review* 40 (May 2021), p. 100379. ISSN: 15740137. DOI: 10.1016/j.cosrev.2021.100379. URL: https://linkinghub.elsevier.com/retrieve/pii/S1574013721000198 (visited on 10/04/2022).

[19] Charu C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Cham: Springer International Publishing, 2018. ISBN: 978-3-319-94463-0. DOI: 10.1007/978-3-319-94463-0. URL: http://link.springer.com/10.1007/978-3-319-94463-0 (visited on 11/08/2022).

[20] Milan Banić et al. “INTELLIGENT MACHINE VISION BASED RAILWAY INFRASTRUCTURE INSPECTION AND MONITORING USING UAV”. In: *Facta Universitatis, Series: Mechanical Engineering* 17.3 (Nov. 29, 2019), p. 357. ISSN: 2335-0164, 0354-2025. DOI: 10.22190/FUME190507041B. URL: http://casopisi.junis.ni.ac.rs/index.php/FUMechEng/article/view/5230 (visited on 09/19/2022).

[21] Tianqi Mao et al. “Defect Recognition Method Based on HOG and SVM for Drone Inspection Images of Power Transmission Line”. In: *2019 International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS)*. 2019 International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS). May 2019, pp. 254–257. DOI: 10.1109/HPBDIS.2019.8735466.

[22] João Marcelo Teixeira et al. “Teleoperation Using Google Glass and AR, Drone for Structural Inspection”. In: *2014 XVI Symposium on Virtual and Augmented Reality*. 2014 XVI Symposium on Virtual and Augmented Reality. May 2014, pp. 28–36. DOI: 10.1109/SVR.2014.42.

[23] Junwon Seo, Luis Duque, and Jim Wacker. “Drone-enabled bridge inspection methodology and application”. In: *Automation in Construction* 94 (Oct. 2018), pp. 112–126. ISSN: 09265805. DOI: 10.1016/j.autcon.2018.06.006. URL: https://linkinghub.elsevier.com/retrieve/pii/S0926580517309755 (visited on 09/19/2022).

[24] Javier Irizarry, Masoud Gheisari, and Bruce N Walker. “USABILITY ASSESSMENT OF DRONE TECHNOLOGY AS SAFETY INSPECTION TOOLS”. In: (), p. 19.

[25] Zahid Ali Siddiqui and Unsang Park. “A Drone Based Transmission Line Components Inspection System with Deep Learning Technique”. In: *Energies* 13.13 (Jan. 2020). Number: 13 Publisher: Multidisciplinary Digital Publishing Institute, p. 3348. ISSN: 1996-1073. DOI: 10.3390/en13133348. URL: https://www.mdpi.com/1996-1073/13/13/3348 (visited on 09/19/2022).

[26] A. S. M. Shihavuddin et al. “Wind Turbine Surface Damage Detection by Deep Learning Aided Drone Inspection Analysis”. In: *Energies* 12.4 (Jan. 2019). Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, p. 676. ISSN: 1996-1073. DOI: 10.3390/en12040676. URL: https://www.mdpi.com/1996-1073/12/4/676 (visited on 09/19/2022).

[27] Shuo Zhang et al. “A Novel Intelligent Recognition of Video Objects in Drone Routing Inspection based on Deep Learning”. In: *2022 7th International Conference on Intelligent Computing and Signal Processing (ICSP)*. 2022 7th International Conference on Intelligent Computing and Signal Processing (ICSP). Apr. 2022, pp. 250–255. DOI: 10.1109/ICSP54964.2022.9778783.

[28] Juan A. Besada et al. “Drone Mission Definition and Implementation for Automated Infrastructure Inspection Using Airborne Sensors”. In: *Sensors* 18.4 (Apr. 2018). Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, p. 1170. ISSN: 1424-8220. DOI: 10.3390/s18041170. URL: https://www.mdpi.com/1424-8220/18/4/1170 (visited on 09/19/2022).

[29] Yahya Zefri et al. “Thermal Infrared and Visual Inspection of Photovoltaic Installations by UAV Photogrammetry—Application Case: Morocco”. In: *Drones* 2.4 (Nov. 23, 2018), p. 41. ISSN: 2504-446X. DOI: 10.3390/drones2040041. URL: http://www.mdpi.com/2504-446X/2/4/41 (visited on 09/19/2022).

[30] Reem Ashour et al. “Site inspection drone: A solution for inspecting and regulating construction sites”. In: *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*. 2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS). ISSN: 1558-3899. Oct. 2016, pp. 1–4. DOI: 10.1109/MWSCAS.2016.7870116.

[31] Owen McAree, Jonathan M. Aitken, and Sandor M. Veres. “A model based design framework for safety verification of a semi-autonomous inspection drone”. In: *2016 UKACC 11th International Conference on Control (CONTROL)*. 2016 UKACC 11th International Conference on Control (CONTROL). Aug. 2016, pp. 1–6. DOI: 10.1109/CONTROL.2016.7737551.

[32] Rutuja Shivgan and Ziqian Dong. “Energy-Efficient Drone Coverage Path Planning using Genetic Algorithm”. In: *2020 IEEE 21st International Conference on High Performance Switching and Routing (HPSR)*. 2020 IEEE 21st International Conference on High Performance Switching and Routing (HPSR). ISSN: 2325-5609. May 2020, pp. 1–6. DOI: 10.1109/HPSR48589.2020.9098989.

[33] Samira Hayat et al. “Multi-objective drone path planning for search and rescue with quality-of-service requirements”. In: *Autonomous Robots* 44.7 (Sept. 1, 2020), pp. 1183–1198. ISSN: 1573-7527. DOI: 10.1007/s10514-020-09926-9. URL: https://doi.org/10.1007/s10514-020-09926-9 (visited on 09/21/2022).

[34] Wei Jing et al. “Multi-UAV Coverage Path Planning for the Inspection of Large and Complex Structures”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). ISSN: 2153-0866. Oct. 2020, pp. 1480–1486. DOI: 10.1109/IROS45743.2020.9341089.

[35] Anastasios Zompas. “Development of a Three Dimensional Path Planner for Aerial Robotic Workers”. In: *Robotics and Mechatronics* (), p. 39.

[36] Bergström Jerker. *Path Planning with Weighted Wall Regions using OctoMap*. 2018. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:ltu:diva-67682 (visited on 09/21/2022).

[37] Fangyu Li et al. “Universal path planning for an indoor drone”. In: *Automation in Construction* 95 (Nov. 2018), pp. 275–283. ISSN: 09265805. DOI: 10.1016/j.autcon.2018.07.025. URL: https://linkinghub.elsevier.com/retrieve/pii/S0926580517311184 (visited on 09/21/2022).

[38] Helen Oleynikova et al. “Continuous-time trajectory optimization for online UAV replanning”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). ISSN: 2153-0866. Oct. 2016, pp. 5332–5339. DOI: 10.1109/IROS.2016.7759784.

[39] Zhaoliang Zheng, Thomas R. Bewley, and Falko Kuester. “Point Cloud-Based Target-Oriented 3D Path Planning for UAVs”. In: *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2020 International Conference on Unmanned Aircraft Systems (ICUAS). ISSN: 2575-7296. Sept. 2020, pp. 790–798. DOI: 10.1109/ICUAS48674.2020.9213894.

[40] Hailong Huang, Andrey V. Savkin, and Chao Huang. “Reliable Path Planning for Drone Delivery Using a Stochastic Time-Dependent Public Transportation Network”. In: *IEEE Transactions on Intelligent Transportation Systems* 22.8 (Aug. 2021). Conference Name: IEEE Transactions on Intelligent Transportation Systems, pp. 4941–4950. ISSN: 1558-0016. DOI: 10.1109/TITS.2020.2983491.

[41] Hailong Qin et al. “Autonomous Exploration and Mapping System Using Heterogeneous UAVs and UGVs in GPS-Denied Environments”. In: *IEEE Transactions on Vehicular Technology* 68.2 (Feb. 2019). Conference Name: IEEE Transactions on Vehicular Technology, pp. 1339–1350. ISSN: 1939-9359. DOI: 10.1109/TVT.2018.2890416.

[42] *Introduction · MAVLink Developer Guide*. URL: https://mavlink.io/en/ (visited on 10/17/2022).

[43] *Controller Diagrams — PX4 User Guide*. URL: https://docs.px4.io/main/en/flight_stack/controller_diagrams.html (visited on 10/17/2022).

[44] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. Apr. 16, 2017. arXiv: 1704.04861[cs]. URL: http://arxiv.org/abs/1704.04861 (visited on 10/06/2022).

[45] Wei Liu et al. “SSD: Single Shot MultiBox Detector”. In: vol. 9905. 2016, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2. arXiv: 1512.02325[cs]. URL: http://arxiv.org/abs/1512.02325 (visited on 10/06/2022).

[46] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. Feb. 20, 2015. arXiv: 1405.0312[cs]. URL: http://arxiv.org/abs/1405.0312 (visited on 10/09/2022).

[47] *ZBar bar code reader*. URL: https://zbar.sourceforge.net/ (visited on 10/31/2022).

[48] Alex Bewley et al. “Simple Online and Realtime Tracking”. In: *2016 IEEE International Conference on Image Processing (ICIP)*. Sept. 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003. arXiv: 1602.00763[cs]. URL: http://arxiv.org/abs/1602.00763 (visited on 10/31/2022).

[49] Enric Galceran and Marc Carreras. “A survey on coverage path planning for robotics”. In: *Robotics and Autonomous Systems* 61.12 (Dec. 2013), pp. 1258–1276. ISSN: 09218890. DOI: 10.1016/j.robot.2013.09.004. URL: https://linkinghub.elsevier.com/retrieve/pii/S092188901300167X (visited on 03/14/2022).

[50] *NASA - National Full-Scale Aerodynamics Complex (NFAC)*. URL: https://www.nasa.gov/centers/ames/multimedia/images/2005/nfac.html (visited on 11/01/2022).

Acronyms

AI Artifical Intelligence 2, 11

API Application Programming Interface 25

cSTER Center for Space Exploration Technology Research v, 81

EKF Extended Kalman Filter 26

FSM Finite State Machine 8, 37

GCS Ground Control Station 1, 21, 27

GPS Global Positioning System 23, 26

GUI Graphical User Interface 24–27

HOG Histogram of Oriented Gradients 19, 20

IMU Inertial Measurement Unit 26

MDS Mission Definition System 21

NFAC National Full-Scale Aerodynamics Complex xii, 2, 33, 34, 58, 59, 61, 63–66, 68, 69

PCA Principal Component Analysis 19

QR Quick Response vi, xii, 1–3, 18, 24, 37, 41, 42, 44, 45, 47, 49, 52, 55–58, 61, 65, 67, 68, 70

ROS Robot Operating System viii, 27–30

RRT Rapidly Exploring Random Trees 7, 22

SITL Software In The Loop 24, 26

SSD Single Shot MultiBox Detector viii, xi, 41–44

SVM Support Vector Machine 19

UAV Unmanned Aerial Vehicle vi, 19, 21–23

UGV Unmanned Ground Vehicle vi, 23

UTEP University of Texas at El Paso 81

Appendix A: A* algorithm

Algorithm 1 A* search algorithm

```
1: procedure A*($start, goal$)
2:     $open_{set} \leftarrow start$
3:     $g_{score}[1:n] \leftarrow \infty$
4:     $g_{score}[0] \leftarrow 0$
5:     $f_{score}[1:n] \leftarrow \infty$
6:     $f_{score}[0] \leftarrow h(0)$
7:     while $open_{set} \neq$ empty do
8:         $current \leftarrow min_{f_{score}}(open_{set})$
9:         if $current == goal$ then
10:            RECONSTRUCT_PATH($current$)
11:            return
12:        end if
13:        $open_{set}.remove(current)$
14:        for $neighbor \in current.neighbors()$ do
15:            $g_{temp} \leftarrow g_{score}[current] + d(current, neighbor)$
16:            if $g_{temp} < g_{score}[neighbor]$ then
17:                $g_{score}[neighbor] \leftarrow g_{temp}$
18:                $f_{score}[neighbor] \leftarrow g_{temp} + h(neighbor)$
19:                if $neighbor \notin open_{set}$ then
20:                    $open_{set}.add(neighbor)$
21:                end if
22:            end if
23:        end for
24:    end while
25: end procedure
```

Curriculum Vita

Alejandro Martinez Acosta was born in Ciudad Juarez, Chihuahua, Mexico in December 11 1992. Alejandro migrated to the United States of America in 2008 as a lawful resident, where he attended Hanks high school and Coronado high school. Upon graduating high school, Alejandro attended the University of Texas at El Paso (UTEP), where he earned a Bachelor's of Science in Electrical and Computer Engineering. Alejandro then pursued a Master's of Science in Computer Engineering at UTEP. While completing his studies, Alejandro became a lawful US citizen in 2017. In 2018, Alejandro completed his Master's degree and continued his graduate studies by pursuing a Mechanical Engineering Doctoral degree.

Alejandro completed an internship at Sandia laboraties on 2018, where he developed a low-cost Aerosol Jet Printer using a microcontroller system. Alejandro also completed three more internships at NASA AMES between 2021 and 2022, where he worked on an autonomous drone system for the inspection of wind tunnels. Alejandro was also graduate advisor for the IEEE school organization, where he was in charge of the micromouse competition.

Currently, Alejandro works as a Research Assistant at the cSTER. He lives in El Paso Texas, USA.