CRANFIELD UNIVERSITY

LI. JIAN

# SIMULATION OF AUTONOMOUS UAV NAVIGATION WITH COLLISION AVOIDANCE AND SPATIAL AWARENESS

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING

MSc by research in manufacturing

MSc by research
Academic Year: 2017–2018

Supervisors: He. Hongmei, Tiwari. Ashutosh
August 2019

CRANFIELD UNIVERSITY

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING
MSc by research in manufacturing

MSc by research

Academic Year: 2017–2018

LI. JIAN

# Simulation of autonomous UAV navigation with collision avoidance and spatial awareness

Supervisors: He. Hongmei, Tiwari. Ashutosh
August 2019

# Abstract

The goal of this thesis is to design a collision-free autonomous UAV navigation system with spatial awareness ability within a comprehensive simulation framework. The navigation system is required to find a collision-free trajectory to a randomly assigned 3D target location without any prior map information. The implemented navigation system contains four main components: mapping, localisation, cognition and control system, where the cognition system makes execution command based on the perceived position information about obstacles and UAV itself from mapping and localisation system respectively. The control system is responsible for executing the input command made from the cognition system. The implementation for the cognition system is split into three case studies for real-life scenarios, which are restricted area avoidance, static obstacle avoidance and dynamic obstacles. The experiment results in the three cases have been conducted, and the UAV is capable of determining a collision-free trajectory under all three cases of environments. All simulated components were designed to be analogous to their real-world counterpart. Ideally, the simulated navigation framework can be transferred to a real UAV without any changes. The simulation framework provides a platform for future robotic research. As it is implemented in a modular way, it is easier to debug. Hence, the system has good reliability. Moreover, the system has good readability, maintainability and extendability.

## Keywords

UAV; Autonomous Navigation; Collision Avoidance; Path Planning; Simulation

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| AHRS | Attitude Heading and Reference System |
| CAD | Computer Aided Design |
| CES | Consumer Electronics Show |
| CoG | Centre of Gravity |
| DoF | Degree of Freedom |
| EFK | Extended Kalman Filter |
| EO | Electro Optical |
| GA | Generic Algorithm |
| GPS | Global Positioning System |
| FoV | Field of View |
| IR | InfraRed |
| IMU | Inertial Measurement Unit |
| INS | Inertial Navigation System |
| KF | Kalman Filter |
| LED | Light Emitting Diode |
| MAV | Micro Aerial Vehicle |
| MDP | Markov Decision Process |
| NN | Neural Network |
| RA | Restricted Area |
| RGBW | Red Green Blue White |
| RPA | Remotely Piloted Aircraft |

| | |
|---|---|
| RRT | Rapidly-expanding Random Tree |
| ROS | Robot Operating System |
| RViz | Ros Visualisation (Vizualisation) |
| SA | Spatial Awareness |
| SA | Situation Awareness |
| SAT | Separating Axis Theorem |
| SDF | Simulator Description Format |
| SLAM | Simultaneous Localisation And Mapping |
| ToF | Time of Flight |
| UAV | Unmanned Aerial Vehicle |
| UAS | Unmanned Aerial System |
| UFK | Unscented Kalman Filter |
| URE | User Range Error |
| VD | Voronoi Diagram |

# Acknowledgements

I would like to give many thanks to my supervisor, Hongmei He, who had given me a lot of valued advice. Autonomous UAV navigation was a new area for me, although I had some background of programming. However, she brought me into the new area step by step. I have enjoyed my research in this area.

I would warmly thank my second supervisor, Professor Ashutosh Tiwari for his encouragement and guidance throughout the time my work went on.

I also would like to thank my parents for their support for my work.

I would like to thank my girlfriend Lili Pang, she offered a careful read-proof of this thesis and gave me a lot of valued advice.

Finally, I would like to thank all staff, working at the IT Services of Cranfield University, who had supported me setting up the working environment for the simulation system.

# Chapter 1

# Introduction

## 1.1 Background

Starting with the Industrial Revolution in the 18th century, automation systems were designed to assist or replace human beings to perform tedious and physically demanding labor tasks, throughout this constant innovation and evolution, it continuously improves human well-being and increases living standard by reducing the amount of human work required and provides a wide range of affordable products and amenities in contemporary society. They are no longer a vision for the future but a reality of the present. In general, automation can be defined as the technology by which a process is performed with minimal human assistance [12], and it uses various control mechanism to ensure the instruction is executed, relevant applications range from a simple gravity food dispenser to sizeable industrial control systems (as shown in Figure 1.1). Advanced automation system represents a level of capability and performance that surpass in many ways, the abilities of humans to accomplish the same activities [13].

(a) Gravity food dispenser [14]

(b) KUKA industrial robots being used at a bakery for food production [15]

Figure 1.1: Autonomous systems with differenct automation level

Within the scope of automation systems, UAV (Unmanned Aerial Vehicles) has attracted significant interest in a wide range of applications and became a major research topic in recent years. Although different terms are used to distinguish UAVs from ballistic vehicles, cruise missiles and artillery projectiles in military applications [16], UAV can be used to describe any aerial devices with no human pilot onboard, either guided autonomously or by remote control. The shape does not restrict UAV; it could be a fixed-wing aeroplane, a helicopter, a robotic bird or any device moving in the air. Figure 1.2 demonstrates an example of existing UAVs in different shapes. There are a number of terms may be used interchangeably to describe UAV, popular ones include UAS (Unmanned Aerial System), RPA (Remotely Piloted Aircraft), MAV (Micro Aerial Vehicle), drone, etc.

Figure 1.2: Different types of UAV [1]

Compared to manned aircraft, UAVs were originally used for missions that are ”dull, dirty or dangerous” for humans [17]. Arguably, the first military use of aerial devices came from the Chinese invention of the kites around 300 BC when they were used to lift men into the air to spot enemy armies and to follow their movement [18]. Similarly, the first recorded use of modern UAV was also built for the military purpose, serving as a balloon carrier loaded with explosives by the Australian military to drop bombs over the city of Venice, Italy [19]. Although kites and balloons would not be considered as aerial vehicles today, they have led to further advancement. The United States began developing UAV technology during the First World War and created the first pilotless aircraft, *Kettering Bug*, which could be controlled with the use of radio wave after launch and served as a great precursor of modern UAV.

Although many of these notable drones were built for the military purpose, the technology continues to advance and receive more attention in civilian applications. In 2010, France-based company Parrot unveiled its AR 1.0 at the Internation Consumer Electronics Show (CES) in Las Vegas. The quadcopter’s abilities were beyond what anyone had seen before and changed the future of civil UAVs. The UAV was demonstrated with iOS application-based controller by creating its own Wi-Fi network, then manoeuvred using the accelerometers of the iPhone combined with a video feed from the UAV’s forward-facing camera. The user could also take photo and videos from the video feed, which introduced the concept of aerial photography to the masses, though the flight time, maximum flying altitude, and photo quality were all minimal. However, the technology continued to thrive and led to the modern consumer UAV precursor. In 2018, the China-based manufacturer DJI introduced its Phantom Pro V2.0 [20], with a high-quality camera attached to the gimbal with significant improvements in image quality over the parrot predecessor. Moreover, it features an intelligent navigation system to compensate the external effect (e.g. wind) to improve its flight stabilisation, in addition with an InfraRed (IR) sensing system for multiple direction obstacle sensing and avoidance, it sparked the consumer and commercial UAV craze.

(a) The Australian baloon [21]

(b) Kettering Bug pilotless aircraft [22]

(c) Parrot AR 1.0 [23]

(d) DJI Phantom 4 Pro V2.0 [20]

Figure 1.3: History and development of UAV systems

## 1.2 Motivation

The ability of an autonomous vehicle could catapult the productivity and quality of various human activities. In 2019, iRobot announced its intelligent lawnmower Terra (Figure 1.4a), It only requires wireless beacons to be placed around the perimeter of a lawn, compared to the traditional automatic lawn mowing systems which require the user to define their working boundaries by laying down wires [24]. However, user is still required to teach the TERRA where to go and where to avoid with the assistant of a companion app, to define a grass heigh and make adjustments as needed. Another excellent example for the autonomous systems would be the state-of-art self-driving cars. In March 2018, Britain had officially started the ”UK Autodrive Trails,” with 40 self-driving pods taking to the pavements and street [25]. They were capable of travelling up to 15 miles per hour

and lasting up to 60 miles off one charge within a pre-defined route in the city of Milton Keynes (Figure 1.4b).

During the last few years, the ongoing trend towards the integration of robotic systems into UAV application is intensifying, with the high levels of connectivity through matured cellular, Bluetooth, wifi, and radio protocols in conjunction with smartphone ubiquity and increasingly comingled hardware and software has resulted in rapid advancement in UAV capabilities. In December 2016, Amazon had completed the first trial of its futuristic UAV-based delivery plan (Figure 1.4c) [26], to utilise UAV technology fly individual packages autonomously to customers. The system was designed to find the target location safely, and the UAV was able to operate autonomously from take-off to landing and return within 30 minutes after the order has been placed. More recently, in the 2018 PyeongChang Winter Olympics, Intel used more than 1200 purpose-built UAVs fly simultaneously above the stadium for the Opening Ceremony (Figure 1.4d) [27]. Each UAV features built-in LED lights create over 4 billion colour combinations based on RGBW (Red, Green, Blue and White) LED, which together formed a larger-than-life choreographed snowboarder and Olympic Rings illuminated the night sky, transformed UAV technology into an entirely new form of entertainment and created a memorial experience at the most-watched event in the world.

However, there are still various aspects that can be improved for the currently deployed automation system, for instance, most of the available autonomous cleaning solutions have a minimal perception of the working area as the Terra requires user to set up the boundaries for every new working environment through Wi-Fi [28], which could also jeopardise the working process when there is only limited signal. Additionally, the working area is covered using a random rather than a systematic and optimised method, which reduces the working efficiency. Furthermore, the self-driving pods could only work within a public environment, which will be rather tedious and impractical for the deployment stage, to get access to a full map of all possible working area.

For most of today's deployed UAV systems are teleoperated and semi-automatic,

(a) The iRobot Terra robot lawn mower [24]


(b) The self-driving pod in Milton Keynes [25]


(c) Amazon delivery drone [26]

(d) UAV light show at PyeongChang Winter Olympics 2018 [27]

Figure 1.4: Example applications of autonomous system

which still rely on humanitarian assistance [28]. For example, Intel used the traditional centralised solution, and the UAV is required to have the ability to communicate their flight status (speed, height, etc.) with ground control station or neighbouring aircraft, and operated by taking command from them. Although Intel was able to control over 1200 UAVs by a single pilot with the assistance of sophisticated software, it will be more difficult and computationally expensive for the pilot to keep track of every single aircraft within the increasingly crowded airspace [28, 6]. As a result, the UAV system needs to have an autonomous navigation system to operate without any human assistance. Additionally, most people use the term automatic and independent interchangeably; it is increasingly tempting to distinguish them as the level of automation improves. Generally, the difference between them is the degree of human intervention. An automatic UAV does not have the level of intelligence or independence that an autonomous UAV has. For example, a genuinely autonomous UAV would decide on target location and route as well as control during the flight. An automatic UAV would follow orders about goal and direction, and may only adopt some distance-keeping guidance to avoid the collision. However, there are still various aspects that can be improved for the currently deployed automation system, for instance, most of the available autonomous cleaning solutions have a minimal perception of the working area as the Terra requires user to set up the boundaries for every new working environment through Wi-Fi [28], which could also jeopardise the working process when there is only limited signal. Additionally, the working area is covered using a random rather than a systematic and optimised method, which reduces the working efficiency. Furthermore, the self-driving pods could only work within a public environment, which will be rather tedious and impractical for the deployment stage, to get access to a full map of all possible working area.

Motivated by this, this thesis will concentrate on ensuring the UAV can navigate itself safely in an unknown environment in a proper manner. Successful application of autonomous UAVs have consequences and implication in various field, such as reconnaissance, communication, goods delivery, weather forecast, etc.

## 1.3 Aims and objectives

This thesis aims to design and implement a simulation framework to test algorithms for an autonomous UAV navigation system, featuring spatial awareness and collision-free ability. Where spatial awareness is the UAV's ability to acknowledge its location and orientation during the navigation process. Collision-free navigation indicates that the UAV must avoid any obstacles while reaching a randomly assigned target location. To be more specific, the objectives are listed as follows:

- To implement static obstacle avoidance, the UAV should be able to avoid static obstacles while approaching to a randomly assigned target location within an unknown 3D environment.
- To implement dynamic obstacle avoidance, the UAV should be able to avoid the dynamic obstacle and plan its trajectory regarding the movement of the dynamic obstacle.
- To implement spatial awareness, the UAV should be able to avoid a restricted rectangular area defined by the user without any map data, it should exit the restricted area by searching the shortest path about the UAV's current position, target location, and the geometric shape of the restricted area.

## 1.4 Challenge faced

Autonomous navigation is described as the process of generating a map representation of the UAV's immediate working environment, detecting potential hazard relative to the UAV's movement, and navigating within the immediate working environment to a target location. It takes input information from both mapping and localisation system in conjunction with the cognition system to execute a mission [29]. For humans, the ability to navigate intentionally is imminent. For a mobile UAV, however, navigation is in dynamic real-world is an extraordinarily complex and challenging task. Such environments are characterised by their complex structure, the dynamics of both humans and moving objects, and the complexity of various flight mission. Modern UAVs aim at higher levels of autonomy and performing flight stabilisation [30]. A fully autonomous system should be able to gain sensory information from the local environment, classifying the types of objects that they detect, reasoning about the evolution of the environment, navigate itself to the target through its operating environment without cooperative communication from either ground control station or neighbouring aircrafts, obey the relevant rule of the airspace, such as maintaining as much as possible the planned trajectories with a minimum separation from other UAVs or obstacles in spite of uncertainties and unexpected situations [31]. An autonomous navigation system roughly includes the following five interrelated competences [29, 28, 32]:

- **Perception**: to obtain, represent and interpret sensory information to recognise objects, places, and events that occur in the or the UAV itself in the real world. In this way, the UAV can prevent damage, known where it is, know how the environment is.

- **Mapping**: to construct a spatial representation of an environment model with an appropriate sensing system. The map allows the UAV to make appropriate decisions and avoid damage.

- **Localisation**: the strategy to estimate the UAV's position within the spatial map

that occurs simultaneously during navigation, to assist the UAV plan and execute movements, and build a correct map of the environment.

- **Path planning**: the process of generating a trajectory towards a goal location from a specific starting point, without colliding to any obstacles detect from the mapping system, within a minimum distance, time or any other constraint required by the mission. Moreover, the UAV should also be able to avoid any dynamic obstacles that the mapping system failed to detect before the navigation task, such as extruding people, animals or any change of the environment.

- **Path execution**: to determine and adapt motor actions to environmental changes, ensure planned movement is executed, despite unexpected uncertainties (such as wind disturbance).

Due to the behaviour, the type of flight mission, and the complexity of the manoeuvres required, it is necessary to increase the level of robustness and accuracy for the autonomous navigation system. Designing such a robust autonomous navigation system in this complicated situation is accomplished by combining a variety of technologies from different disciplines and has posted a variety of challenges, the remainder of this section lists some of the critical challenges currently facing for designing a collisions free autonomous navigation system for UAV.

### 1.4.1 Perception

The degree of a successful autonomous navigation system is highly linked to their ability to perceive. Human beings use the combination of their five senses to perceive their environment, comprehend the situation by fusing their environmental perceptions with relevant contextual information and mission goals, and determine the best action by the predictions based on their perception and comprehension of the situation [33] [29]. To sum up, **Perception**, **Comprehension**, and **Prediction**.

For an autonomous UAV navigate at least on the same level of a human pilot, the aircraft should be able to recognise the working environment, classifying the type of object that they detect, reasoning about the evolution of the environment and planning complex motions that obey the relevant rule of the airspace [31]. An excellent sensory system is a crucial element to achieve this. Moreover, besides the imperfection of current sensor technology, there are some challenges to complete human-level situational awareness for autonomous systems.

- **Perception**: Situational awareness for UAV is typically defined with a particular mission, and it is vital for the UAV to determine the required information by reasoning the associate flight mission goals. This will result in UAVs ignore relevant percept as they are only programmed to detect or interpret only particular environmental aspect [33] [29].

- **Comprehension**: Human comprehends by reasoning the environmental with their goal and associated history information about this particular goal. A similar level of comprehension ability is a key element to achieve human-like UAV situational awareness, which requires that UAV integrate a large amount of sensory information and prioritise the perceived data with regards to flight mission. Current situational awareness system cannot comprehend the same level as a human pilot as they rely upon their supervisors to prioritise the importance and meaning of sensory information received [33] [29].

- **Prediction**: Human makes predictions by perceiving and comprehending the particular situation. Consider the example of human try to cross the street. Firstly, people would start by observing both directions of the street for any in-coming traffic, in the case of approaching vehicle, they will try to estimate the time required for the vehicle to reach the point where they stand, and then decide to cross the street if the estimated time to cross the street is shorter than the previous estimation. Currently implemented systems have only limited prediction ability.

For an autonomous navigation system, a difficult situation may be indistinguishable to the UAV, and some problems can occur when the UAV has only limited spatial awareness. Therefore, it is essential for the UAV to choose the appropriate sensor based on the given function required, process the sensor data with noise, determined the necessary trade-off between safety and efficiency, and be prepared for any other uncertainties. Finn summarised that the sensors might be roughly divided into two classes, 'proprioceptive' sensor and 'exteroceptive' sensor [29].

Proprioceptive sensors (i.e. radar, laser, IMU), provide the internal information about the UAV's motion, may measure cursing speed (odometer), acceleration (accelerometer), rotation angle (gyroscope), steering angle, ground level, and provide the UAV's position and orientation data with respect to some absolute frame of reference [34] [35] [36]. They provide measurements of UAV's internal factors that are affected both by the environment and UAV's behaviour.

While exteroceptive sensors (i.e. LADAR, EO and IR) are responsible for recognising a place or a situation or be converted to map representation of the environment, they perceive external factors that are not under the control of the UAV [29].

In most case, the sensor reading is imprecise and unreliable due to the noise, or any system error from the sensory system, compared to human's comprehensive sensing abilities [33]. For this thesis particularly, a comprehensive UAVs perception system is necessary for the following reasons:

- To estimate the UAV's current position and orientation.
- To estimate the UAV's velocity and acceleration.
- To estimate static obstacle's position within the navigation environment.
- Identify any moving obstacle and predict its moving trajectory.

### 1.4.2 Mapping

Mapping is crucial to autonomous navigation for several reasons: (1) to localise the UAV itself and the target location; (2) to plan a path between the UAV's current position and the target location; (3) to support UAV's collision avoidance. Besides providing an intuitive visualisation of the explored space, it allows further analysis which assists the development of other types of high-level navigation tasks, such as dimension evaluation, object recognition, etc. Proper mapping will improve the accuracy and robustness of localisation [37].

For certain types of navigation task, it is sufficient for the navigation system to get access to the map information before the mission, such as warehouse sorting UAV illustrated in figure 1.5. However, it will be impractical and tedious to build a full map to navigate in a large and unknown environment such as goods delivery. Furthermore, the environment is not always static, and this can be due to objects that either change position or its shape in time. As a result, the UAV must create its own map. Interestingly, the map building process requires knowing the UAV's location, and the location estimation requires a map, which led to another critical aspect: Simultaneous Localisation And Mapping (SLAM), which raised two challenges. The first one is the map errors relative to the real environment due to the limited range of observation or imperfection of the sensory system. The latter is the dynamic environment modelling due to the change in the object's position or shape.

The mapping system should be able to distinguish between the free and occupied space of the immediate working environment; Identify the moving obstacles and keep a record their historical trajectories; update the map information to keep track of new entered obstacles; and if possible, predicting the moving obstacles' future position from the recorded historical movement.

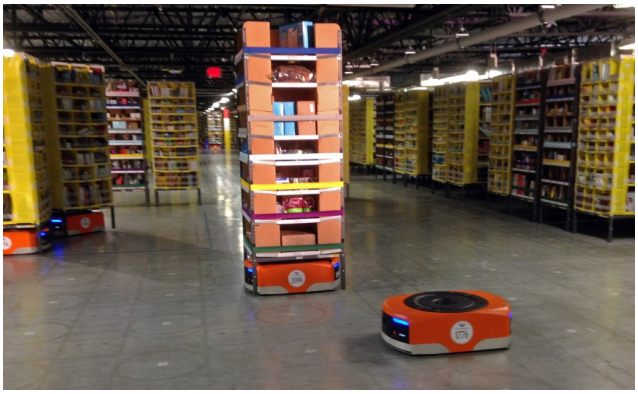


Figure 1.5: Warehouse sorting robot [2]

### 1.4.3 Localisation

Autonomous UAV navigation systems need a robust localisation system to avoid catastrophic control actions; it is the process of determining the UAV's position, velocity and orientation information relative to a reference frame, without any prior external information to the localisation system, except for the instantaneous sensing of the working environment [38].

Global Positioning System (GPS) is the most common outdoor localisation solution for navigation systems to estimate their position [39, 30]. It provides geological location and time information to a GPS receiver device anywhere on or near the earth. High-end GPS receiver could reach the accuracy within a few centimetres, and millimetre accuracy level if it is in a long-term measurement, but this is expensive to be used in civil applications. Furthermore, as it is stated in the US government information about the Global Positioning System [40], The committed accuracy of the broadcasted GPS signal in space only has a global average User Range Error (URE) of 7.8m within 95% probability. Moreover, URE is not user accuracy, and the actual received signal is additionally depended on server factors including satellite geometry, signal blockage, atmospheric conditions, and receiver design features (quality). The accuracy is worse if the device is not in an open sky, such as indoor usage, or the device is near a bridge or trees, which will cause uncertainty and a potential hazard if the UAV localisation system is solely depended on GPS.

Therefore, an additional localisation solution is preferable when the UAV is undertaking tasks in these scenarios.

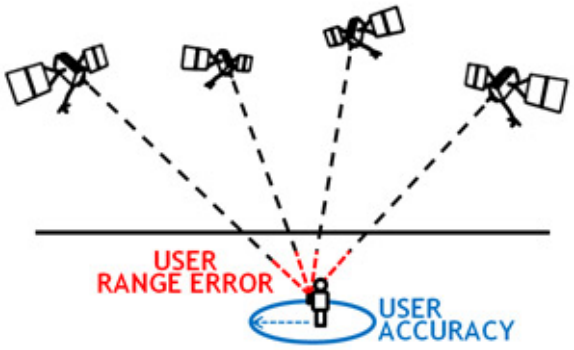



Figure 1.6: URE and actual user accuracy [3]

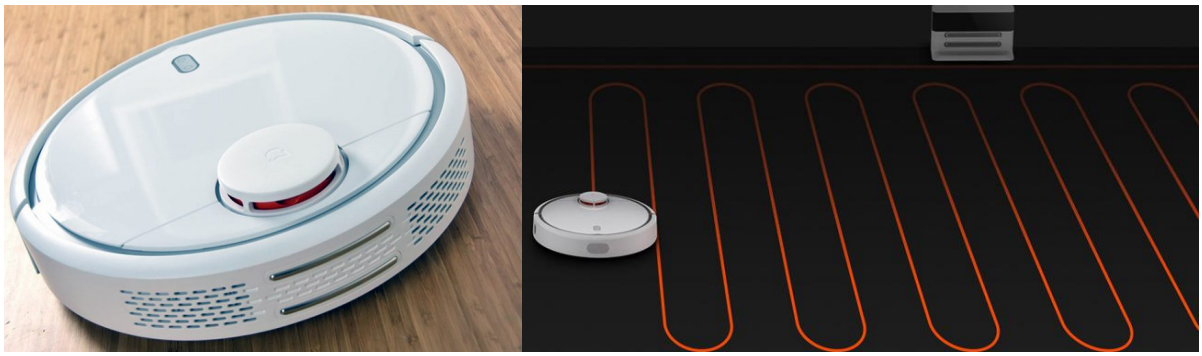
### 1.4.4 Path planning

Path planning refers to the process of determining a sequence of waypoints from a specific starting position to a target location while avoiding obstacles and impediments detected from the mapping system to minimising (or eliminating) potential hazards to the surrounding objects or the UAV itself [29]. It is one of the essential responsibilities for an autonomous system, random movement, which does not have a specific navigation plan, may be useful for a particular task, such as mobile cleaning robot, surveillance and exploration robot [41]. However, for most of the scientific or industrial autonomous navigation system, it is almost pointless for the UAV to move without a purposeful manner. Hence, path planning plays a vital role in the success of the autonomous UAV navigation system, and also serves as a baseline for the related applications of an autonomous navigation system.

To be more specific, the planner should be able to make the appropriate decision by predicting action into the future based on the characteristics of the UAV, its current and potential behaviour, the goal of the flight mission, with respect to the environment. Then evaluate the possible outcomes according to a cost function with selected criteria [42] [29] [43]. The evaluation function should bring the optimised high value for plans that meet

mission goals by representing the UAVs' objectives and constraints [29]; this normally involves some form of search through a set of potential plans until the optimised one is found.

For this thesis mainly, the path planning system should be able to find the optimal flight trajectory to the target location in a dynamic and complex environment, in a way that the trajectory avoids or minimises the hostile threat, while subject to system constraint (such as limited time, fuel, etc.) and satisfies the task requirement. Moreover, apart from avoiding obstacles that may cause physical damage to the UAV (**obstacle avoidance**), The UAV should also be able to avoid any dynamic obstacles that the perception system failed to detect at the first place (**collision avoidance**). Lastly, the path planning system should also avoid some predefined region that will not physically cause damage to the UAV, such as military restricted fly zone or any area that UAVs are prohibited.



(a) Mobile cleaning robot [44] (b) Real time planned path by the cleaning robot [44]

Figure 1.7: Example of path planning for autonomous navigation system

### 1.4.5 UAV control

In building autonomous UAV systems, the navigation system must implement means of controlling the vehicle in an uncertain environment throughout the mission [45]. It involves the process of combining the outputs of perception, mapping, localisation, path planning, and collision avoidance and translating them into actuator commands for UAVs' mobility and payload response [29]. Therefore, the UAV should have the capability of self-governance in the performance of control function in a very unstructured environ-

ment. Autonomous UAV control is described as a higher level of autonomy, with regards to detect and respond to anticipated events and condition, compared to automatic control, which only provides the necessary standard control operation. It should act as a replacement of a human pilot to ensure robust and constant operation for UAV within an unfavourable working environment for an extensive flight duration [46].

The specific behaviours will vary with the operational requirement. In this thesis, the UAV should be able to perform the essential flight operation such as taxi, take-off, climb, cruise, glide, landing, etc. Besides, the UAV is also required to have the ability to perform actions required by the particular flight mission, such as manoeuvre, reconnaissance, combat, health condition monitoring, payload management, etc. [46]. The control methods for these action varies depending on the vehicle's dynamics of each type of UAV; different control methods need to be applied for a fixed-wing and rotary-wing UAV. It depends on the task of determining the type of the UAV, and different control methods need to be investigated for that specific type of UAV.

### 1.4.6 Simulation framework

To test algorithms on real UAVs, it cost researchers both financially and timely to get access to expensive hardware and to train safety-pilots. Furthermore, most of the navigation data occurring on real UAVs, are hard to replicate, and often cause harm or destruction to the UAV, which brings up the revolution to simulate the behaviours of the UAV navigation system. Craighead summarises the following qualities on an excellent simulation framework [47].

- Variety of hardware that can be simulated (both sensors and UAVs)
- Graphical simulation accuracy
- Physical simulation accuracy
- Cross-platform capabilities

- Openness of source code (for future development and addition of new or custom simulations by the user).

A well-implemented simulation system facilitates the development of the navigation system, but it also raises the question of how well the simulation system represents in the real world. Ideally, the simulated navigation system should be migrated to a real UAV with minimum efforts. The simulation framework should be a tool that enables the development of algorithms, to be deployed on a real UAV later on.

### 1.4.7 Conclusions

There are still numerous system constraints for developing such a robust autonomous UAV navigation system, such as UAVs physical constraint (speed and acceleration); system integration (hardware and software); system uncertainty (system error and malfunction); environment uncertainty (e.g. air turbulence and other extreme weather conditions); flying efficiency (time and energy); limited onboard computational capability. Apart from the technical challenges, problems like government regulation on the UAV operation, public privacy concern will also prevent the commercial deployment of UAV related applications. This thesis will focus on the technical challenges faced to both designs and demonstrate autonomous navigation algorithms for UAV, featuring collision-free navigation and spatial awareness. The necessary components to simulate autonomous UAV navigation system are concluded as follows :

- A comprehensive simulation framework, to simulate UAV and environment dynamics, ideally, the system should be analogue to its real-world counterpart and easy to be transferred to a real UAV.

- Mapping system, the system deals with the perception and mapping challenges, it should transfer the input data from perception units into a map representation, where the map should contain sufficient information about the obstacles location and size with respect to the working environment and UAV itself.

- Localisation system, the system deals with the perception and localisation challenge; it should retrieve an accurate estimation of the UAVs position and orientation within the world coordinate system. This system complies with the spatial awareness objective identified in Chapter 1.3.

- Cognition system, the cognition system should make an appropriate decision based on the input information from mapping and localisation systems, then make the appropriate command to navigate the UAV to the target location by avoiding any potential collision; it should comply with the static/dynamic obstacle avoidance objectives identified in Chapter 1.3.

- Control system, the control system should execute the decision made from the cognition system, to drive the UAV from point A to point B with a desired orientation.

# Chapter 2

# Literature review

In this thesis, autonomous navigation can be described as the process of determining a sequence of waypoints between a start position and a target location, whereby executing the sequence of waypoint the UAV can avoid the potential collision with its surrounding obstacles and reach to the target location with a speedy manner. This process mostly replies the UAV's knowledge of both its own and obstacles position concerning the working environment.

In order to complete the scheduled flight mission, as it is described in Chapter 1.4.1, the autonomous navigation system requires inputs from proprioceptive and exteroceptive sensors for the SLAM process. Where the proprioceptive sensors are responsible for the UAV's awareness of its internal states, such as position, orientation, navigation velocity, as well as the start and target position (**localisation**). On the other hand, exteroceptive sensors are responsible for the process of observing the UAV's immediate working environment at a given time or period, then display the environment in a map representation (**mapping**). With the inputs from both proprioceptive and exteroceptive sensors, after internal processing of localisation and mapping, the cognition should produce a sequence of waypoints for the UAV to reach the target location, where the waypoints should eliminate (or minimise) potential collision with the detected obstacles from the mapping system. Finally, the control system takes inputs from the cognition system and drive the UAV to

its target position.

This chapter includes survey work for SLAM techniques and path planning algorithms. Various perception methods and path planning algorithms are evaluated based on different performance criteria.

## 2.1 SLAM

During an autonomous flight, UAVs are required to equipped with the necessary information to avoid obstacles and reach the target location. However, it cannot be assumed to have an accurate map for all potential environment before the flight. Additionally, as the world is continuously changing, the UAV can enter a previously unmapped area. So to ensure safe operation, autonomous UAVs must be able to use onboard sensors to construct highly accurate maps of their observed environments.

In the early development stages, mapping and localisation are considered to be separately, but later researchers found that they are often considered as ”chicken-and-egg” problem [48, 37]. They are interdependent, as each of them can be solved if the other one is known accurately [37]. For example, It is relatively easy to locate and identify a UAVs pose in a known environmental map, and it is simple to map an environment given a UAVs pose. This concept refers to the Simultaneous Localisation And Mapping (SLAM). It constructs a map of the environment while localising the UAV on this map.

However, simultaneously estimating the map and localising itself relative to the map is a far more complicated process. It involves the integration of measurements from a variety of sensor including the IMU, GPS, laser scanner, or vision-based sensor. As each sensor has unique characteristics that lead to varying levels of effectiveness in different environments [48]. This section includes the survey work for how these sensors affect the UAV’s perception and localisation ability and the performance in terms of collision avoidance and efficiency by different path planning algorithms, for an accurate and robust autonomous navigation system.

### 2.1.1 State estimation

The State estimation problem is a fundamental issue when dealing with autonomous UAV navigation. Knowing its position in the environment enables better awareness and more precise navigation to avoid catastrophic control actions. It involves the process of estimating the UAV's state (position and orientation, linear velocity, angular velocity and acceleration) during the movement, from either single sensor reading or fusing a different number of sensors [30]. State estimation can be decomposed into three slightly different problems which differ on the UAV's knowledge of the environment both initially and during its movement [49, 50].

- **Local Localisation** addresses local uncertainty of the UAV's configuration, such as the initial position relative to an external coordinate frame. The local position will then be tracked over time.
- **Global Localisation** specifies the UAV's unknown position in the environment, such as buildings, trees and objects of interest, is regarded as one of the most difficult state estimations due to the high dimensionality of the parameter spaces.
- **Kidnapping** regards to the problem of determining the change of UAV's position over time, including UAV itself and any other objects within the environment. This problem is like the Global Localisation, with the added difficulty of changing locations over time.

Generally, state estimation can be divided into the following two parts: the Attitude Heading and Reference System (AHRS) the Inertial Navigation System (INS). As shown in Figure 2.1. AHRS predicts the UAV's orientation by receiving inputs of Euler angles or rate. Then the estimated orientation data are processed in conjunction with other sensor information to estimate the UAV's position, velocity and acceleration [4]. Different approaches are proposed to solve the problem. Since sensor characters determine the system architecture, it is necessary to understand the characteristics of various sensors used for state estimation. In general, state estimation techniques are categorised as follows [4]:

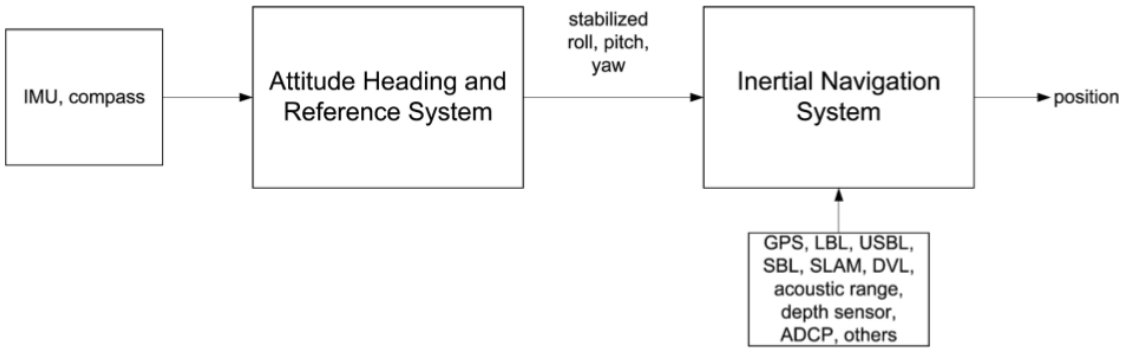



Figure 2.1: UAV state estimation with AHRS and INS [4]

- **Acoustic**: refers to the techniques based on the measurement of Time of Flight (ToF) of certain types of signals such as acoustic.
- **Geophysical**: refers to the techniques based on external reference information to estimate position. This types of techniques require the implemented sensors are capable of detecting and classifying the perceived environment based on specific features.
- **Inertial/dead reckoning**: refers to the techniques of estimating state with accelerometers and gyroscopes.

Acoustic based techniques estimate the velocity vector of a UAV moving across the environment. It determines the UAV's position by measuring the ToF to a reference point in the environment, in conjunction with the velocity measurement by transmitting acoustic pulse and measuring the Doppler-shifted returns from these pulses. A popular way to implemented geophysical-based techniques is to use visual sensors; it is based on the information provided by the visual sensors of the cameras. Different approaches have been proposed. Major difference between these approaches are the type of the visual information used; such as horizons detection [51], landmarks tracking [52], or edges detection [53].

Both Acoustic and Geophysical-based methods rely on the UAV's knowledge of the map, to estimate a location relative to an object or feature within the reference frame,

which is highly depended on the UAV's ability of environment perception. Survey work for Acoustic and Geophysical-based perception methods are given in Chapter 2.1.2. On the other hand, researchers have proposed various methods to determine the AHRS with less environment depended on techniques (dead reckoning), such as implementing pressure sensor to measure UAV's attitude, using a magnetic compass to shows UAV's direction relative to the geographic direction [54].

Although these techniques are capable of measuring the UAV's position within a global reference, there is still a lack of information about the UAV's orientation. IMU (Inertial Measurement Unit) has been used extensively to solve this. It is an Inertial/dead reckoning-based device to estimate UAV's position, orientation, velocity (both linear and angular) and gravitational force without the need for external reference. It consists of an accelerometer and a gyroscope. The accelerometer is responsible for the estimation of the orientation UAV with respect to the earth, by measuring the force required to accelerate a proof mass. On the other hand, gyroscope was responsible for the measurement of UAV's angular rotation rate around an axis, and The integration will result in a drift in the estimated Euler angle. Additionally, the slight measurement errors can compound in significant errors with the accumulation of navigation period.

The performance of state estimation techniques is highly reliant on the type of flight mission, and the performance can be increased by the fusion of data from multiple collaborative sensors [55, 34, 56]. The most important factors are the interest regions of the working environment and the desired localisation accuracy [4].

### 2.1.2 Environment perception

When discussing the mapping of the environment, it is useful to review the type of environment perception sensors used for the currently deployed UAV navigation system. Generally, the sensors used for 3D environment perception can be categorised into **passive** and **active** systems [57, 58, 11]. Where passive sensors provide image data which can be later processed with certain algorithms to extracts 3D information from the measured

2D images (such as a vision-based sensor), on the other hand, active sensors are capable of providing 3D information of the environment directly from the measurements (such as LiDAR). This chapter includes survey work done for the recent popular sensor solutions include vision-based, LiDAR and radar sensors.

**Vision-based sensors**

Vision-based sensors can be used to detect information about the immediate working environment with a specific interest range of the region. The information can be then extracted with image processing algorithms for stabilisation, navigation and further information collectoin [59]. For example, determine the distance to particular objects in the 2D image by using complex processing algorithms [60]; classifying the type of objects by running feature detector and extractor [61]; distinguish motion in a scene by using optical flow technique [62]; Exploit 3D reconstruction method for navigation and mapping [63]. Vision-based sensors play a vital role and prove to be a primary direction for autonomous UAV navigation [62, 30, 64]. Firstly, they can provide sufficient information about the UAV's immediate working environment. Secondly, they are suitable for the perception of dynamic obstacles due to their high anti-interference ability [64]. Lastly, vision-based sensors have the advantage of lightweight, low power consumption and relatively low cost [30].

However, there are a few challenges to apply vision-based sensor for autonomous UAV navigation [11, 65]. The first challenge is due to the characteristic of vision-based sensors could only detect the working environment with a specific angle, which means the sensors are required to deliver the detected obstacles with some additional information, where the additional information could be the size of the obstacle, the distance from the sensor to the obstacle, or angle information at different position during the flight mission. The second challenge is that the intensity of light has a significant impact on the sensors' sensitivity. The captured scene contains different information about the environment with the change of illumination. It poses a great difficulty when the UAV is working in an

environment with frequent change of light since the perception can only extract a limited feature with weak illumination. Furthermore, the performance of vision-based sensors can be affected by several other factors, such as the resolution of the image, capturing time, different structures of the obstacles from the environment.

### LiDAR

LiDAR (Light Detection and Ranging) refers to the remote sensing techniques which utilise light in the form of a pulsed laser to directly measure ranges to different points on the surface of the object, then construct the obstacle's 3D geometry in conjunction with the various measurements. It provides 3D digital representations (such as point cloud) within a given Field of View (FoV) [58]. LiDARs can be either 2D or 3D. In order to achieve a 3D representation of the environment, as a laser beam is highly concentrated, either the laser scanner needs to change its viewing angle periodically, or multiple laser beams are positioned from different angle together so that they cover the given part of a 3D object.

The performance of LiDAR is heavily influenced by the range parameter, which defines the maximum distance they can detect, and determines the distance to objects. The range is profoundly affected by the reflectivity of the obstacles' surface, as well as weather conditions (such as humidity). Despite those disadvantages, LiDar sensors are capable of measuring the direction and distance to an object directly, while almost other types of sensors require significant processing of the perceived environment data.

### Radar

Radar technology is frequently used to measure obstacles' distance and velocity. Similarly to the acoustic-based localisation techniques, it measures the obstacle' distance by emitting a radio wave which travels at the speed of light, then calculate the distance between the radar and the obstacle with the measured ToF. On the other hand, obstacles' velocity is measured by detecting the change of frequency or wavelength with respect to

the obstacle, which is moving relative to the radar device caused by the Doppler Effect, whereby the obstacle's velocity is proportional to the returned signal.

Due to the fundamental principle of how radar-based sensor works, it emits broad wave and measure by the reflection of the wave bouncing back from objects. Radar waves are by nature physically "wide", which means the device will also receive feedback from other objects rather than the particular object that is being aimed at, that makes it relatively less preferable if the UAV is working in a crowded environment.

The main advantage of radar-based sensors is that they are less affected by weather and lightning condition compared to vision-based sensors and LiDAR. Another advantage of radars is that they are capable of providing a direct measurement of the obstacle's relative velocity, which is essential for decision making in autonomous UAV navigation, especially in the situation of dynamic obstacle avoidance. For these reasons, radar-based perception is increasingly employed in collision avoidance and obstacle detection for autonomous UAV navigation [66, 67].

**Summary**

This chapter listed the survey work for the types of sensor used for environment perception, which includes vision-based sensor, LiDAR and radar. The vision-based sensor can provide abundant information about the environment, which makes it suitable for higher-level navigation tasks. As it is passive sensor, it requires rather complicated image processing algorithms to extract the useful information for specific flight mission, and the performance relies on the particular algorithm. LiDAR and radar are active sensors, they may generally provide better observation capabilities, as the sensor can be more dedicated and targeted towards the remote sensing objectives and, in addition, they may depend less on the environment circumstances compared to vision-based sensors [57]. The main disadvantage of LiDAR and radar are their high operational cost and limitation when operating in a crowded environment.

Based on the above considerations, as well as by the results in [11],

Table 2.1: Sensor comparison ( [11])

| **Criteria** | LiDAR | Radar | Vision-based |
|---|---|---|---|
| short range | very good | very good | good |
| long range | medium | very good | poor |
| velocity measurement | No | Yes | No |
| weather conditions | poor | very good | poor |
| night condition | very good | very good | limited |

Table 2.1 illustrated the comparison for the surveyed types of sensors for environment perception. As it is illustrated in the table, as all these properties are not often found in a single technique, so to give a more accurate estimate of the world the UAVs, Researchers have proposed different solutions for better environment perception includes cooperative perception [68, 31] and multi-sensor fusion [69, 70]. Cooperative perception refers to the process of multiple UAVs sharing perceived environment information, and multi-sensory fusion refers to the concept of sharing the perceived environment from the same navigation device. Each of these solutions requires the phase of the integration process of combining (or fusion) different source of sensory information into a single observation. The advantage of multi-sensor fusion is the reduction of uncertainty, rejection of noise, toleration of sensor failure, increase in resolution and the extension of the sensor coverage, which enables more accurate and robust estimation of the environment [64, 71]. Popular fusion algorithms include Kalman Filter, Extended Kalman Filter (EFK) and Unscented Kalman Filter (UFK) [69, 72].

### 2.1.3 Navigation techniques

Successful autonomous navigation relies on the UAV's spatial awareness ability, to localise itself and be aware of the dynamic situation within the working environment. Either through a pre-loaded map or constructing a new map during the flight mission. Both ways require the UAV to obtain an accurate mapping system; it refers to the process of integrating the perceived environment from sensory system and represents them in a spatial

model. This process can be described by the question ”What does the environment look line in which to operate”, central issues relate to environmental representation and interpretation of data from sensors. It plays an essential role in navigation for three principal reasons. Firstly, a comprehensive map representation assists high-level tasks such as path planning and obstacle avoidance, which are fundamental requirements of autonomous UAV navigation. Secondly, the map provides an intuitive visualisation of the UAV's immediate surrounding, which allows further analysis of the explored space (such as object recognition). Thirdly, the proper mapping improves the precision and robustness for the navigation task. This section includes the survey work for different mapping techniques used for UAV navigation.

**Map-using-based navigation**

Map-using based methods rely on UAV's prior knowledge of the working environment before navigation. Those maps may contain different level of details, ranging from a complete 3D CAD model of the surrounded environment to a simple graph of interconnections between the key elements from the environment. The main idea for map-based navigation is to represent the environment with a sequence of landmarks which are expected to be encountered during navigation, the task then is to search and identify the landmarks observed in an image. Once they are identified, the UAV can use the provided map to estimate the UAV's position (self-localisation) by matching the observation against the predefined landmark description in the database. The process can be divided into four steps [73]:

1. Acquire sensory information.

2. Detect landmarks, extracting edges, smoothing, filtering, and segmenting regions.

3. Matching, identify observation and compare with the database for possible matches according to defined criteria.

4. Localisation, calculate the UAV's location from the observed landmarks and the database.

Map-based navigation system predefines the spatial layout of the environment in a map, which enables the UAV to navigate with detour behaviour and movement planning ability [64]. One of the key classification criteria in this approach depends on the level of details provided by the map [74]. The primary challenge for this method is the matching criteria; it requires a sufficient map database which is generally constrained by prior landmark knowledge and will cause a significant effect on the UAV's localisation. Atiya used a method based on online image recognition [75]; the extracted features stay invariant in regards to UAV's movement. This method is sufficient for the UAV to match online images with the environment for the self-locating step. Landmark tracking algorithms determine the UAV's position, and extract landmarks from the live image and re-organised them into consecutive scenes. The landmark can be natural features from the environment or artificial ones defined by the user. In both cases, the UAV needs to be aware of the extracted features of the landmarks in order to track them.

A well-known example of this method is the view sequenced route proposed by Matsumoto [76]. It finds the route by extracting features from a sequence of images, and then use a matching algorithm to self-localise. During the navigation, the UAV will first construct a sequence of images, then associate each image with a motion to a corresponding target location where the sequence of motion forms the trajectory. Another technique derived from this is the object recognition which comprehends the environment instead of memorising. It provides the ability to recognise command objects such as desk and doors which can be used as a landmark, then represents objects by categorising their functions which are described by the objects' surface  for example, a desk is characterised by a work surface and some surfaces that correspond to the support structure (legs). In this case, the UAV utilises symbolic commands such as "go to the corner", which advise UAV the landmark is a corner and the path points straight ahead. The UAV builds a 2D map which stored the projections of the observed landmark. After the location of the landmark

is projected into the map, the UAV will then generate a trajectory and employs odometry to reach the target.

### Map-building-based navigation

Map-Using-based approach requires the UAV to obtain prior information about the environment before navigation. However, it is not always easy to generate a map, especially with correct metrical information. Furthermore, most available maps come on external storage devices are costly expensive and by far more extensive than the onboard memory storage available. Moreover, most existing maps are also very likely to be inaccurate and contain systematic errors which could reduce the reliability and jeopardise its safe operation [77]. Therefore, many researchers have proposed the method for the UAV to explore the working environment and construct the map simultaneously.

Moravec was the first using map-building approach [78]. This method uses an interest operator to extract images characteristics. These features were then correlated to generate their 3D coordinate. The environment was represented occupancy grid with two squared meter cell, along with coordinate information of the extracted features. Although this approach can provide obstacles' position within the environment, it did not represent the environment with a meaningful model. Occupancy grid-based strategies usually require a significant amount of computational capacity and cause delays if the UAV is working within a complicated environment [36].

As an alternative, topological representation has been used extensively for map-based navigation. The map is simplified with only vital information remains, such as nodes linked by links where nodes represented the most dominant characteristic of the working environment, and links represent direction, time and other relationships between the nodes an excellent example of the topological map the London underground map. However, topological maps lack some details such as scale, a distance which could potentially cause efficient navigation for the UAV. Thrun went one step further with a remarkable contribution; He proposed a method by integrating the best of occupancy grids and topo-

logical maps for navigation [79]. The system first constructs an occupancy map with neural networks and Bayesian integration technique, where the occupancy map is then transformed into a topological map. Hornung has proposed a solution of representing 3D environment models by classifying the occupied, free and unknown space [8]. The environment is described with octree representation, where the octree contains eight children segmentation, which allows the system to store and update the 3D models efficiently.

### Mapless navigation

Map-less navigation refers to the navigation system that does not need knowledge of the environment to operate. It operates by observing and extract elements from the environment, such as optical flow; sonar; object features; appearance; object recognition; or any other vital environmental features can be used to navigate. It is not required for the UAV to obtain the absolute (event relative) position of those objects within the environment. However, the navigation process is constrained by those elements.

**Optical-flow-based method** As an animal or a UAV moves, the image of the visual environment moves on its retina, the changing pattern of apparent motion between the observer and the scene is called optical flow. This optical flow contains several vital sources of information [80]. For example, a car's position can be estimated by analysing the associated camera images, once the position is estimated, the relative motion can be calculated by comparing the according to image at a different frame. Hultqvist proposed a solution to detecting and localising overtaking vehicles by using 1D optical flow [62]. It works by tracking and evaluating the optical parallel to the vehicle's motion, and then estimate its motion based on those features. The system was concluded to be capable of detecting overtaking vehicles and their dynamic position relative to the sensor itself. Santos-Victor et al. have developed an optical-flow-based system to emulate the flying behaviour of the bee [81]. The system contains two cameras mounted on each side of the UAV and measures velocities by calculating the differences in optical flow perceived from

each side. While the UAV is moving along the centre of a corridor, the UAV continues if both velocities are equal, and moves to the side with slower velocity if there is any difference. The limitation of this approach is that the walls need to be well textured for the system to perceive enough optical flow data.

**Apperance based method** This approach is based on recognising the previously recorded image, the UAV then self-localise and operate by comparing current received image with old ones. The process can be divided into two steps [74]. Firstly, extract prominent features from the environment and stored them as templates where the templates contain specific position information, which is then linked with the appropriate control command. Secondly, during the navigation process, the UAV has to recognise the environment and self-localise by matching the current image with the stored template. Zhou et al. proposed an appearance-based method by utilising multi-dimensional histograms [82]. The histogram contains various features extracted from a pre-recorded image, which consists of colour, texture, gradient and edge density. The matching and self-localising are achieved by constructing a new multi-dimensional histogram from the current image and compare them with the stored one. Storing those feature in a histogram bring two main advantages: it requires less computation capacity, and the correlation process is more efficient than full images.

**Object recognition based method** For the previously discussed appearance-based approaches, in most cases, the UAV reaches its target by only having limited sequences of images or a few pre-defined images of target goal that it can use to track and pursue. Kim and Nevatia have proposed an alternative method which utilises symbolic navigation approach instead of the appearance-based method to self-localise by memorising the environment [83] [84]. For this method, the UAV manoeuvres itself by taking symbolic commands such as ”go to the door on the left” or ”take the main exit” etc. The system then extracts information from those commands to recognise the environment and the trajectory it needs to follow to reach the target. For example, a command such as ”go to

the corner behind you" contains the information that the landmark is a corner, and the trajectory command is going to the opposite direction the UAV is facing. The UAV builds an occupancy grid map called a *squeezed 3D space into 2D space map* or *s-map*. Which is a 2D projection of landmarks as they are recognised. After the location of the target landmark is transferred to the *s-map*, the UAV calculates the trajectory and self-localising with a GPS-like path planner and employs odometry to approach the target.

**Conclusions**

This section discussed different mapping techniques used for the navigation system. The reviewed solutions are classified into three categories: mapping-using, map-building and mapless. The primary challenge for this map-using-based method is to find an appropriate algorithm for environment representation and define the dynamic matching criteria. For the map-building system, the UAV build the whole map by itself and simultaneously self-localise within the environment during map construction. An efficient map building system should have the appropriate combination of the sensory system to perceive sufficient data from the environment in the navigation process and build a map in regards to obstacles that need be avoided and safe zones for flight [36]. Map-less navigation refers to the UAVs that operate without environment information. The main limitation for current mapless navigation is that there is no global environment representation, and most of the techniques can only be achieved in an indoor environment [85] [36].

## 2.2 Path planning

This section includes the survey work has been taken out for path planning algorithms. A comparison table is given at the end the section includes the performance of various path planning algorithm concerning their operational and computational aspects.

### 2.2.1 A star algorithm

$A^*$ can be viewed as an extension of Dijkstra algorithm; it was first published in 1968 and corrected in 1972 by Harte [42] [86], it is a tree search algorithm that guarantees to find the existed optimal path from a given initial node to a goal nod. It reduces the pathfinding problem to a graph searching by fitting the graph into the space, then uses a heuristic function $h(n)$ to estimate the lowest cost path from the start point to the goal point in addition with the local path cost $g(n)$, therefore to construct a total cost function $f(n) = h(n) + g(n)$. The algorithm finds the optimal path by continually updating the lowest cost path and propagate path data to the successor node. The algorithm's completeness, the time complexity is highly related to the heuristic function because it determines which node to be expanded nextly, larger estimates usually result in fewer nodes being expanded. Thus, the trade-off between speed and accuracy can be used to adjust the algorithm according to the system requirement. However, $A*$ will only be admissible if it provides a lower bound to the shortest path distance and it is consistent, which is not always the case in practice since then it is possible for the heuristic function to overestimate the distance to the goal.

### 2.2.2 Voronoi diagram

Voronoi diagram is constructed from the known threat, and it partitions a plane into subplanes based on the distance to points in a specific subset of the plane. The set of points is called *Voronoi seed*[1] which is specified in advance. It constructs a corresponding region where all consisting points within the region are closer to the seed than any others; this region is called *Voronoi cell*. The boundary between each Voronoi cell forms a set of lines that are called *Voronoi edge*, where all points on edge are equidistant from the corresponding Voronoi seeds. An example of a Voronoi diagram with a finite set of points $\{p_1, ...., p_n\}$ in Euclidean plane is given in Figure 2.2. Where the dot represents the seed, and its corresponding Voronoi cell consists of all the points where their distance to the site

[1] also called Voronoi site or Voronoi generator

is less or equal to the length to any other sites. Each cell is obtained from the intersection of half-space, and hence it is a convex polygon where the line segments of the VD are all the points in the plane that have the same distance to their adjacent seeds. The *vertice* is the point of having the same distance to three or more seeds [5].

The properties of the VD make it an accessible mechanism for constructing a roadmap for path planning algorithms [87, 88], to use the obstacles' location as Voronoi seed and the Voronoi edge as the flight path which maximises the UAVs' distance from the closest obstacle. However, the experiment result showed the VD roadmap algorithm complete with information beforehand on a 2D map, but it has some limitation within a dynamic environment [88].

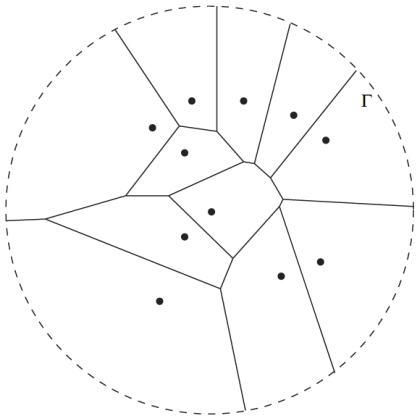



Figure 2.2: A Voronoi diagram of 11 points in the Euclidean plane [5]

### 2.2.3 State-space-based algorithm

For a UAV within working airspace, the dimension of the system is determined by the dynamic characteristics of the vehicle, which consists of velocity, position, orientation and angular rate. Each of those characteristics can be divided into sub-elements along x, y, z axis, therefore for a total of 12 variable, which is known as the state space [89]. State space-based algorithms work by decomposing the vehicle's state space into grids, and search for the optimal path satisfies system requirement (such as safety, time-efficient, etc.) within the grid. However, it is sophisticated to solve a real-time implementation due

to the high dimensionality of the state space; many implemented algorithms may reduce the set of states to represent the UAV's physical dynamics characteristics.

For example, Yu [6] proposed a rolling optimisation algorithm based on reduced state space. In order to solve a 2D example of state space, the turning angle for each UAV is delimited to 45° when flying horizontally. As it is shown in the right part of Figure 2.3, it is not possible for the UAV to reach the red marks for the next move, there are three green marks achievable for each UAV, hence $3^N$ states for N UAVs, which constructs the possible future states for the next interval.

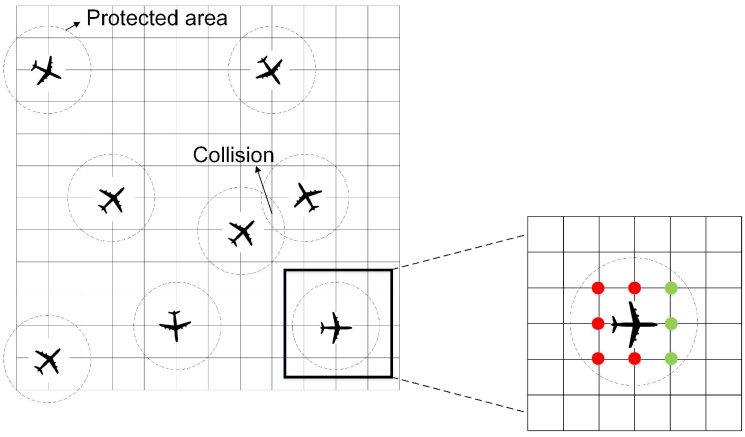



Figure 2.3: Top view for a collision case [6]

### 2.2.4 Geometric based algorithm

Geometric based approach involves generating the position and orientation trajectories in Cartesian space, This types of algorithms solve the problem by purely pursuing geometric law and it is the most flexible types of algorithm surveyed in this thesis, as the algorithm properties such as completeness, optimality, and etc. are totally depended on the shape of the Cartesian spatial curves. An example algorithm is the point of closest approach [7].

Figure 2.4 shows the geometry for two UAVs flying in a 2D horizontal plane, to analyse the conflict detection, the miss distance vector $\vec{r_m}$ is defined [90]:

$$\vec{r_m} = \hat{c} \times (\vec{r} \times \hat{c}) \tag{2.1}$$

Where $\vec{r_m}$ is the relative distance vector from UAV 'B' to UAV 'A' and $\hat{c}$ is the unit vector of $\vec{c}$ of the relative velocity vector $\vec{c}$. The miss vector $\vec{r_m}$ and relative motion vector $\vec{c}$ are orthogonal:

$$\vec{r_m} \times \vec{c} = 0 \tag{2.2}$$

The Point of Closest Approach can be derived with the relation between $\vec{r_m}$ and $\vec{r}$:

$$\vec{r_m} = \vec{r} + \vec{c} \cdot \tau \tag{2.3}$$

The Time of Closest Approach $\tau$ can be derived with equation ( 2.2) and ( 2.3):

$$\tau = -\frac{\vec{r} \cdot \vec{c}}{\vec{c} \cdot \vec{c}} \tag{2.4}$$

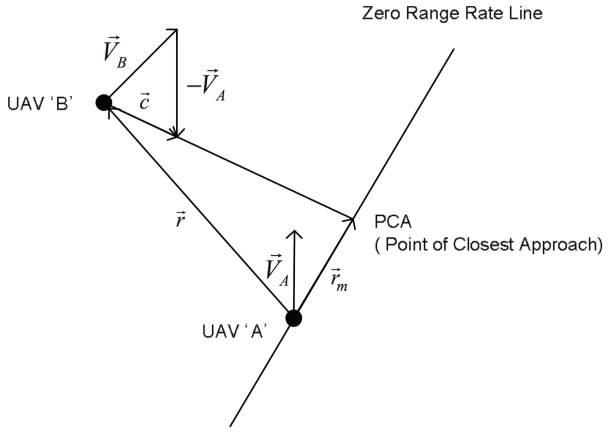



Figure 2.4: Relative motion of two UAVs in a horizontal plane [7]

From equation 2.4, it is known that $\tau < 0$ when the two UAVs are getting further. Therefore, it is essential to analyse the conflict detection when $\tau > 0$; it is considered to be a potential collision when $\vec{r_m} < r_{safe}$, where $r_{safe}$ is the pre-defined minimum protected distance for two UAVs navigate without collision.

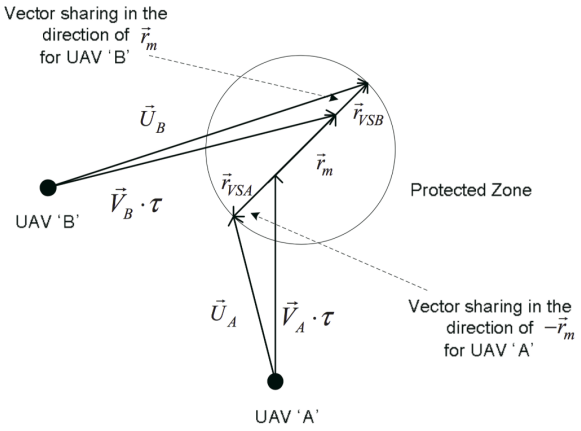



Figure 2.5: Relative motion of two UAVs in a horizontal plane [7]

As it is illustrated in figure 2.5, it is most efficient for both UAVs heading left in order to increase the miss distance to avoid the potential conflict; otherwise, it will take a longer distance to avoid the conflict if heading for the right.*Vector sharing resolution* was introduced to resolve UAV's manoeuvre in the line of miss distance vector [7]. This method gives each UAV a direction unit vector $\vec{U_A}$ and $\vec{U_B}$ to follow. The slower UAV takes more sharing due to its better manoeuvrability.

### 2.2.5 Markov decision process

Markov Decision Process (MDP) is a mathematical decision-making model for stochastic problems where the effect of an action taken in a state depend only on the current state.

In the MDP framework, the mapping from state $s$ to a stochastic action $a$ is known as control policy. The reward function $R(s,a)$ determines an instantaneous reward for the path planner's action $a$ taking at each state $s$, e.g. a positive reward for good actions and negative reward for bad actions. Markov assumption is referred to the property that the next state $s_(t+1)$ only depends probabilistically on the previous state $s$and action $a$. The ultimate goal of the path planner is to act in such a way that it maximises the long-term reward. A MDP model can be defined by the following tuple [91]:

- $S$ : set of all possible states.

- $A$ : set of all possible actions.
- $R$ : reward function. $R(s,a)$
- $T$ : probabilistic transition function, description of each action's effect on each state. $T(s,a,s')$

The MDP algorithm can be described as various optimal with probabilistic completeness. However, it has been proved that the MDP based algorithm tends to have colossal state/action spaces, leading to the solutions intractable, which is also known as the curses of dimensions [91].

### 2.2.6 Genetic algorithm

Genetic Algorithm (GA) is a powerful random search method which is adapted from the mechanism of a natural section; it seeks an optimal solution from a population in a search space in order to find the global solution by initialising and evolving population. It starts by randomly selecting the first generation with a feasible solution and takes the environment, UAV's constraints, mission goal, and other constraints into consideration. The algorithm then evaluates the fitness of each according to the specific optimisation criteria. Finally, a set of individuals are selected as parents, and the mutation and crossover will be applied to them, which the concept is that the genetic material of different members can be combined to produce an individual that would benefit from the strengths of both parents. This progress is repeated to create a new generation of solutions, and continue until a pre-set value is achieved, such as a maximum number of generation, an acceptable approximate solution, or any application-specific criterion. Due to the randomness of the crossover operation, a genetic algorithm often suffers premature convergence [87]. Therefore, the algorithm does not guarantee the path completeness, and high time complexity is required.

### 2.2.7 Rapidly-exploring random tree

Rapidly-exploring random tree (RRT) algorithm searches high-dimensional spaces by randomly building a space-filling tree. The path planner begins at the initial location and randomly expands a graph or tree by pushing the search tree away from previously constructed vertices, which allows it to search in large and high-dimensional spaces rapidly. The vehicles are considered holonomic; neither dynamic nor kinematic constraints in the original paper [92], But extension on this algorithm are developed for capturing these constraints [93], and can be immediately extended to allow for moving obstacles. It is proven to be complete in the probabilistic sense, and to produce a trajectory that is feasible given the dynamic constraints of the vehicle. However, there is no proof of the convergence rate or optimality [94].

### 2.2.8 Potential-field-based approach

The potential fields algorithm is another dominant type of representation used in path planning. This method treats the UAV as a point under the influence of the force field generated by the target node and obstacle in the area. Obstacles generate repulsive forces that repel the vehicle and target node generate attractive forces. These methods are characterised with low computation but incomplete [95]. Much of the effort of adapting potential fields have been spent in overcoming the algorithm's incompleteness. The modified potential-field-based algorithms tested to satisfy the navigation goal can be roughly divided into two categories, harmonic function-based approach, and approaches involving solving the optimal navigation distance. The modification requires segmentation of the working environment into an occupancy grid map with M points, where the segmentation scale as $O(M^D)$ with the dimension D of the environment. A single navigation function produces a trajectory for every possible starting point in the configuration space [89].

### 2.2.9 Summary

Categorising the types of algorithms is not a trivial task for this project. However, it will facilitate the choice of the appropriate algorithm by distinguishing algorithms in various aspects: 2D or 3D path planner; global or local path; path planner with or without differential constraints, etc. Table 2.2 lists the performance of various path planning algorithm surveyed concerning their operational and computational aspects.

**UAV modeling** Many algorithms have modelled the UAV as a node (point) vehicle within the airspace, which the UAV is usually represented by node and sphere in two dimensional and three-dimensional space respectively. For most of the small UAV applications, the node modelling method is sufficient to provide enough details for a trajectory because UAVs do not operate in a crowded space. Another approach is to model the UAV with its real shape, geometric information along with coordination data. Which is a much more complicated problem for the path planner to cope with, with additional geometric and kinematic properties need to be considered, such as air dynamics caused by the geometric model.

**algorithm optimality** Constantly keeping a safe distance from the obstacle is a typical requirement for autonomous UAV. Much work has been carried out to realise this goal. Such as multi-layer buffer zone for collision avoidance [96]; Voronoi diagram algorithm advocates creating a path which maximises the distance between the UAV and obstacle [87, 88]. However, taking the safest route could potentially be caused the UAV operating on an inefficient path when the flight mission has the time or energy constraints. This thesis will examine the trade-off between safety and efficiency optimality for the surveyed algorithms; the algorithm is described as various optimal if it can adjust between safety optimality and efficiency optimality.

**Weighted route** Some region of the world is more desirable for UAV to fly in that others, for example, as described in section 1.4.3, it is preferable for the path planner to

avoid the lousy GPS signal coverage area to achieve more accurate localisation information. Moreover, it also can be used to deal with uncertainty like air turbulence, bad weather, or the military restricted fly zone as listed in the objective for this project.

**2D or 3D environment** Path planning in a two-dimensional environment may be useful for certain types vehicle or navigation missions, but it is not sufficient for a UAV operating in the airspace, this thesis will examine whether the algorithms support path planning in a three-dimensional environment.

**Differential constraints** In real-world, UAV operates with its system constraints, such as maximum vehicle velocity and acceleration. In a practical case, the time has to satisfy the equation of the vehicle's motion, the velocity and acceleration should be constrained with respect to the vehicle's specification. Furthermore, the problem becomes even more sophisticated if it is modelled in a general vehicle rather than a noded vehicle, which adds *kinematics* and *dynamics* constraints to this model, and it will not be sufficient to model the vehicle by a point but with more variables to represent the vehicle's location in three dimensional space [89].

**Algorithm completeness** In this thesis, the path planning algorithm is considered to be completed if it guarantees to find a trajectory from initial point to goal point, a statement need to be returned by the algorithm is there is no possible trajectory. Furthermore, Goerzen introduced the notation of *resolution completeness* and *probabilistic completeness* depends on the type of algorithm [89]. The former one is related to the discretisation of the solution, the completeness increases with the increase of the discretisation, and it reaches the exact completeness when the discretisation reaches the continuum limit. The latter one means the algorithm will reach exact completeness with infinite computing time.

**Real time planning** It is desirable for the path planning algorithm to compute the trajectory in real-time, to allow the UAV to react in a dynamics environment, and any malfunction encountered during the flight.

**Static/Dynamic environment** Given the often uncertain nature of the real-world problem, another critical problem when planning a path is to the ability to operate with moving obstacles. The algorithm should be capable of operating in a dynamic environment.

**Time complexity** It is crucial for the path planner operate with low computational ability, to provide a faster algorithm and more rapid update for the solution, therefore a more reliable and real-time computation without lags. This thesis will compare the time complexity expressed using big $O$ notation; each parameter is explained below [89]:

- $N$ denotes the complexities of describing the obstacles space, and represent the number of bits need to define this space.
- $M$ denotes the discretisation level.
- $D$ denotes how many dimensions for the modelled environment.
- $O()$ denotes the algorithm is bounded from above.
- $(N^2)$ represents the performance of the algorithm is directly proportional to the square of the size of the input data set.

Table 2.2: Path planning algorithm comparison

| | UAV Model | Weighted Route | Algorithm Optimality | 2D/3D | Differential Constraints | Algorithm Completeness | Real-time Planning | S/D | Time Comlexity |
|---|---|---|---|---|---|---|---|---|---|
| **VD** | node | no | safety | 2D | no | yes | no | S | $O(N\log N) \leq T \leq 0(N^2)$ |
| **RRT** | both | no | efficiency | both | yes | probabilistic complete | no | SD | $O(N\log N) \leq T \leq 0(N^2)$ |
| **Dijkstra** | node | yes | various | both | yes | yes | no | SD | $O(M\log N) \leq T \leq 0(N^2)$ |
| **A*** | node | yes | various | both | yes | yes | no | SD | $O(M\log N) \leq T \leq 0(N^2)$ |
| **NN** | both | yes | various | both | yes | various | yes | S | $T > O(N^2)$ |
| **MDA** | node | yes | various | both | yes | probabilistic complete | yes | unkown | unkown |
| **GA** | both | yes | various | both | yes | no | yes | S | $T > O(N^2)$ |
| **State space** | both | yes | various | both | yes | resolution complete | no | unkown | unkown |
| **Geometric approach** | both | yes | various | both | no | depends on algorithm | yes | SD | various |
| **Potential field** | both | various | non-optimal | both | no | unkown | various | various | various |

# Chapter 3

# Methodology

This chapter explains in detail the methodological approach for the simulation framework and the autonomous navigation system. As it is illustrated in Figure 3.1, The work starts first with the problem definition to identify the challenge faced at current stage to design an autonomous navigation system for UAV, follows by the relevant literature view to identifying the current research gap, defining the aims and objectives for the thesis. The experiment process starts with the design and implement of a comprehensive simulation framework include the mathematical modelling of both the UAV and the environment and appropriate integration with the navigation system. Navigation system design contains the implementation process for mapping, localisation, cognition and control system. Lastly, the validation steps are presented with how data is collected and evaluated for the measurable objectives defined in Chapter 1.3 for this thesis.

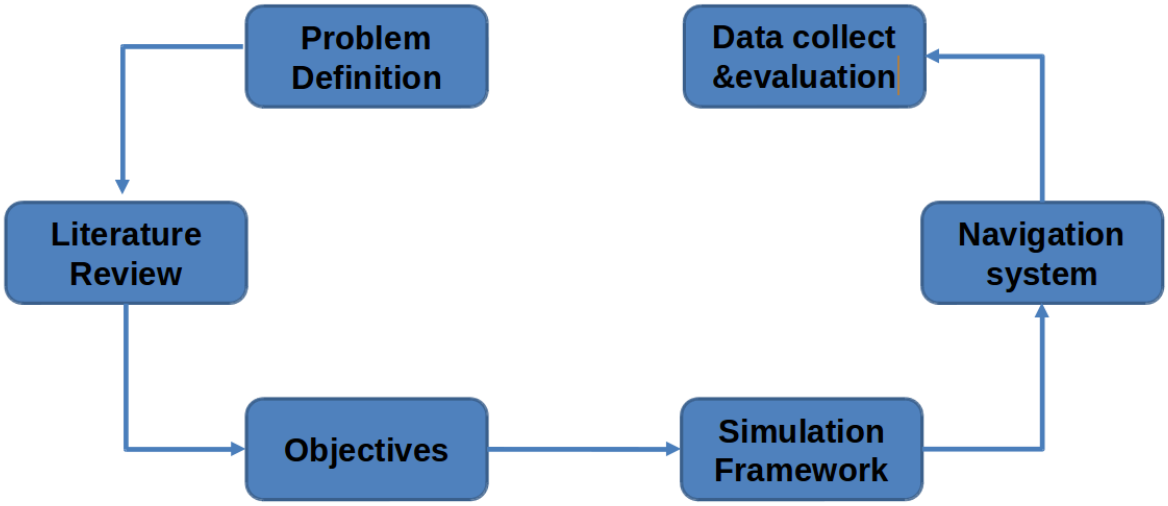



Figure 3.1: Methodology approach overview

## 3.1 Problem defination and objectives

The aim of the thesis is to design a comprehensive simulation framework for autonomous navigation system for UAV, test the algorithm's feasibility of spatial awareness and obstacle avoidance.

Where spatial awareness is one of the fundamental requirement for the UAV system, it is the UAV's ability to localise itself within the working environment, two experiments are designed to test its spatial awareness ability: 1).to estimate the UAV's position by fusing different combination of sensor implemented with noise, to evaluate the performance of the localisation system. 2).to avoid a restricted area, where the restricted area is a user-defined rectangle range defined by four coordinates points within the world coordinate system. The UAV should constantly be aware of its current location, and avoid the area if the proposed navigation waypoint is within or near the area.

Obstacle avoidance refers to the UAV's ability to keep a safe distance with both static and dynamic objects to minimise potential collisions within an unknown 3D environment.

Table 3.1: Simulated autonomous navigation system issue category

| **Simulation-related issues** | **Navigation-related issues** |
|---|---|
| simulated UAV dynamic | mapping system |
| simulated working environment | localisation system |
| simulated perception system | path planning system |
| | restricted area avoidance algorithm |
| | static obstacle avoidance algorithm |
| | dynamic obstacle avoidance algorithm |
| | control system |

In order to do this, the UAV should be equipped with an appropriate perception system to represent the detected objects within the environment. A path planning system is therefore required to determine an appropriate sequence of waypoints to the target location without colliding with the detected obstacles. Additionally, for the dynamic obstacle avoidance, the UAV should be able to predict the obstacle's future range of position based on its historical movement. Lastly, a control system is required to execute the commands to manoeuvre the UAV to a specific location with the desired orientation.

Several issues need to be solved to simulate such a sophisticated navigation system; they are categorised into the simulation-related and navigation-related issue, as it is shown in Table 3.1.

## 3.2 Simulation framework design

The simulation framework is developed based on Rotors simulator package developed by Autonomous Systems Lab at ETH Zurich [9], which is based on the integration of Gazebo and ROS (Robot Operating System). ROS provides libraries and tools to facilitate software development for the UAV autonomous navigation system. On the other hand, Gazebo is responsible for simulation, a well-designed simulator to design UAV and test navigation algorithms. It provides a robust physics engine, high-quality graphics, and

convenient programmatic and graphical interfaces.

For this thesis particular, Gazebo is responsible for the simulation framework; it provides simulated UAV dynamics, sensors, as well as the working environment for the autonomous UAV. ROS is responsible for the implementation of the actual navigation system. Instead of receiving perception information from actual sensors, it processes the simulated sensory information about the UAV's state and the environment from Gazebo, then converts and provides ROS compatible map and localisation information for the cognition system to make the appropriate operation command. Lastly, send the command to the control system to perform particular behaviour depends on the flight mission.

The design of the simulation framework can be split into three tasks. Firstly, the assembly of the UAV which consists of a body and a fixed number of rotors with motor dynamics. Secondly, A simulated environment for the UAV to work within, it should contain several static and dynamic obstacles for the evaluation of obstacle avoidance and path planning ability. Lastly, A simulated perception system, which requires the modelling and sensor characteristics and noise models, provide necessary original information for mapping and localisation system. Both the UAV and environment models are described in Simulator Description Format (SDF), which provides a solution to describe the object and the environment with their corresponding properties, such as kinematic and dynamic attributes, surface properties, joint friction, etc. For the perception simulation, Gazebo sensor plugins were used to extract the desired features from the simulated environment and UAV.

Figure 3.2 illustrates the necessary blocks for the simulation framework. This thesis focus on the left and middle blocks. The structure of the simulation framework is designed to match the real UAVs as close as possible to test real UAV on the right block.

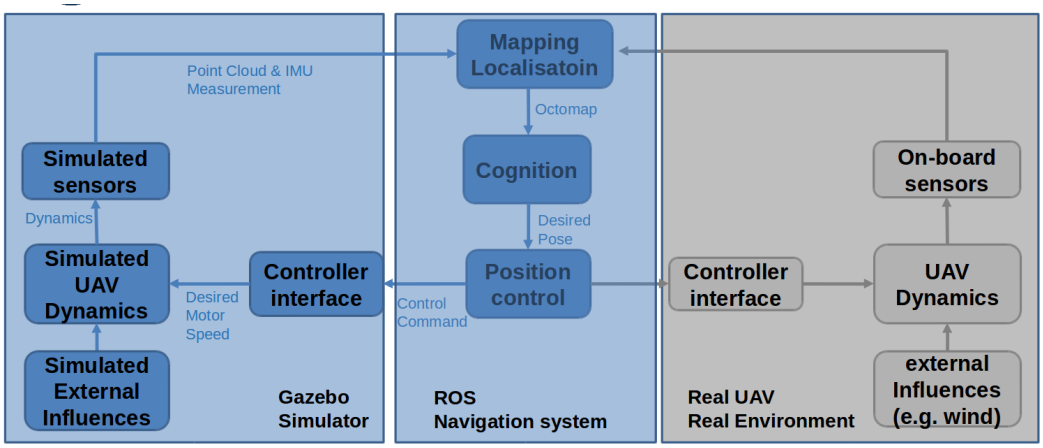



Figure 3.2: Simulation Framework overview

## 3.3 Navigation system design

The design of the navigation system is split into two low-level and high-level tasks, where the low-level task serves as a fundamental function for the implementation of high-level navigation task, such as path planning, restricted area avoidance, static and dynamic obstacle avoidance.

For the low-level tasks, firstly, the UAV is required to have a map representation of the UAV's immediate working environment, it should contain sufficient information about the obstacle's location and size with respect to the environment, and provide a world coordinate frame to allow the UAV to localise itself within. The mapping system received the input information from the simulated perception units. As it is described in Chapter 2.1.2, vision-based perception is chosen for this system due to its ability of providing rich information about the obstacle and the environment compared to LiDAR-based perception, while radar-based sensors are heavily influenced by the number of obstacles within the environment. Secondly, the localisation system should provide accurate information about the UAV's state within the environment. From the surveyed work in Chapter 2.1.1, the performance of UAV localisation can be increased with the fusion of multiple collaborative sensors. Hence experiments are designed to test the state estimation ability in

different scenarios. Lastly, a control system is required in order to drive the UAV to the target location with the desired orientation.

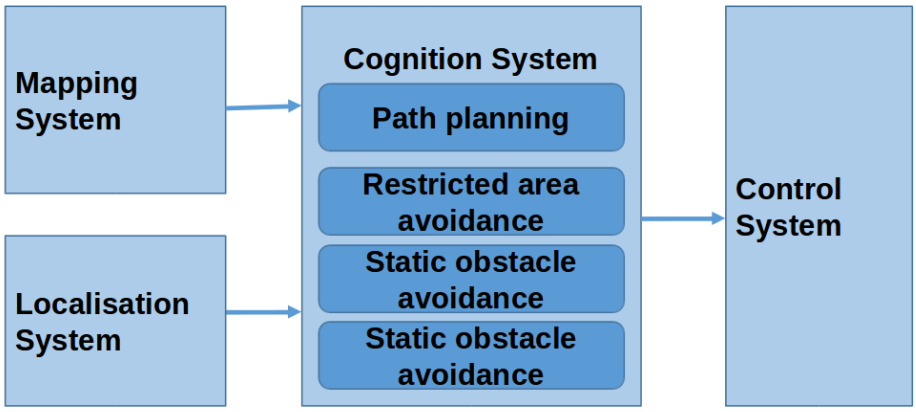



Figure 3.3: Navigation system overview

The navigation system's high-level tasks include the implementation of restricted area avoidance, static and dynamic obstacle avoidance, all of the three tasks are integrated with the implementation of path planning.

Firstly, Restricted area avoidance is a hybrid task of both low-level and high-level. On the one hand, it provides a second chance to prove the UAV's localisation ability, as the UAV is required to constantly comparing its current odometry with the user-defined restricted area. On the other hand, with the rapid development of UAV technology, it poses the challenge of airspace management, and there will inevitably be a certain airspace area that the UAV is not supposed to trespassing. Restricted area avoidance is a necessity for modern autonomous UAV navigation. The inputs for restricted area avoidance are four coordinates which define a rectangle shape area, the UAV is required to avoid the area if the initially proposed waypoint is within or close the area, ideally, with a relatively short path to avoid it while navigating to the target location.

Secondly, static obstacle avoidance, the user will need to define a 3D coordinate as the target location, along with several static obstacles within the environment which the UAV need to avoid during navigation, with the expected outcome of UAV reaching to the target without colliding with any of these objects.

Table 3.2: Navigation system input & output

| **Inputs** | **Outputs** |
|---|---|
| target coordinate & map | UAV reaching to the target location |
| restricted area coordinate & map | UAV reaching to the target location by avoiding the restricted area |
| obstacle's dynamic position | UAV reaching to the target location by avoiding the moving obstacle |

Lastly, the input for dynamic obstacle avoidance scenario is a configuration file defines the obstacle's trajectory, the UAV is expected to predict the obstacle's future movement based on the detected historical movement, and reach to the target location without colliding with the moving obstacle. A summary of the system input and output for each task is given in Table 3.2.

Due to the inherent nature between path planning, restricted area, static and dynamic obstacle avoidance, these high-level tasks are implemented together as a cognition system. Hence the navigation system could be divided into four sub-systems to solve the navigation tasks described earlier, Figure 3.3 shows the logical flow between each subsystems, the specific function and the corresponding input& output for each sub-system is given in table 3.3.

To implement dynamic obstacle avoidance in a real-world scenario, the navigation system needs to classify the types of objects detected from the perception system, identify the dynamic objects by comparing the revolution between UAV's position and obstacles' position, then chose an appropriate path by predicting the dynamic obstacles' movement based on their historical trajectory. This process normally requires the mapping system to have segmentation and feature extractor algorithm [97, 98]. Due to the limited time constraint, the dynamic obstacle is implemented separately as a stand-alone scenario without the segmentation and feature extractor process. Only one dynamic obstacle is implemented to test the algorithm's feasibility to work within a dynamic environment.

Table 3.3: Sub-system input, output & function

| Sub-systems | Input | Output | Function |
|---|---|---|---|
| mapping system | simulated visual sensor | octomap | convert visual sensory data into octomap |
| localisation system | simulate IMU | UAV pose | calculate UAV's pose based on the IMU data |
| cognition system | octomap, UAV pose & all system input Table 3.2 | operation command | make appropriate operation command, comply with identified objectives in Chapter 1.3. |
| control system | operation command | UAV behaviour | perform the desired UAV behaviour base on the operation command |

The moving obstacle is simulated with a second UAV by executing predefined trajectories. The obstacles' trajectories are designed to cover different scenarios compared to the UAV's motion (such as obstacle moving toward or away from the UAV). The experiment results are discussed and evaluated in Chapter 6.2. Furthermore, both restricted area and obstacle avoidance are implemented with an ideal odometry sensor which provides the UAV's truth state, However, A realistic localisation system is also implemented, which is capable of estimating the UAV's state by fusing different sensors such as IMU, odometry and GPS, the result is evaluated and discussed in Chapter 5.

## 3.4 Data collection and evaluation

This section includes the data collection and evaluation method for each of the scenarios. Rosbag and RViz (Ros Visualisation) are used extensively for logging experiment data. Rosbag plugin allows the user to efficiently record and playback the executed simulation by storing ROS message data in a bag-formatted file. The recorded ROS message data can then be analysed and visualised with various tools. RViz is a 3D visualiser for displaying various sensor data and state information from ROS; it is used extensively for the

Table 3.4: Case study data collection

| Case study | Data | Tool |
|---|---|---|
| restricted area avoidance | UAV trajectory; RA parameter; | gazebo animation; rosbag |
| static obstacle avoidance | UAV trajectory; map information; observation; operation time | gazebo animation; RViz; rosbag |
| dynamic obstacle avoidance | UAV trajectory; dynamic obstacle; observation; operation time | gazebo animation; RViz; rosbag |

visualisation and examination for UAV's perception system.

A mixture of qualitative and quantitative method is used to evaluate the experiment results. It includes direct observation from both Gazebo 3D animation and RViz, in additional with numerical analysis of the experiment data from rosbag.

### 3.4.1 Restricted area avoidance

Restricted area avoidance is evaluated with Gazebo simulation visualisation and rosbag package. The visualisation provides an intuitive way of observing the executed trajectory with the defined restricted area. Another aspect of restricted area avoidance is to find a relatively short path while exiting the restricted area; different scenarios are designed to test the algorithm's ability to find the exiting corner by placing the UAV at different positions to the restricted area. The executed trajectory can be evaluated by accessing to the recorded rosbag data.

### 3.4.2 Static obstacle avoidance

The primary method to evaluate static obstacle avoidance performance is by observing the visualisation from both gazebo simulator and RViz. Where the gazebo simulator provides the actual human-readable real-world animation of the navigation process, on the other hand, the RViz provide the sensory visualisation how the UAV is perceiving about

the working environment, include the map resolution, UAV's position and orientation in relation to the map etc.

As the aim of the thesis is to provide an outdoor navigation algorithm, the primary solution to avoid collision with obstacles is to 'go-up' where there is a high chance of free airspace. Two sets of the environment model are provided to test the algorithm's ability when the UAV is working within an outdoor and roofed environment. The recorded operation time from gazebo simulator (or from ROS log data) is used to evaluate the algorithm's time efficiency. It is also noted that the UAV tend to struggle to find a valid path when it is working in a complex environment with a lot of obstacles, and relative open airspace with extremely small obstacles. Hence rosbag and RViz packages are chosen to evaluate the algorithm limitation by observing the UAV's position and orientation relative to the map representation. Lastly, different experiments are designed to evaluate effects caused by mapping and cognition system, which includes the map resolution size, the calculation of the next waypoint, and the region of interest covered for obstacle detection.

### 3.4.3 Dynamic obstacle avoidance

For dynamic obstacle avoidance case study, the UAV is given a fixed target location and expected to reach the target by avoiding the detected moving obstacle. It is important for the algorithm to deal with different scenarios how the moving obstacle's trajectory is in relation to the UAV's movement, such as the obstacle is moving in the same direction or opposite direction with the UAV. Chapter 6.2 contains a detailed report of different encountered scenarios between UAV and the dynamic obstacle. Furthermore, the algorithm works on the basis of predicting the obstacle's future position range from the estimated obstacle's velocity, where the obstacle's velocity is estimated by comparing its historical displacement within a fixed time interval. Experiments are designed to evaluate how the time interval affects the accuracy of velocity estimation. The executed UAV and obstacle's trajectory are evaluated by accessing to the record rosbag data.

# Chapter 4

# Navigation system design

## 4.1 Mapping System

This section includes the design and implementation of the mapping system. As it is shown in Figure 4.1, the mapping is divided into two steps: perception and map construction.

The perception step utilises a depth camera to convert the simulated environment from Gazebo into ROS compatible point cloud data. Because Gazebo and ROS are separate projects that do not depend on each other, the simulated sensor from Gazebo cannot directly communicate with ROS, libgazebo_ros_openni_kinect.so plugin is used to make the depth camera data publish point clouds to ROS topics by defining the relevant namespace and topics. Furthermore, the plugin also allows the user of fine-grained control over how the environment is perceived with several parameters. Table 4.1 shows the depth camera's relevant parameters which can affect the mapping performance and their values. Due to limited time constraint, depth camera is not implemented with simulated noise, therefore with zero value for all the distortion tags.

Table 4.1: Perception unit parameters for mapping system

| Parameter | Description |
|---|---|
| frame rate | refers to the number of individual frames that comprise each second of the recorded video, also known as FPS (frames per second.) |
| max range | defines the maximum distance that the depth camera can detect, with defaul value of 10 meters. |
| width & height | the resolution of the recorded video, with default value of 640×480. |
| horizontal fov | Field of View is the extent of the observable environment that can be detected by the depth camera, horizontal FoV describes how wide the depth camera can detect horizontally, with default value of 2 radian. |
| near&far clip | near &far clip distance refer to the near and far plane of the viewing frustum, anything closer than the near clips distance or further than the far clip distance will not be displayed. the far clip value should not exceed the max range. the default value are 0.1 and 10 meter (s) by default. |
| pointcloud cutoff min&max | similar to near&far clips, defined the viewable processed point cloud data range. only the objects within this range are displayed as point cloud data, the min&max value cannot exceed the near&far clip value. |
| distortion K1 K2 K3 T1 T2 | defines the distortion of the depth camera. |

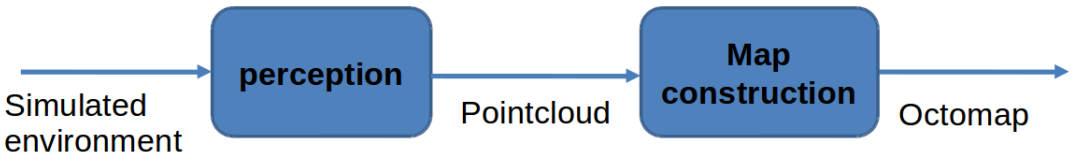



Figure 4.1: Mapping system overview

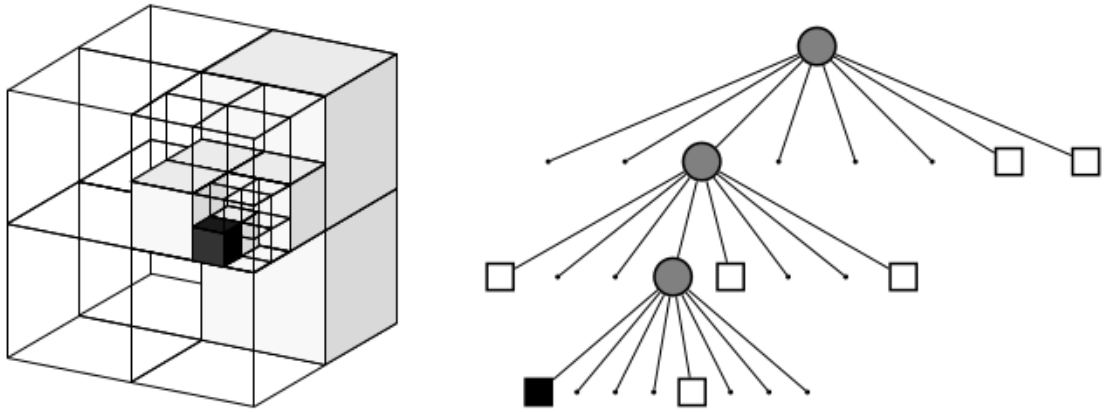


Figure 4.2: An octree example [8]

Octomap is used for the map construction step, and it converts the processed point cloud data into a map representation [8] . The environment is described with octree representation, where the octree is a node that contains eight children nodes, which is more efficient for memory storage. It can be used to describe 3D space occupancy. The 3D environment is segmented into multiple spaces which are denoted as a node, and each space can be further segmented into eight sub-space with the same size, denoted as octants. The segmentation process can be applied recursively until the map reaches the desired resolution to sufficiently represent more complex parts of the working environment. The left part of Figure 4.2 shows an example of an octree, where the free cells are shaded white and occupied cells are shade black, along with its tree representation.

To use the octomap package[1], the user needs to add an octomap rosnode into the launch file by:

```
<node pkg="octomap_server" type="octomap_tracking_server_node" name
    ↪ ="octomap_talker" output="screen" args="$␣(arg␣path)">
  <param name="resolution" value="␣(with␣its␣value)" />
  <param name="frame_id" type="string" value="␣(with␣its␣value)" />
```

[1] available at //github.com/OctoMap/octomap.git

```
    <param name="sensor_model/max_range" value="␣(with␣its␣value)" />
        ↪ %
    <remap from="cloud_in" to="/firefly/vi_sensor/camera_depth/depth/
        ↪ points" />
</node>
```

As it is described in the ”param name”, it will launch the octomap rosnode with user-specified properties of map resolution, maximum range (cannot exceed the max range of the depth camera) and the world it will listen to. Lastly, the ’remap’ line connects the ”/firefly/vi_sensor/camera_depth/depth /points” topic as the input topic (”cloud_in”) to the mapping system which will transfer the point cloud data into octree representation. Figure 4.3, 4.4 and 4.5 shows the visualised transformation process.

Within all the parameters, resolution can directly affect the mapping and navigation performance, as higher resolution (smaller value) provides better detail of the environment but requires more computational capability, and vice versa. Chapter 6.1 contains a details report on how resolution affects the autonomous navigation.

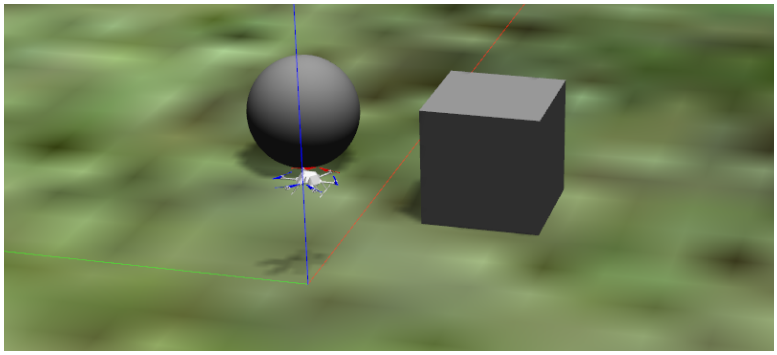


Figure 4.3: The simulated real world environment

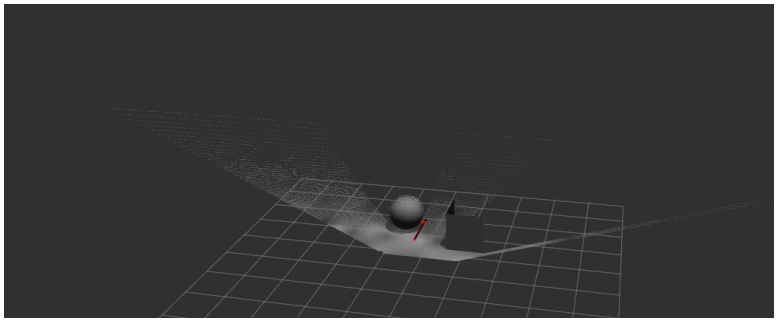


Figure 4.4: The environment represented in point cloud

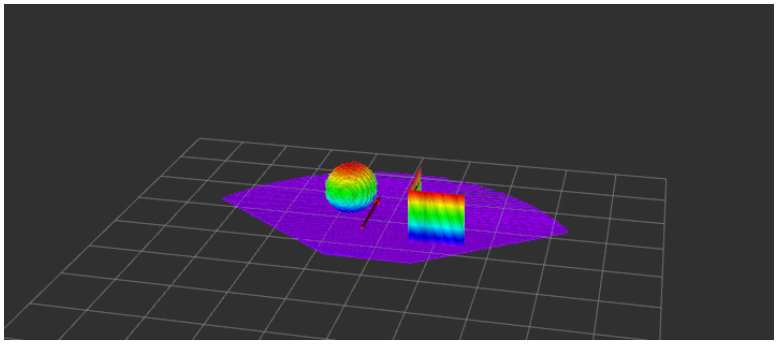


Figure 4.5: The environment represented in octomap

### 4.1.1 Sensor range and mounting position

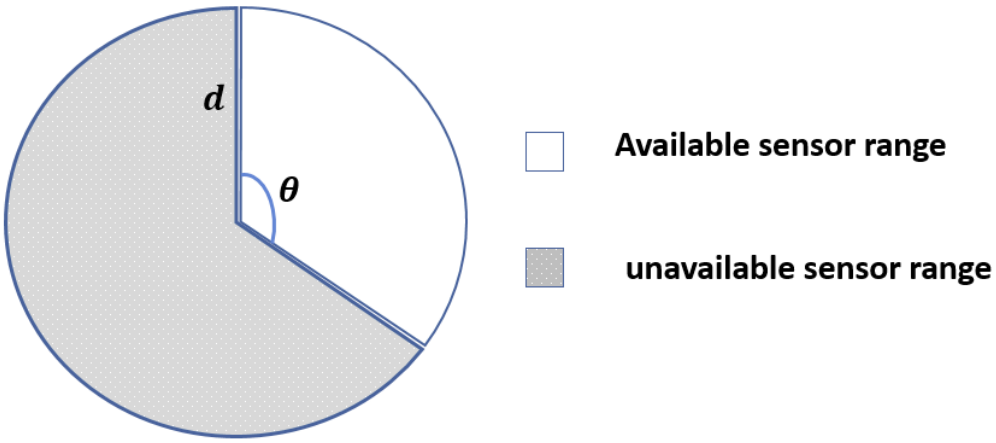



Figure 4.6: A depth camera with its available sensing range in 2D

Figure 4.6 shows a 2D horizontal overview of the Field of View (FoV) in the camera's first-person perspective, with a maximum distance of $d$ and viewing angle of $\theta$. However, as the camera is not mounted by the edge of the UAV and the sensor will constantly perceiving data about the UAV itself, A cutoff value $c$ (point cloud cutoff in Table 4.1) is introduced to eliminate the unnecessary sensory information, any objects detected within the cutoff range will be ignored. As it is illustrated in Figure 4.7, the UAV is represented by a rectangular with width of $w$, with horizontal FoV of of $\theta$ and cut-off value $c$. The shaded area represents the cutoff range that will not be displayed, which results in a minimum viewing width of $w_{min}$. The ideal value for $w_{min}$ should be equal to the width$w$ of the UAV for the sensor to perceive sufficient information from the environment while neglecting the UAV itself. Hence the ideal cutoff value $c$ can be calculated with given horizontal FoV angle $\theta$ and UAV width $w$, by the following equations.

$$\begin{aligned} w_{min} = w &= 2 \times c \times \tan\frac{\theta}{2} \\ c &= \frac{w}{2} \times \cot\frac{\theta}{2} \end{aligned} \tag{4.1}$$

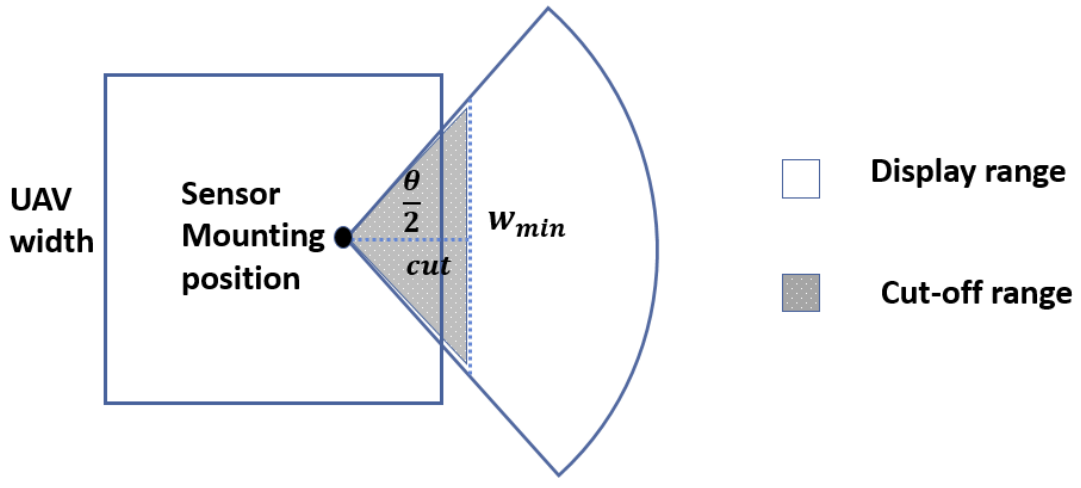



Figure 4.7: Available sensor range with compensated cutoff value in 2D

## 4.2 Localisation system

One of the essential aspects to allow stable and robust navigation is the access to an accurate knowledge of the state of itself, it normally consists of UAV's position and orientation with respect to the working environment. While IMU measurement is available on all of today's UAVs, it measures acceleration and angular velocity by using a combination of accelerometers and gyroscopes. Mostly, the acceleration measurement is integrated into velocity then further integrated into position information where the orientation estimation is calculated straight from the angular velocity measurement. IMU measurements are normally available as high rate with low delay. However, the measurements are corrupted with noise and time-varying bias [9]. This chapter includes the implementation process of fusing an IMU sensor with a generic odometry sensor to achieve for UAV's state estimation. Experiment results are evaluated and discussed in Chapter 5.

### 4.2.1 IMU model

The IMU is assumed to consist of a certain bias ($b_a$ for acceleration, $b_\omega$ for angular velocity) and additive, zero-mean Gaussian noise $\eta$. Thus, the IMU model for angular velocity ($\theta$) and acceleration ($a$) are given in Equation 4.2 and 4.3. Where $a_m$ is the measured acceleration in $m/s^2$, $\omega_m$ is the measured angular velocity in $degrees/sec$. The biases $(b_a, b_\omega)$ are modelled as time-derivative, with zero mean Gaussian stochastic process in Equation 4.4 and 4.5. A plugin is used to attach the simulated IMU sensor to the UAV by including imu_plugin_macro xacro file in the UAV's description file with the correct namespace to connect to the UAV and sensor noise parameters. The parameters are identified on a real IMU sensor, ADIS16448 by Analog Devices[2]. As it is shown in Table 4.2, the IMU plugin will take the noise parameter as input and output sensor_msgs/Imu data type with the measured orientation[3]angular velocity, linear acceleration, and their noise

[2]datasheet available at https://www.analog.com/en/products/adis16448.html

[3]ADIS16448 does not provide straight orientation measurement, the orientation is calculated based on the angular velocity,hence the associated value will be set to -1

Table 4.2: IMU system inputs & outputs

| **Input** | | | **Output** |
|---|---|---|---|
| **parameter** | **value** | **unit** | |
| gyroscope_noise_density | 0.0003394 | $rad/s/sqrt(Hz)$ | orientation |
| gyroscope_random_walk | 0.000038785 | $rad/s/s/sqrt(Hz)$ | orientation covariance |
| gyroscope_bias_correlation_time | 1000.0 | $s$ | angular velocity |
| gyroscope_turn_on_bias_sigma | 0.0087 | $rad/s$ | angular velocity covariance |
| accelerometer_noise_density | 0.004 | $m/s^2/sqrt(Hz)$ | linear velocity |
| accelerometer_random_walk | 0.006 | $m/s^2/s/sqrt(Hz)$ | linear velocity covariance |
| accelerometer_bias_correlation_time | 300.0 | $s$ | |
| accelerometer_turn_on_bias_sigma | 0.1960 | $m/s^2$ | |

covariance respectively.

$$a = a_m - b_a - \eta_a \tag{4.2}$$

$$\omega = \omega_m - b_\omega - \eta_\omega \tag{4.3}$$

$$\dot{b_a} = \eta_{b_\omega} \tag{4.4}$$

$$\dot{b_\omega} = \eta_{b_a} \tag{4.5}$$

### 4.2.2 Generic odometry sensor

An odometry sensor is used to mimic any generic on-or off-board tracking system such as GPS. The plugin is responsible for publishing nav_msgs/Odometry messages type. This means position, orientation, linear and angular velocity of the UAV are directly provided by a Gazebo plugin. The noise parameter of the odometry sensor can be set in the UAV description file with odometry_plugin_macro xacro definition. As it is shown in Table 4.3, the noise parameters are available as normal or uniform distribution for both pose (position and orientation) and velocities (linear and angular), the implemented odometry sensor is simulated with noise model of the normal distribution as the standard deviation of additive white gaussian noise, with the corresponding unit. The position noise value is implemented based on the standard deviation of daily worst site $95^{th}$ percentile position error for 2016 [99]. The noise for linear velocity is based on the GPS accuracy analysis in [100]. Although it is possible to use GPS to determine the orientation and angular velocity, it is not generally used for on Earth or in the air. GPS provides the orientation typically through its onboard compass or other types of sensor, the orientation noise is chosen based on a low-cost gyroscope ADxRS450 provided by Analogue Devices [4], Similarly to the implemented IMU, the orientation is also calculated based on the angular velocity measurement, therefore, no noise for the orientation measurement.

### 4.2.3 Sensor fusion

Robot_localisation package is used to fuse the simulated IMU and odometry sensor data to achieve UAV's state estimation [101], which is implemented as nonlinear state estimator for any UAV moving in 3D space. It provides two types of Kalman Filter techniques: Extended Kalman Filter (EFK) and Unscented Kalman Filter (UKF). EKF is generally faster, and UKF is slower but more accurate for nonlinear transformations. However, EKF is proved to perform as good as UKF for robot localisation problem [102], only EFK is implemented for the localisation system.

[4] datasheet available at :https://www.analog.com/en/products/adxrs450.html#product-overview

Table 4.3: Odometry system inputs & outputs

| **Input** | | | **Output** |
|---|---|---|---|
| **parameter** | **value** | **unit** | |
| noise_normal_position | 0.37/0.37/0.36 | $m$ | position |
| noise_normal_quaternion | n/a | $rad$ | position covariance |
| noise_normal_linear_velocity | 0.0012/ 0.0012/ 0.003 | $m/s$ | orientation |
| noise_normal_angular_velocity | 0.00026 in $rpy$ | $rad/s$ | orientation covariance |
| noise_uniform_position | n/a | m | linear velocity |
| noise_uniform_quaternion | n/a | rad | linear velocity covariance |
| noise_uniform_linear_velocity | n/a | m/s | angular velocity |
| noise_uniform_angular_velocity | n/a | rad/s | angular velocity covariance |

The package is capable of fusing any arbitrary number of sensors, and functions when the UAV is equipped with multiple IMUs or multiple odometry source to track the the UAV's state in 15 dimensions $[x, y, z, roll, pitch, yaw, x', y', z', roll', pitch', yaw', x'', y'', z'']$. Different ROS message type is supported, including the sensor_msgs/ Imu and nav_msgs/ Odometry from IMU and odometry model respectively. More importantly, it supports continuous estimations, which means the localisation system will continue to estimate the UAV's state via IMU when there is no external odometry data. Finally, the package also provides navsat_transform_node, which aids in the integration of GPS data when transferring the simulate navigation system into a real UAV.

**EKF model**

The EKF algorithm can be divided into two parts: predict and update. In the predict part, the EKF estimates the UAV's states base on the sensor measurements along with their uncertainties. Once the system receives the next sensor measurement, the previous precited state is then will be compared with the sensor measurement and updated with a weighting factor (Kalman Gain), with more weight given to the sensors with lower noise parameter. The process is recursive, and it operates in real-time with only the present sensor measurement and previous calculated UAV's state and its covariance matrix.

$$x_k = f(x_{k-1}) + w_{k-1} \tag{4.6}$$

$$z_k = h(x_k) + v_k \tag{4.7}$$

Equation 4.6 and 4.7 shows the mathematic model for the predicted UAV's state and sensor measurement. where $x_k$ and $z_k$ are the predicted state and predicted measurement at time $k$ respectively. $f()$ and $h()$ are nonlinear function for state transition and sensor measurement. $w$ and $v$ are normally distributed process noise and measurement noise with subnote denoting the time sequence ($k-1$ and $k$). However, $f()$and $h()$ can only

be applied to the covariance with the computation of a partial derivative (the Jacobian) matrix. The Jacobian is evaluated with the predicted state at each time step. This process essentially linearises the non-linear function around the current estimate.

$$\hat{x}_k = f(x_{k-1}) \tag{4.8}$$

$$\hat{P}_k = F P_{k-1} F^T + Q \tag{4.9}$$

Equation 4.8 and 4.9 shows the linearised estimation model for UAV's state and sensor measurement respectively. These two equations carry out a prediction step that projects the current state estimate and error covariance forward in time. The estimated error covariance $P$ is projected via $F$, which is the Jacobian of $f()$, and then perturbed by the process noise covariance $Q$ (As it is given in Table 4.4).

$$K = \hat{P}_k H^T (H \hat{P}_k H^T + R)^{-1} \tag{4.10}$$

$$x_k = \hat{x}_k + K(z - H\hat{x}_k) \tag{4.11}$$

$$P_k = (I - KH)\hat{P}_k (I - KH)^T + KRK^T \tag{4.12}$$

After predicting the UAV's state and sensor measurement, the algorithm now needs to determine the Kalman Gain to update and correct those predicted values. As it is shown in Equation 4.10, the Kalman Gain is calculated with the estimate covariance matrix $\hat{P}_k$ (where $\hat{P}_0$ is the initial noise covariance described in Table 4.4), along with observation model matrix $H$ and measurement covariance $R$ from the sensor. Equation 4.11 and 4.12 gives the model for the updating UAV's state and error covariance with Kalman Gain, where $I$ denotes the identity matrix. In standard EKF formulation, $H$ should be a jacobian matrix of the observation model function $h()$. In robot_locolisation package, $H$ is simpli-

fied into an identity matrix under the assumption that each sensor produces measurement of the estimated state [101].

### Effect of noise covariance

The performance of EKF localisation is significantly affected by the selection of the assumed covariance matrices $Q$, $R$. $\hat{P}_0$ is coupled with the presumed initial state and affects the initial convergence of the filter. In many situations, the effect of $\hat{P}_0$ is not significant. The $Q$ and $R$ can be considered as the weighting factors between the estimation and the measurement equation. The ratio is represented as the Kalman Gain equation (Equation 4.10) where larger $Q$ is equivalent to less confidence of these state equations and more correctness with the measurement update. Likewise, Larger $R$ is equivalent to less confidence in the measurement and less correctness with the measurement update.

For this thesis particularly, the value of measurement noise $R$ is calculated from the combined sensor noise from IMU and generic odometry sensor (Give in Table 4.2 and 4.3), it is represented using "standard deviation" ($\sigma$) of the measured value from true values during the calibration. Hence, the performance of the filter will mainly rely on the choice of the process noise covariance. Higher $Q$ value results in more weight to the noisy measurement and the estimation accuracy are compromised. On the other hand, lower Q results in better estimation accuracy but with a time lag required. Choice of the process noise is based on the trade-off between estimation accuracy Vs time lag.

```
<node pkg="robot_localization" type="ekf_localization_node" name="
    ↪ ekf_se" clear_params="true">
<rosparam command="load" file="$␣(find␣robot_localization)/params/
    ↪ ekf_template.yaml" />
</node>
```

To utilise robot_localisation package for state estimation with EKF, the above code is added to the simulation launch file. Where the first line states the simulation to include the robot_localization node within its EFK mode, the second line includes the

Table 4.4: Sensor fusion configuration parameters

| Parameter | Description |
|---|---|
| frequency | The frequency that the node will output a state estimation. The filter will not work until it receives at least one message from any one of the inputs. It will then run continuously at the frequency specified here, regardless of whether it receives more measurements. set with the default value of 30Hz. |
| sensor timeout | The period, in seconds, after which we consider a sensor to have timed out. In this event, we carry out a predicted cycle on the EKF without correcting it. This parameter can be thought of as the minimum frequency with which the filter will generate new output. Defaults to $1/frequency$ if not specified. |
| imu/odom config | A $1 \times 12$ boolean matrix defines which sensor reading will be used as input for the filter, and it allows the user to control over which values from each measurement are fed to the filter. it is ordered as $[x, y, z, roll, pitch, yaw, x', y', z', roll', pitch', yaw', x'', y'', z'']$ |
| imu/odom differential | When measuring the UAV's pose from two sensors, the filter will face the problem of switching between each measurement when both sensors are under report of covariance. When the differential mode is enabled, all pose measurement will be converted to velocity data by differentiating the pose. This mode has no effect on velocity measurement. Set to false. |
| imu/odom relative | When this mode is enabled, the first measurement is treated as "zero-point" for all future measurement. set to True |
| process noise covariance | the process noise matrix, the matrix represents the noise the filter adds to the total error after each prediction step, ordered as $[x, y, z, roll, pitch, yaw, x', v', v', roll', pitch', yaw', x'', y'', z'']$ |
| initial noise covariance | This represents the initial value for the state estimate error covariance matrix, Setting a diagonal value (variance) to a large value will result in rapid convergence for initial measurements of the variable in question, ordered as $[x, y, z, roll, pitch, yaw, x', y', z', roll', pitch', yaw', x'', y'', z'']$ |

Table 4.5: Filter input state from imu &odometry sensor (T:True F:False)

| | $x$ | $y$ | $z$ | $roll$ | $pitch$ | $yaw$ | $x'$ | $y'$ | $z'$ | $roll'$ | $pitch$' | $yaw'$ | $x''$ | $y''$ | $z''$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| imu config | F | F | F | F | F | F | F | F | F | T | T | T | T | T | T |
| odom config | T | T | T | F | F | F | T | T | T | T | T | T | F | F | F |

ekf_template.yaml file which contains the configuration file describes how the sensor fusion process is implemented. Necessary parameters within the ekf_template.yaml file and their descriptions are given in Table 4.4. Succesful configuration leads to the node publish a nav_msgs/Odometry type of message on firefly/odometry/filtered topic, which contains the UAV's state estimation in 15 dimensions. The yaml file links the output data from IMU and odometry sensor by specifying the output topic from firefly/vi_sensor/ground_truth/ odometry and firefly/vi_sensor/imu respectively.

In robot_localisation, there are two general rules about the sensor choice for fusion:

- If the odometry provides both position and linear velocity, fuse the linear velocity.
- If the odometry provides both orientation and angular velocity, fuse the orientation.

The above snippet holds true most of the time because most odometry sources approximate the absolute position by integrating velocity. However, in this thesis particularly, the odometry sensor is implemented to mimic GPS, It can be treated as a combination of single GPS with a gyroscope, and the orientation is provided by angular velocity measurement from the gyroscope. Hence, the odometry sensor provides position, linear velocity and angular velocity measurement for the EKF filter. On the other hand, the IMU provides its angular velocity and linear acceleration measurement. The integration state is given in Table 4.5. Other combinations of filtered results are evaluated and discussed in Chapter 5.1.

## 4.3 Cognition system

This section includes the design and the implementation for the cognition system, along with selected objectives to support the aim of achieving collision-free navigation and spatial awareness. Where the experiment results are evaluated and discussed in Section 6. The remainder of the section is structured as follows:

- Section 4.3.1: implementation for spatial awareness, include the the design process for the UAV to avoid an restricted area.
- Section 4.3.2: implementation for navigation with static obstacles
- Section 4.3.3: implementation for navigation with dynamic obstacles

### 4.3.1 spatial awareness - restricted area avoidance

To avoid a restricted area, the UAV needs prior information about the size, location of the restricted area. Secondly, the UAV needs to identify potential intrusion with the restricted area and determine an appropriate trajectory to the target while avoiding the restricted area. However, it is futile and a waste of computer resource to run the recursive function for the whole navigation process. A condition is given to the algorithm to make sure these steps are only executed when the UAV is relatively close to the restricted area. Figure 4.8 shows the flowchart of the method the implemented restricted area avoidance algorithm. It contains three main steps::

1. Constructing a bounding box with received restricted area coordinate.
2. Check whether the trajectory conflicts with the restricted area.
3. Find the exit solution if the trajectory conflicts with the restricted area.

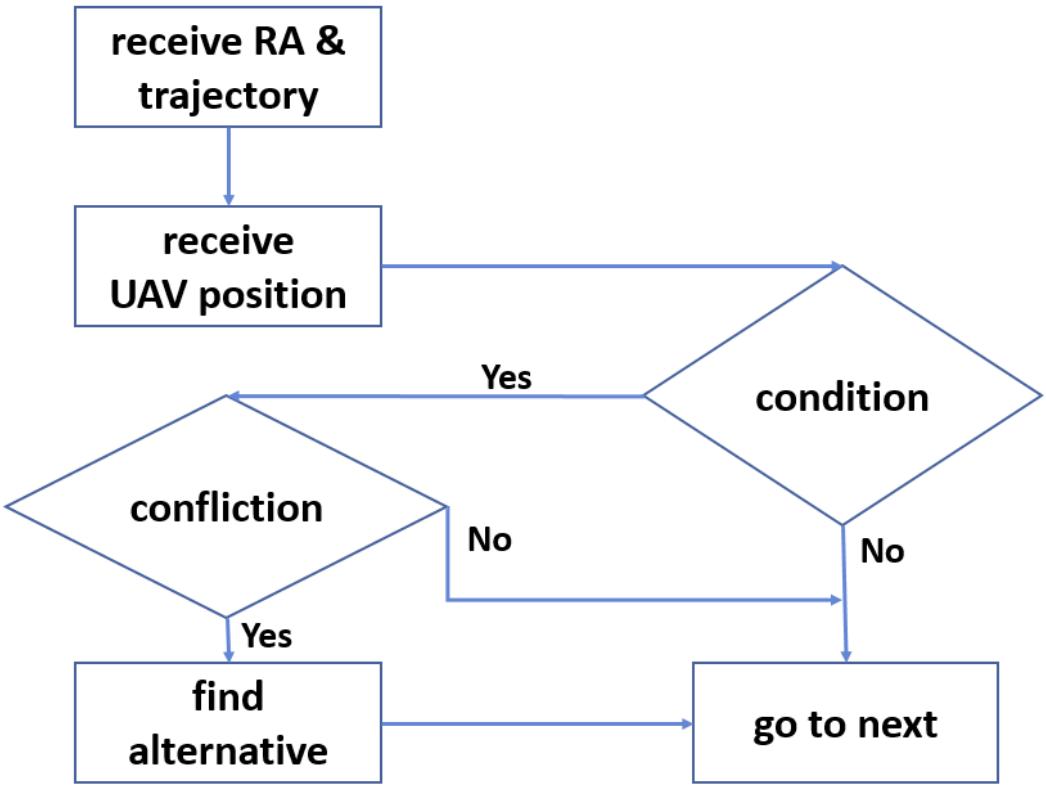



Figure 4.8: Flowchart of the methodology implemented for restricted area avoidance

**Step 1** Figure 4.9 shows a rectangular shape restricted area with both sides aligned to x, y-axis with maximum and minimum coordinate of $(x_{max}, y_{max})$ and $(x_{min}, y_{min})$. Thus, the geometric shape of the restricted area can also be represented by four bounding lines $(x_{min}^{boundary}, x_{max}^{boundary}, y_{min}^{boundary}, y_{max}^{boundary})$ respectively. As it is shown in Figure 4.10, the size of the restricted area will be automatically increased by $e$ after receiving instruction from the user, to compensate the error from odometry sensor and the rotor control. where $e$ consist the distance the UAV has travelled during the response time $e_r$, the odometry sensory distance error $e_s$, and the rotor control error $e_c$. Due to the time constraint, system error for the implemented system is set to be two meters by default.

$$e = e_c + e_s + e_r \tag{4.13}$$

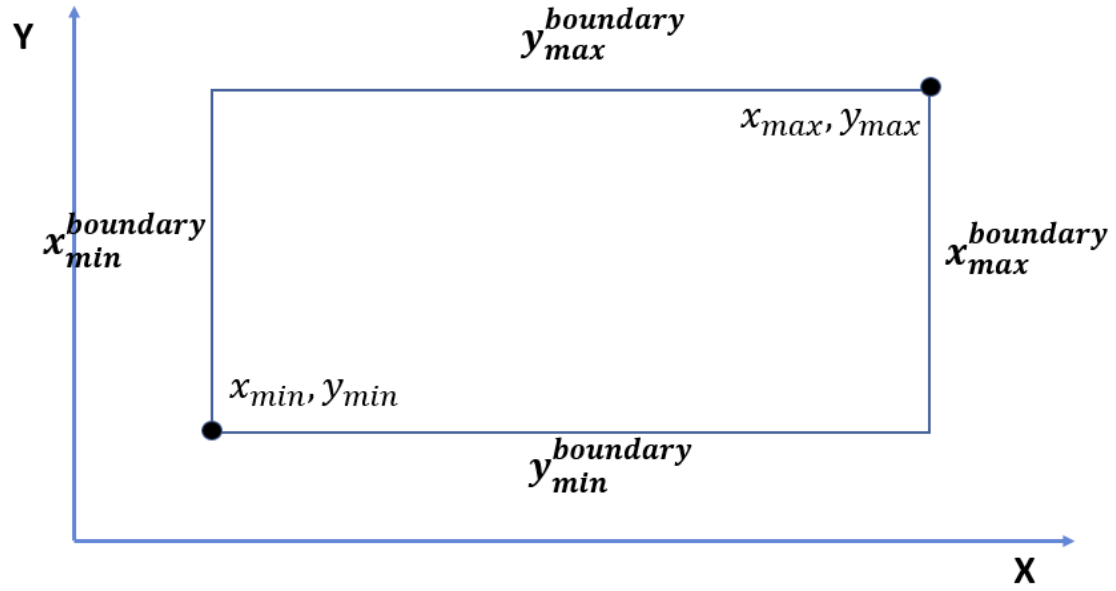



Figure 4.9: A user defined rectangle restricted area

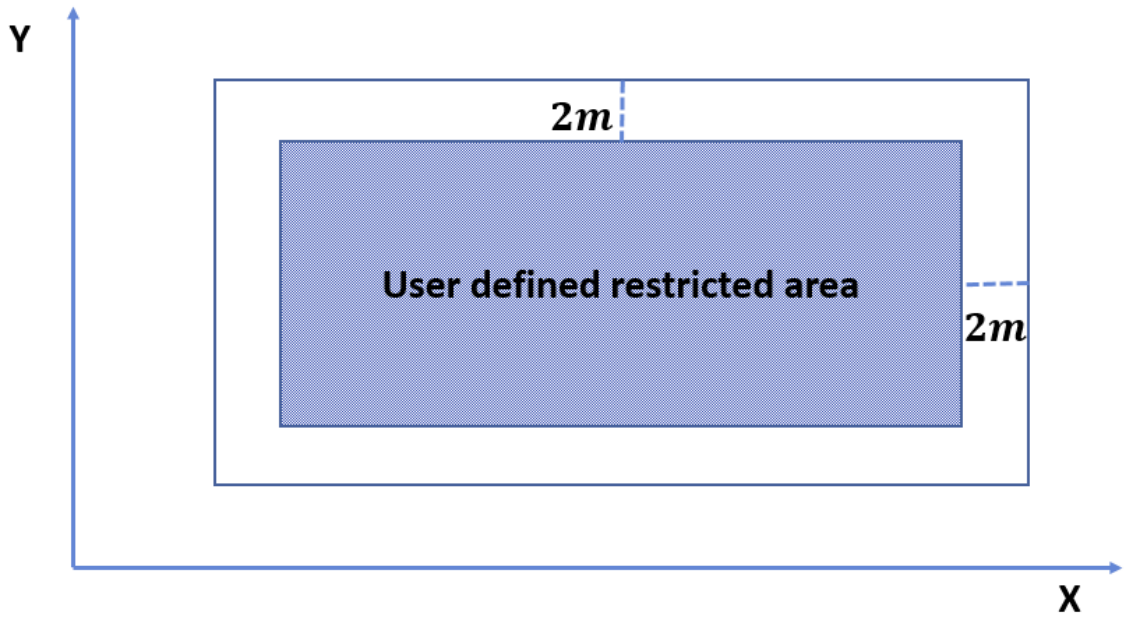



Figure 4.10: Resize restricted area by introducing a buffer-zone

**Step 2** As the UAV will only start to check the restricted area when the UAV's current odometry is approaching any one of the four restricted boundaries, therefore it is safe to assume the trajectory will not conflict with the restricted area when both the UAV's position and target lies on the same side of any of the bounding line. With the following two conditions. 1): both the current odometry and target coordinates are located outside of the rectangle. 2): they are located on the same side of the rectangle. As it is shown in Figure 4.11, the trajectory is considered to be safe, and the UAV will continue to navigate when any one of the following four scenarios are met, where $(x_c, y_c)$ and $(x_t, y_t)$ are coordinate for UAV's current position and target respectively.

- $x_c \leq x_{min}^{boundary}$ AND $x_t \leq x_{min}^{boundary}$
- $x_c \geq x_{max}^{boundary}$ AND $x_t \geq x_{max}^{boundary}$
- $y_c \leq y_{min}^{boundary}$ AND $y_t \leq y_{min}^{boundary}$
- $y_c \geq y_{max}^{boundary}$ AND $y_t \geq y_{max}^{boundary}$

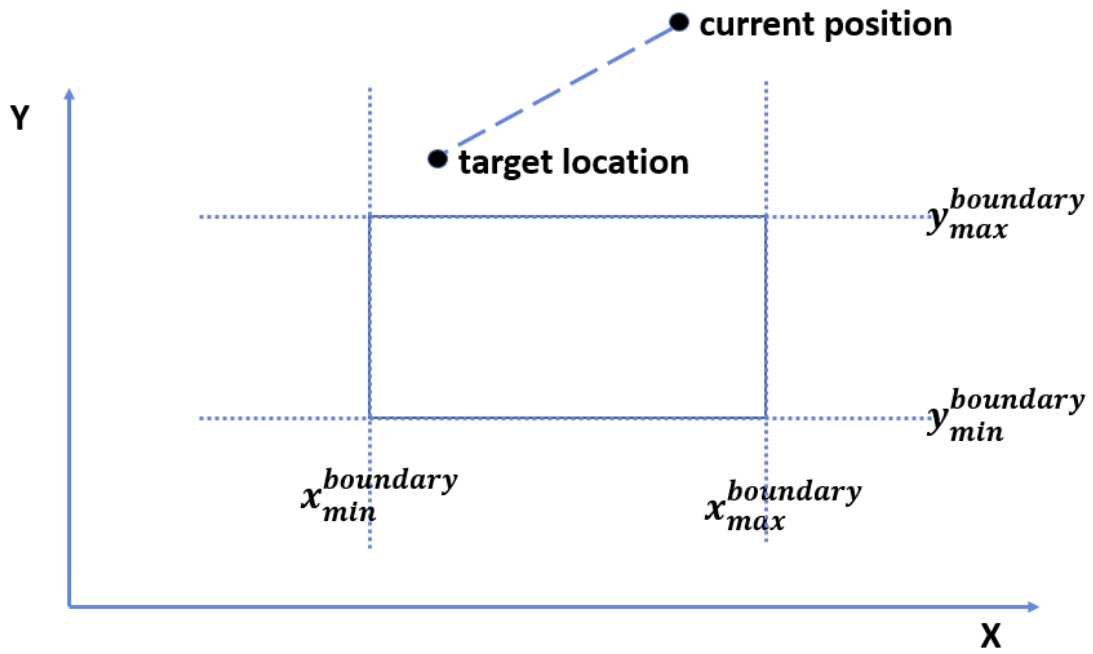



Figure 4.11: Determine whether the trajectory is blocked by the restricted area

For any others cases where the current odometry point and target points do not both reside on the same side of the rectangle, the UAV will determine which one of the four boundaries the trajectory will potentially intrude and corresponding exit strategy will be given. To find out which boundary the trajectory will potentially collide into, the UAV will first derive the line function the current point and target point by:

$$\frac{y-y_1}{x-x_1} = \frac{y_2-y_1}{x_2-x_1} \tag{4.14}$$

Derive Equation 4.14 to get the solution for $x$ and $y$ on the line:

$$x = \frac{x_2-x_1}{y_2-y_1}x + \frac{x_1y_2-x_2y_1}{y_2-y_1} \tag{4.15}$$

$$y = \frac{y_2-y_1}{x_2-x_1}x + \frac{x_2y_1-x_1y_2}{x_2-x_1} \tag{4.16}$$

After getting the line function (potential trajectory) between the UAV and the target. With Equation 4.15 and 4.16 and the given rectangular information $(x_{min}^{boundary}, x_{max}^{boundary}, y_{min}^{boundary}, y_{max}^{boundary})$, it is possible to find out the four intersection points$(x_1,y_1),(x_2,y_2),(x_3,y_3),(x_4,y_4)$ between the trajectory and the four boudning lines, where $(x_1,y_1)$ are the insection point between the trajectory and $y_{min}^{boundary}$ bounding line and the rest are illustrated in Figure 4.12. The trajectory will be identified as potential collision if the insection points are within the boundaries by checking the following conditions.

- $y_{min}^{boundary} \leq y_1 \leq y_{max}^{boundary}$
- $y_{min}^{boundary} \leq y_3 \leq y_{max}^{boundary}$
- $x_{min}^{boundary} \leq x_2 \leq x_{max}^{boundary}$
- $x_{min}^{boundary} \leq x_4 \leq x_{max}^{boundary}$

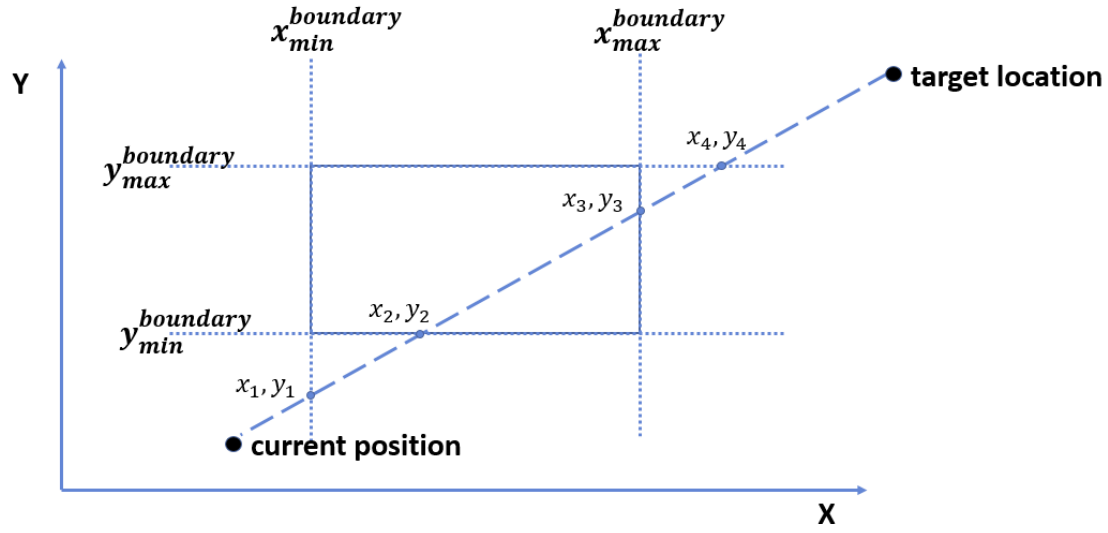



Figure 4.12: Intersection points between the potential trajectory and the bounding box

**Step 3** After the trajectory is determined to be in potential collision with the restricted, the UAV needs to find the appropriate exit strategy to avoid the restricted area. As it is shown in Figure 4.13, there are two scenarios of how the trajectory conflicts with the restricted rectangle area, the trajectory intersects with two parallel boundaries or two adjacent boundaries. The different exit strategy will be given according to how the trajectory is intersected with the restricted area.

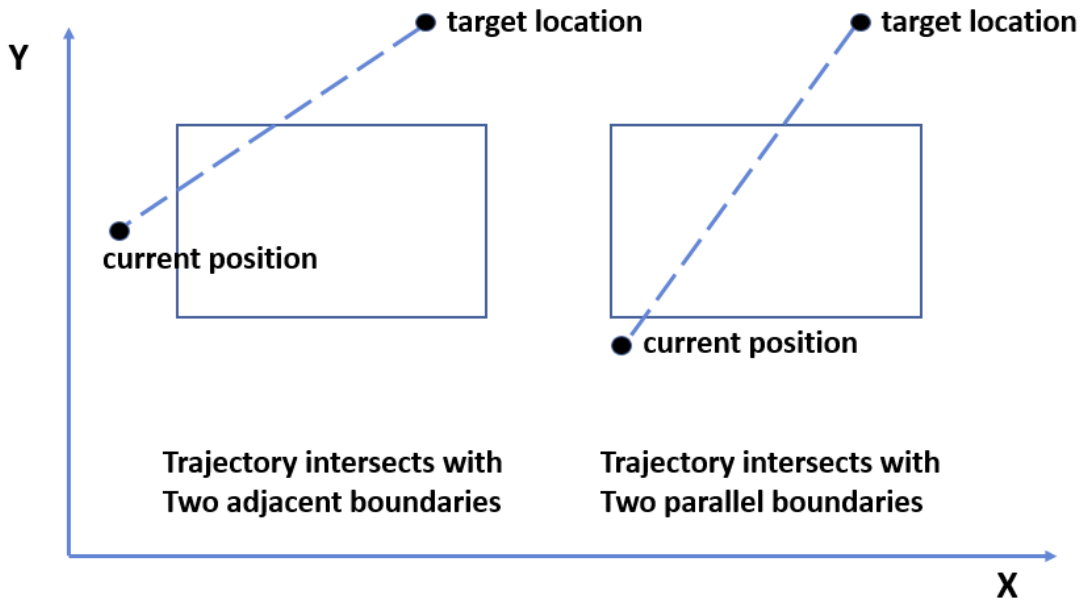



Figure 4.13: Two scenarios for the trajectory intersects with the bounding box

Figure 4.14 illustrated the scenarios when the trajectory intersects with two parallel boundaries ($x_{min}^{boundary}$ and $x_{max}^{boundary}$). In order to exit the restricted area, the algorithm will calculate and compare the distance of $d1$ and $d2$ from the UAV to $x_{min}^{boundary}$ and $x_{max}^{boundary}$ boundaries respectively, the algorithm will move only in $x$ axis according to the shortest distance's direction. If the two distances are happened to be equal, the algorithm will repeat the previous step by comparing the distance from target to $x_{min}^{boundary}$ ($d3$) and $xx_{max}^{boundary}$ ($d4$) boundaries and chose the exit corner respectively. Additionally, the desired yaw will be changed to either 0 or 180 degrees for the sensor to get a view of the surrounding environment for path planning and collision avoidance.

**NOTE:** There are two unique scenarios that the line function method could not solve the problem they are when $x1 = x2$ and $y1 = y2$, which the denominator of the function is zero and will lead the program to crash. In order to cope with this, the algorithm will compare the $x1, x2, y1, y2$ coordinate value at the beginning, and treat them as the two parallel boundaries cases, with the only exception it is only required to compare which

corner is close to the current point, and select the more intimate way to exit the restricted area.

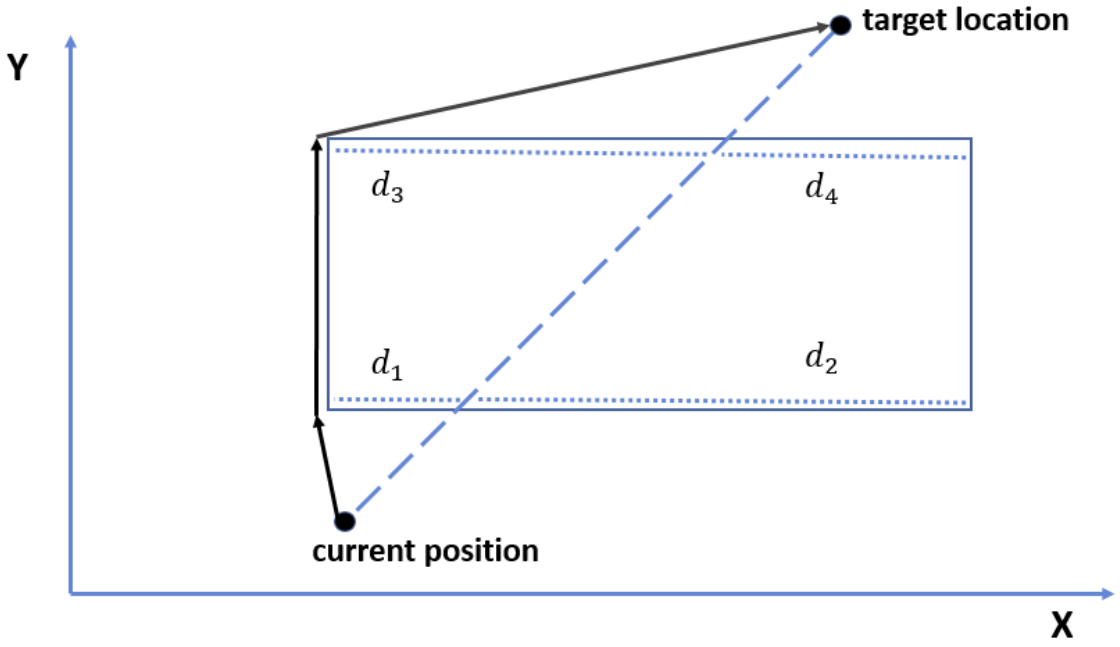



Figure 4.14: Exit strategy for parallel boundaries

For the scenarios where the trajectory intersects with two adjacent boundaries, the algorithm will always choose to exit the rectangle by moving towards to the insection point between those two adjacent boundaries (Figure 4.15) for a fast exit, along with appropriate yaw angle to obtain sensory information for the mapping system.

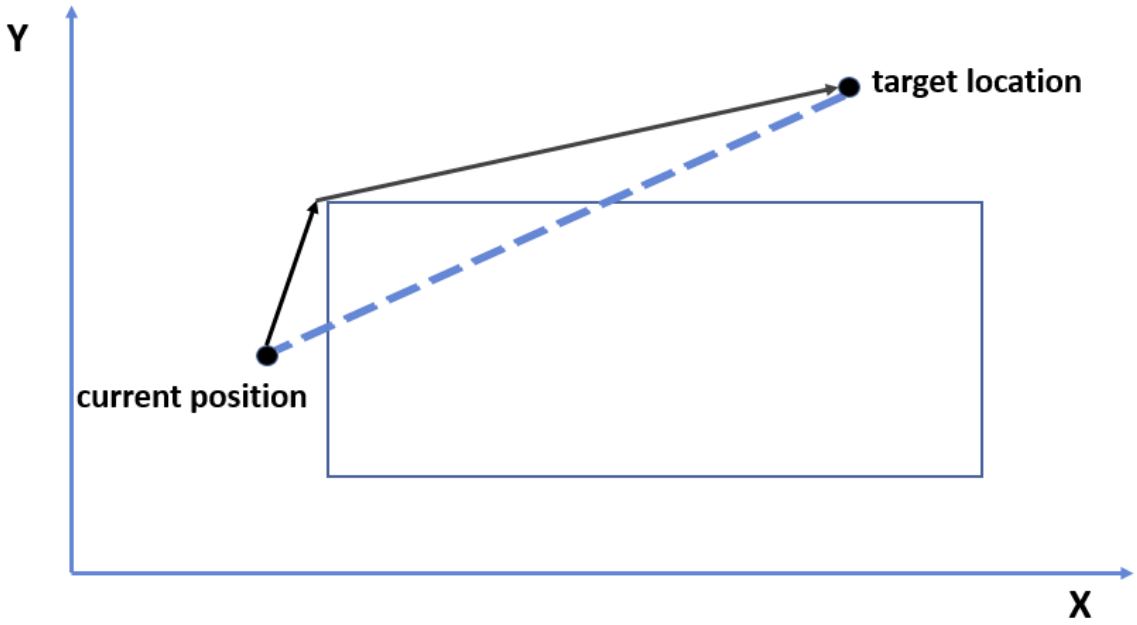



Figure 4.15: Exit strategy when intersect with two adjacent boundaries

### 4.3.2 Static obstacle avoidance

This section contains the implementation steps for static obstacle avoidance. Figure 4.16 gives an overview of the implementation for the static obstacle avoidance. As it is illustrated, the algorithm will start by comparing its current odometry with the target coordinate. It hovers at the desired if the detected odometry from the localisation system is within one-meter range of the desired position in both $x, y, z$ axis. Then it will search for the corresponding next intermediate waypoint to keep a safe distance from the restricted area if the UAV is within the safe distance from the restricted area proposed in Chapter 4.3.1. When the UAV is relatively far from the restricted area, the navigation system will search for a next intermediate waypoint based on the UAV's current location and the target location, and either transfer the waypoints to the control system to execute the navigation command or re-calculate for a collision-free waypoint based on the feedback from the collision checking system. The following paragraph includes the implementation of how to determine the UAV's next waypoint and avoid the obstacle (s) when a potential collision is detected. The task can be roughly broken down into three main steps.

- Calculate next waypoint according to the UAV's current odometry and pre-defined target.
- Check potential collision for the initially proposed waypoint.
- Find an alternative waypoint if the initially proposed waypoint is in a potential collision.

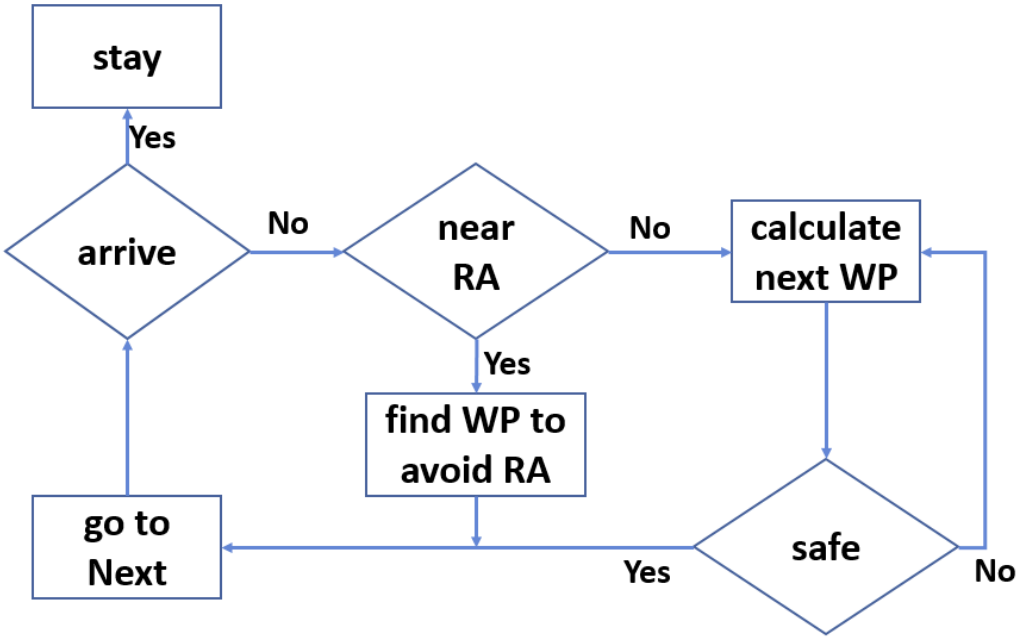



Figure 4.16: Flowchart for the implemented static obstacle avoidance

**Calculate next waypoint**

The next waypoint is calculated based on the target coordinate and UAV's state information, which are the UAV's coordinate in x, y, z-axis and yaw angle (where the camera is facing). For example, to calculate the next waypoint in x-axis, the algorithm starts by calculating the difference between the target point and current odometry in x-axis. The UAV will stay at the current x-axis value if the difference is less a buffer distance, to compensate the system errors. Ideally, the buffer distance should be determined by the total errors from both localisation, mapping and control system, due to limited time constraint, the buffer distance is set to be 0.5 meter. After finding the difference between the UAV and the target location, the algorithm will increase or decrease the UAV's odometry by one meter according to the difference, the choice of the next waypoint step size is discussed and evaluated in Chapter 6.1. The pseudo-code is given as follows:

- $diff_x = target_x - current_x$

- $next_x = current_x + 0$ if $|diff_x| \leq 0.5$
- $next_x = current_x + 1$ if $diff_x > 0.5$
- $next_x = current_x - 1$ if $diff_x < 0.5$

Likewise, to get the next waypoint value in y and z-axis. In order for the camera on UAV to get a map for its surrounding environment, as it is shown in Figure 4.17, the next yaw is calculated by:

$$next_{yaw} = \arctan\left(\frac{target_y - next_y}{target_x - next_x}\right) \tag{4.17}$$

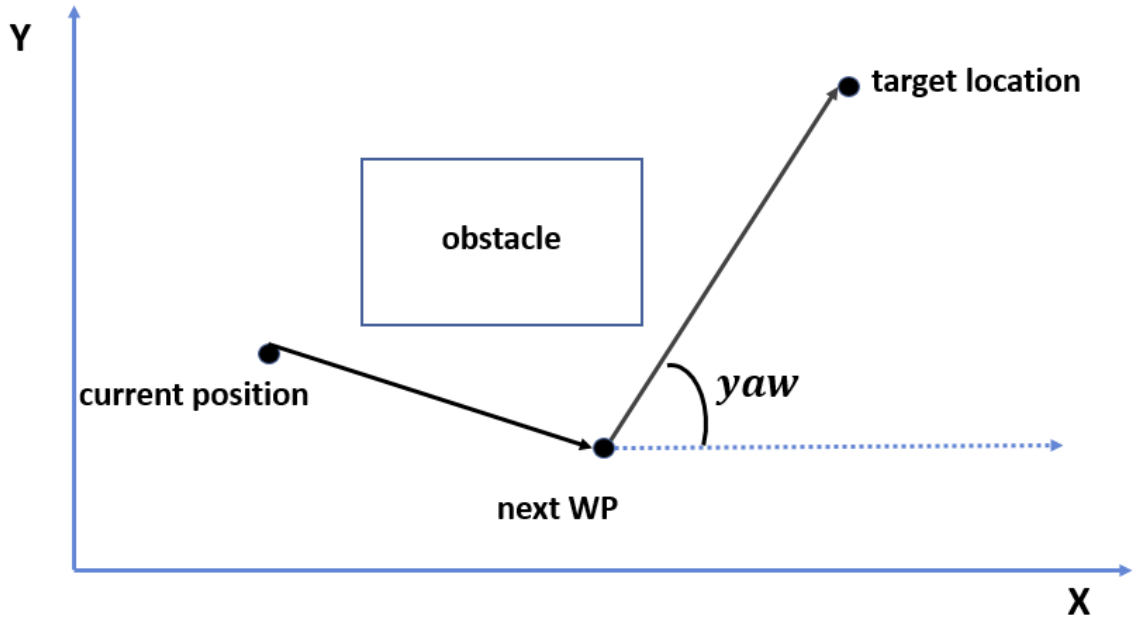



Figure 4.17: Calculation for the yaw angle

**Collision checking**

After finding a suggested waypoint to the target, it is essential to find out whether there are any obstacles between UAV's current and the suggested next waypoints. Which can

be done by running an iterator to traverse all leaf node within a given axis-aligned bounding box, the UAV will execute the suggested waypoint if there is no obstacle within the searched bounding box. Otherwise, it will search for an alternative route if the bounding box is occupied or unknown. Hence, the problem now lies in how to select an appropriate bounding box from the UAV's current odometry and next waypoint.

Figure 4.18 shows a 2D representation of the UAV model, it is represented by a rectangle with its maximum length $l$ and width $w$, to achieve collision-free navigation, the bounding box region needs to have at least $UAV_l$ length and $UAV_w$ width in order for the UAV to pass if the UAV is travelling without changing the yaw angle. However, for the cases when the UAV changes its heading direction, the size of the bounding box will be constrained by the diagonal length of the UAV's 2D model. Thus, to constructed a squared shape bounding box with the length of $\sqrt{UAV_l{}^2 + UAV_w{}^2}$. Lastly, the bounding box should also contain the compensation from UAV's system error in each axis, which includes the error from localisation, mapping, and control system. Therefore to calculate the length $bbx_l$ of the bounding box in 3D by:

$$\begin{aligned} UAV_{diagonal} &= \sqrt{UAV_l{}^2 + UAV_w{}^2 + UAV_h{}^2} \\ bbx_l &= UAV_{diagonal} + e_m + e_l + e_c \end{aligned} \tag{4.18}$$

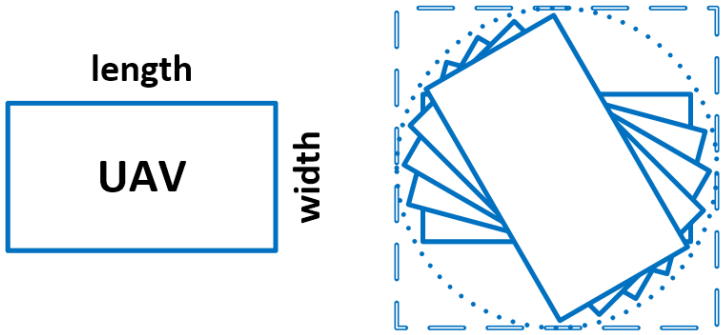



Figure 4.18: Determining the minimum length required for the bounding box- 2D overview

where $e_m$, $e_l$ and $e_c$ denote maximum system error from mapping and localisation and control system respectively, $UAV_l$, $UAV_w$ and $UAV_h$ denote the length, width and height of the 3D model of the UAV. Therefore, the minimum and maximum of the bounding box in $x$ axis can be calculated by Equation 4.19, and the maximum and minimum value of the bounding box in $y$ and $z$ axis can be determined by the same method, due to limited time constraint, the length of the cubical bounding box is set to be a fixed distance with a minimum length of $bbx_l$, the impact of the choice of the fixed distance is evaluated and discussed in Chapter 6.1.

$$\begin{aligned} bbx^x_{min} &= next_x \quad \textbf{if} \quad next_x < current_x \\ bbx^x_{min} &= current_x \quad \textbf{if} \quad next_x > current_x \\ bbx^x_{max} &= next_x + \frac{bbx_l}{2} \quad \textbf{if} \quad next_x > current_x \\ bbx^x_{max} &= current_x + \frac{bbx_l}{2} \quad \textbf{if} \quad next_x < current_x \end{aligned} \tag{4.19}$$

**ALternative waypoint searching**

After any potential collision is detected, it is essential to find an alternative collision-free waypoint to reach the target by avoiding the obstacle. Which is achieved by dividing the UAV's near-space into six bounding boxes in both positive and negative direction of x, y, z-axis respectively, with the minimum length $bbx_l$ derived from Equation 4.18 in order for the UAV to pass. Hence to find an obstacle-free bounding box to avoid the detected obstacle while navigating to the target. Figure 4.19 shows a 2D overview of the four bounding boxes in the x-y panel. As it is illustrated, the bounding box is always axis aligned regardless of the coordinate from current odometry and target. This is because the UAV's sensor now needs to be pointed to the obstacle instead of the target for a better perception of the obstacle rather than the full environment.

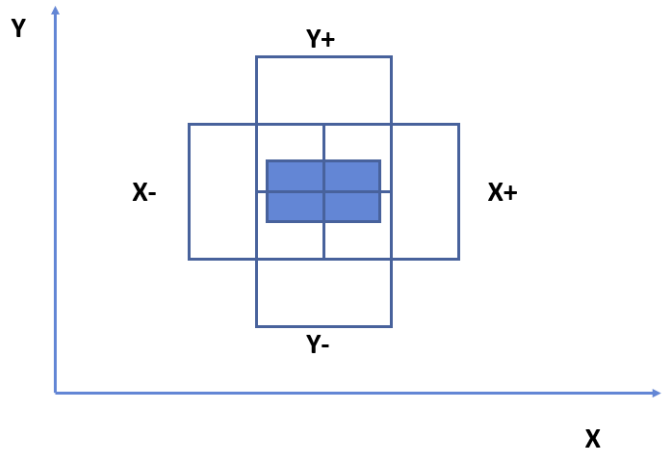



Figure 4.19: 2D overview of the bounding box division based on the UAV's current location

As the working environment for the UAV is outdoor, it is reasonable to assume the UAV is more likely to avoid collision by going over the obstacle. Hence, the bounding box in the positive z-axis is chosen by default to be iterated, and the UAV will travel in a positive direction with a fixed step of one meter until pass the maximum height of the obstacle if the bounding box is determined to be safe. For the cases when the 'up' space is occupied, the algorithm will order each bounding box that will be checked based on the target location and UAV's current location, then check each bounding box in a particular order to search for available safe space to avoid the obstacle. The process is broken down into the following steps.

- determine the order for the bounding boxes need to be checked.
- check the bounding box in order, manoeuvre to the bounding box when an obstacle-free space is found.

In order to select the bounding box, The UAV will first calculate the length to the target in x and y-axis respectively. Then chose the first two bounding box from $x_{pos}, x_{neg}, Y_{pos}, Y_{neg}$ according to the resulting sign (positive or negative) of the length in both x and y-axis. For the next step, the algorithm will order the two chosen bounding box

from the previous step, based on the distance from UAV's current position to the target location in both x and y-axis. The order of the two bounding boxes will be evaluated based on the future distance of the corresponding bounding box to the target location, with the shorter distance one being evaluated first and longer distance one being evaluated second. The remaining two bounding boxes can be put into any order lastly.

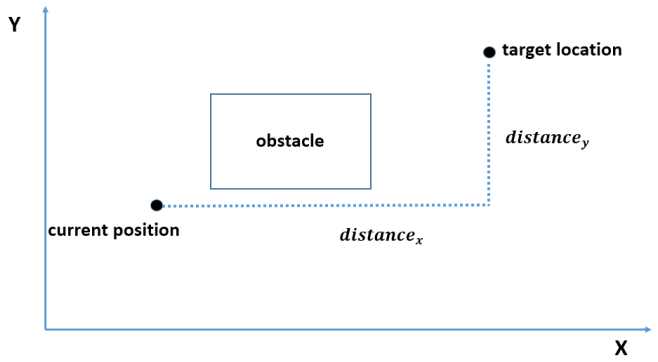



Figure 4.20: Select the appropraite bounding box

Figure 4.20 shows an example with given coordinate for current odometry and target, with the green shaped region being the obstacle, first to calculate the length in both the x, y-axis:

$$\begin{aligned} distance_x = target_x - UAV_x = \textbf{ a positive value} \\ distance_y = target_y - UAV_y = \textbf{ a positive value} \end{aligned} \tag{4.20}$$

with both $distance_x$ and $distance_y$ being positive, it can be determined that $x_{pos}$ and $Y_{pos}$ bounding boxes will be evaluated firstly, and $x_{neg}$ $Y_{neg}$ being evaluated lastly at any random order. The algorithm now will determine the correct order between the first two bounding box based on the future waypoint distance to the target location:

$$\begin{aligned} |distance_x| = \textbf{ a positive value} \\ |distance_y| = \textbf{ a smaller positive value} \end{aligned} \tag{4.21}$$

Therefore, with a smaller length to the target location, the $Y_{pos}$ bounding box is chosen to be the first one.

### 4.3.3 Dynamic obstacle avoidance

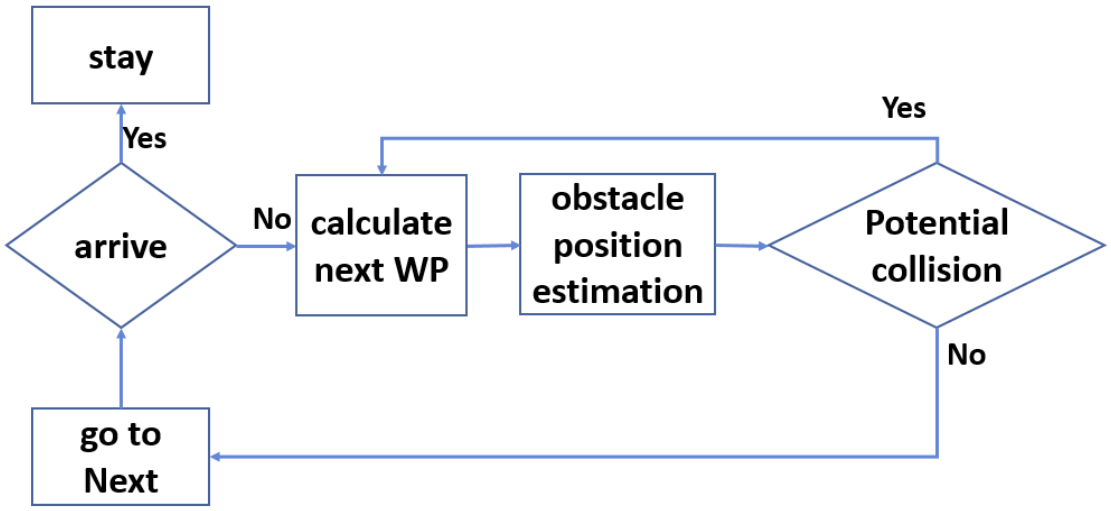



Figure 4.21: Flowchart for dynamic obstacle avoidance

For dynamic obstacle avoidance, the UAV is required to reach a randomly assigned target location by avoiding dynamic obstacle which the UAV has no prior knowledge of its movement trajectory. In order to do this, the UAV needs to constantly be aware of the position of the moving obstacle and hence chose an appropriate path by predicting the obstacle's future location based on its historical movement trajectory. This section includes the design and implementation of a dynamic obstacle avoidance system with only one moving obstacle. Figure 4.21 shows an overview of the methodology used for dynamic obstacle avoidance. The algorithm starts with calculating a waypoint (as is implemented for static obstacle avoidance in Chapter 4.3.2) according to the UAV's current odometry and the assigned target, then check for a potential collision based on the predicted range of obstacle's future position. Lastly, the algorithm will either pass the command to the control system to manoeuvre the UAV to the proposed waypoint or calculate another one based on the result from the collision checking system. This section will focus on the details of implementation for the following tasks:

- obstacle trajectory estimation.

- potential collision checking.
- alternative next waypoint searching.

### Obstacle trajectory estimation

With access to information of the obstacle's historical position, the moving obstacle's velocities $(v_x^o, v_y^o, v_z^o)$ in each axis can be estimated with the obstacle's displacement $(d_x^o, d_y^o, d_z^o)$ by comparing the obstacle's current position and its historical position at $t$ second (s) ago. Equation 4.22 illustrates the mathematical model for obstacle's velocity estimation, where $(x_{now}^o, y_{now}^o, z_{now}^o)$ are the obstacle's current positions and $(x_{now-t}^o, y_{now-t}^o, z_{now-t}^o)$ are obstacle's historical position at $t$ second (s) ago. The choice of how duration $t$ affects the performance of the dynamic ostacle avoidance algorithm is discussed and evaluated in Chapter 6.2.

$$\begin{bmatrix} d_x^o \\ d_y^o \\ d_z^o \end{bmatrix} = \begin{bmatrix} x_{now}^o \\ y_{now}^o \\ z_{now}^o \end{bmatrix} - \begin{bmatrix} x_{now-t}^o \\ y_{now-t}^o \\ z_{now-t}^o \end{bmatrix}$$
$$\begin{bmatrix} v_x^o \\ v_y^o \\ v_z^o \end{bmatrix} = \begin{bmatrix} d_x^o \\ d_y^o \\ d_z^o \end{bmatrix} \div \begin{bmatrix} t \\ t \\ t \end{bmatrix} \tag{4.22}$$

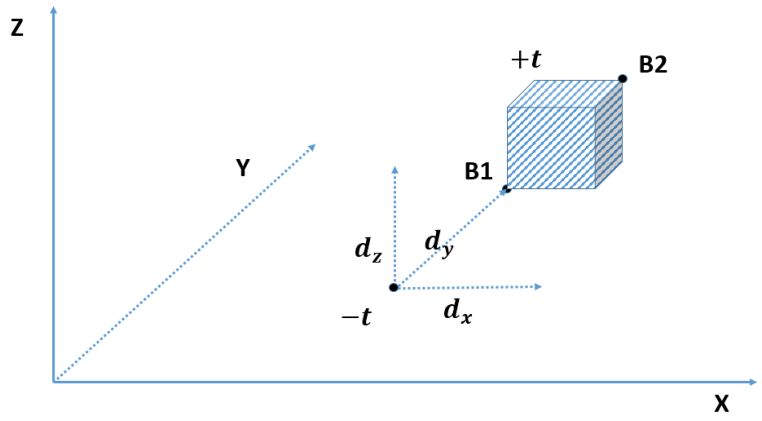



Figure 4.22: Prediction of obstacle's future position range

$$B_{min} = \begin{bmatrix} x^o_{now} - x_e \\ y^o_{now} - y_e \\ z^o_{now} - z_e \end{bmatrix}$$
$$B_{max} = \begin{bmatrix} x^o_{now} + v^o_x \times t^u + x_e + x^o_a \\ y^o_{now} + v^o_y \times t^u + y_e + y^o_a \\ z^o_{now} + v^o_z \times t^u + z_e + z^o_a \end{bmatrix} \quad (4.23)$$

With the estimated obstacle's velocity in each axis, it is possible to predict the obstacle's range of position for the next $t^u$ second, where $t^u$ denotes the time required for the UAV to reach the initially proposed waypoint. As it is shown in Figure 4.22, the algorithm predicts the obstacle's position range at time $+t^u$ based on its displacement from its position historical position at time $-t$ and its current position. Besides the displacement generated by the obstacle's movement $(v^o_x \times t^u, v^o_y \times t^u, v^o_z \times t^u)$, the range is extended with two additional lengths $[x_e, y_e, z_e]$ and $[x^o_a, y^o_a, z^o_a]$, where $[x_e, y_e, z_e]$ accounts for the additional space required due to UAV's rotation and system errors from mapping, localisation and control sub-systems (as it is described in Equation 4.18 at Chapter 4.3.2) and half of the obstacle's length in each axis for the special scenarios when the velocity is zero in any of the three axes, $[x^o_a, y^o_a, z^o_a]$ accounts for the additional space required due to the change

of speed from the obstacle[5]. Equation 4.23 shows the minimum and maximum value for the range of predicted obstacle's position at time $t^u$, assuming the obstacle is travelling in positive in both x, y, z-axis.

### Collision checking

By constructing a 3D cubical bounding box to represent the possible range of the obstacle's future position within the time interval of $t_u$, the potential collision can be detected by determining whether the initially proposed trajectory is in interference with the constructed bounding box. The initially proposed trajectory for the next interval is represented by $P1, P2$, where $P1$ denotes the UAV's current position, and $P2$ denotes the proposed next waypoint.

$$P1 = \begin{bmatrix} x^u_{now} \\ y^u_{now} \\ z^u_{now} \end{bmatrix} \qquad P2 = \begin{bmatrix} x^u_{next} \\ y^u_{next} \\ z^u_{next} \end{bmatrix} \tag{4.24}$$

With the given bounding box representing the obstacle's predicted position range and the proposed trajectory for UAV. In order to determined whether the UAV is in potential collision with the moving obstacle, the algorithm will first check if $P1$ $P2$ are entirely on one side of the bounding box so that it can return false quickly. This is implemented by comparing $P1, P2$ value with the maximum and minimum value of the bounding box in each axis, if both $P1, P2$ are greater the maximum (or smaller than the minimum) value of the bounding box in any one of the three axes, the trajectory is considered to be reside outside the bounding box and hence collision-free.

[5] increase of speed, to be more specific.

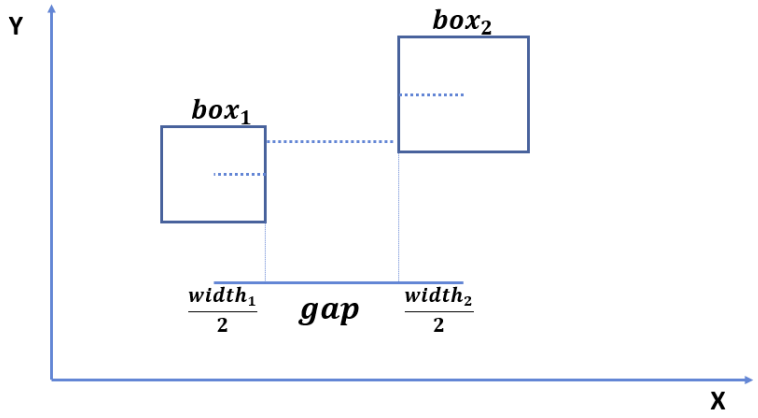



Figure 4.23: Overview of Seperating Axis Theorem in 2D

When both $P1, P2$ do not reside on the same side of the bounding box, the algorithm will apply Separating Axis Theorem (SAT) to check whether the trajectory is in the box. The theorem states that if it is possible to draw a line to separate two polygons, then they do not collide. Figure 4.23 shows an overview of the theorem in 2D space. Where the box1 is the bounding box for predicted obstacle position range, P2 is the second bounding box constructed with UAV's trajectory $P1, P2$. The methodology is to find the gap between box 1 and box2 to determine whether the gap is big enough for a line to pass through. The example pseudo-code to evaluate the separation in the x-axis is given as follows.

- $x_{center}^{box1} = \frac{1}{2}(x_{max}^{box1} - x_{min}^{box1})$
- $x_{center}^{box2} = \frac{1}{2}(x_{max}^{box2} - x_{min}^{box2})$
- $length = x_{center}^{box2} - x_{center}^{box1}$
- $width1 = x_{max}^{box1} - x_{min}^{box1}$
- $width2 = x_{max}^{box2} - x_{min}^{box2}$
- $gap = length - \frac{width1}{2} - \frac{width2}{2}$
- if $gap < 0$, boxes are intersecting each other, potential collision detected.

- else if $gap = 0$ boxes are next to each other, safe to travel
- else, there is a gap between boxes, safe to travel.

where $x^{box1}_{center}, x^{box2}_{center}$ are the geometrical center of these two box in x-axis, $length$ is the distance between those two centers, $width1, width2$ denotes the length of each box in x-axis. The gap is calculated by substracting half-length of box1 and box2 from the length between them. Ideally, it is only safe for the UAV to travel if the gap is greater than zero. However, the bounding box is constructed with the adjustment of UAV's diagonal length and its system error, and the algorithm will treat collision-free when the gap is zero. Lastly, the theorem states the trajectory does not collide with the bounding box area if there is a gap in any of the three axes, the algorithm only reports potential collision if there is no gap in any axis.

**Alternative waypoint searching**

When the initially proposed trajectory is determined to be in potential collision with the moving obstacle, before searching for an alternative waypoint, the algorithm will first to check whether the target location is safe to travel, to cope with the special case that the obstacle is moving around the target position. The algorithm will trigger the UAV to hover at its current location to wait for the obstacle moving away, instead of assigning any next waypoint. Otherwise, the algorithm will search for an alternative collision-free trajectory by assigning a waypoint towards one of the four corners of the constructed bounding box in 2D. The waypoint in z-axis is determined based on the relationship between the estimated obstacle velocity $v^o_z$ and the direction from UAV towards the target location $d^u$. The feasibility and results of the algorithm are discussed and evaluated in Chapter 6.2.

## 4.4 Control System

This section describes the design and implementation of the control system, which is, manoeuvre the UAV to the desired location with the desired orientation.

### 4.4.1 UAV dynamics

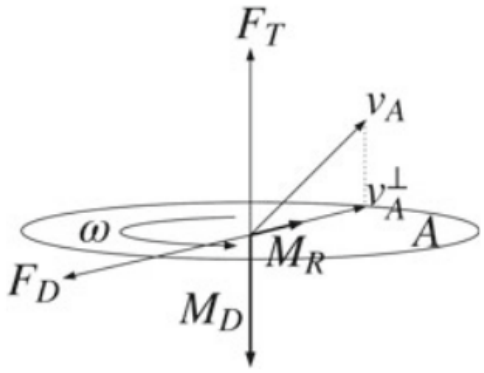



Figure 4.24: Forces and moments can be divided into: $F_T$ thrust force, $F_D$-drag force, $M_R$ rolling moment, $M_D$ the moment originating from the drag of a rotor [9]

Before the design of the control system, it is essential to find the relationship between rotors speed and UAV's translational speed. The forces and moments applied to the UAV can be separated into individual ones applied on every rotor blade, along with the gravity applied to the UAV's CoG. The full dynamics of the UAV can be calculated by combining all these forces and moments together. Figure 4.24 shows different forces and moments acting on a single rotor, which leads to the following equation analysed by Martin [103].

$$F_T = \omega^2 C_T \times e_{ZB} \tag{4.25}$$

$$F_D = -\omega C_D \times v_A^{\perp} \tag{4.26}$$

$$M_R = \omega C_R \times v_A^{\perp} \tag{4.27}$$

$$M_D = -\varepsilon C_M \times F_T \tag{4.28}$$

where:

$\omega$ is the rotor's positive angular velocity.

$C_T$ is a constant describes the rotor thrust

$C_D$ is a constant describes the rotor drag

$C_R$ is a constant describes the rolling moment c

$C_M$ is a constant describes the rotor moment

$\varepsilon$ is the rotor's turning direction (+1 for counter clockwise and -1 for clockwise).

$e_{ZB}$ is the unit vector pointing to $z$ according to the UAV's coordinate.

Therefore it is possible to derived the projection of the vector $u$ onto the rotor plane by:

$$u^{\perp} = e_{ZB} \times (u \times e_{ZB}) = u - (u \times e_{ZB}) \times e_{ZB} \tag{4.29}$$

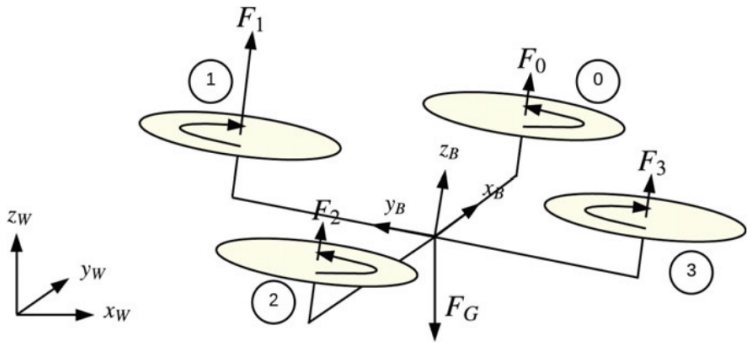



Figure 4.25: Forces and moments acting on the UAV [9]

Figure 4.25 shows a quadrotor with four mounted rotors, represented in UAV's own coordinate system $B$ and world coordinate system. $W$. The total forces consist of the combined rotors forces $F_i$ and gravitational force $F_G$. In order to derive the model dynamics for a UAV with $n$ rotor, first to apply Newton's Law to derive the UAV's motion and Euler equation:

$$F = m \times a \tag{4.30}$$

$$\tau = J \times \dot{\omega} + \omega \times J \times \omega \tag{4.31}$$

Where the linear part (equation 4.30) is expressed in the world's coordinate system, and the rotational part (Equation 4.31) is represented in the UAV's coordinate system. For each of the parameters, $m$ is the mass of the UAV, $a$ is the acceleration, $J$ is the inertial matrix, and $\omega$ is the angular velocity. Hence, for an UAV with $n$ rotors, equation 4.30 and 4.31 can be represented by:

$$F = m \times a = \sum_{n-1}^{i=0} (R_{WB}(\underbrace{F_{T,i} + F_{D,i}}_{\text{Fi}})) + F_G \tag{4.32}$$

$$\tau = J \times \dot{\omega} + \omega \times J \times \omega = \sum_{n-1}^{i=0} (M_{R,i} + M_{D,i} + F_i \times r_i) \tag{4.33}$$

$R_{WB}$ is the rotation matrix derived from the UAV's coordinate system $B$ translation into the world's coordinate system $W$, $r_i$ is the vector from the UAV's CoG to $i^{th}$rotor's CoG.

### 4.4.2 State representation

As it is shown in Figure 4.26, the UAV's states are represented by separating the controller into an outer loop tracks its position, and an inner loop tracks its attitude. In the position loop, the UAV's velocity ($v$) and position ($p$) are represented in the world's coordinate

system. Meanwhile, the inner loop is tracking the UAV's angular velocity ($\omega$) and orientation, which are represented in the UAV's coordinate system. Combine these with IMU's bias states leads the state vector in Equation 4.34:

$$x = [p^T v^T \bar{q}^T b_a^T b_\omega^T]^T \tag{4.34}$$

As the system is independent of the dynamic and model parameter of the specific vehicle, the measurement will not be affected by the UAV's dynamics. Furthermore, remove the angular rate from the state vector. Hence, the model dynamic can be described as in Equation 4.35, where $g$ is gravity in the world coordinate system, $C$ is the rotation matrix computed from $\bar{q}$.

$$\begin{aligned}
\dot{p} &= v \\
\dot{v} &= g + C \times (a_m - b_a - n_a) \\
\dot{\bar{q}} &= \frac{1}{2}\bar{q} \bigotimes [ \begin{pmatrix} & 0 & \\ \omega_m & -b_\omega & -n_\omega \end{pmatrix} ] \\
\dot{b_a} &= n_a \\
\dot{b_\omega} &= n_\omega
\end{aligned} \tag{4.35}$$

Therefore, the UAV's state estimation can be separated into position measurement ( $p_m$) and attitude measurement ($\bar{q_m}$), which express the measured state of the IMU in the world coordinate system. The equations are given by:

$$p_m = p + n_p \tag{4.36}$$

$$\bar{q_m} = \bar{q} \otimes \delta\bar{q_n} \tag{4.37}$$

where $\otimes\delta\bar{q_n}$ is a small rotation error and $n_p$ consists zero-mean white Gaussian. How-

ever, the implemented model is only valid when the origin of the pose-sensor occurs at the same time as the IMU. Additionally, the pose sensor measurements are not guaranteed to be aligned with the world axis. However, Those misalignments still can be compensated as they are often observable. Therefore, no prior calibration is required. The derivation process was referred to Achtelik's work in [104].

### 4.4.3 Controller mechanism

In order to control the UAV, it is essential to find out the relation between the input and output for the system. The output $T$ is the accumulated thrust from each rotor, and $\tau$ is the torque applied on the UAV's CoG, where the input is the accumulated angular velocity $\omega$. Hence the control system can be formulated by:

$$\begin{pmatrix} T \\ \tau \end{pmatrix} = A \times \begin{pmatrix} \\ \omega_2^0 \\ \omega_2^1 \\ . \\ . \\ . \\ \omega_2^n \end{pmatrix} \tag{4.38}$$

For a quad-rotor UAV shows in Figure 4.25, the allocation matrix $A$ is given by:

$$\begin{pmatrix} \\ C_T & C_T & C_T & C_T \\ 0 & lC_T & 0 & -lC_T \\ -lC_T & 0 & lC_T & 0 \\ -C_TC_M & C_TC_M & -C_TC_M & C_TC_M \end{pmatrix} \tag{4.39}$$

In this simulator, a geometric control approach proposed by Lee is chosen to directly calculate the thrust and moments required, with different levels of commands, such as

position, angular rate, or orientation [10]. It is possible to produce a thrust $T$ with the same direction of the rotor's normal vector by looking all rotor axis-aligned same, assuming it coincides with $Z_B$. Therefore, only the thrust $T$ around all three body axis $x_B$, $y_B$ and $z_B$ can be directly controlled. In order for the controller to work in a 3D environment, it is necessary to move the UAV into a slopping position as a setpoint. Hence to produce thrust in the direction of $Z_B$ and the yaw rate, which is usually called the attitude controller. A cascaded control method is implemented because the translation change is normally much slower than the change of the attitude. The position loop operates at a lower rate while calculating the desired thrust and attitude. The attitude operates onboard a microcontroller at a higher rate. As it is illustrated in Figure 4.26, the controller is separated into two parts: an outer loop controls the UAV's position, and an inner loop controls the UAV's attitude.

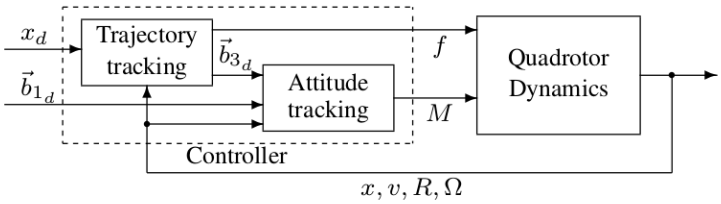



Figure 4.26: Overview for the controller [10]

# Chapter 5

# Evaluation of spatial awareness

## 5.1 localisation with EKF

This section includes the experiment designed to evaluate the performance of the state estimation ability from the ekf_localisation_node. Although orientation data is required as input for the EKF within robot_localisation package, due to the limitation of the implemented position controller, which can only achieve four DoF tracking of the UAV, namely three position variables and one heading direction, this experiment focus only on the performance of the position estimation from different sensors combinations. In the experiment, the UAV is designed to follow a pre-defined waypoints (Table 5.1) by using waypoint_publisher_file node from the Rotors simulator, where $(x, y, z)$ indicates the position, $T$ is the waiting time at each waypoint, $YAW$ is the heading direction of the UAV.

As it is shown in Table 5.2, the fusion integrates sensory information from generic odometry sensor and IMU sensor. The odometry sensor is designed to mimic a GPS, by providing position, linear velocity and angular velocity separately, in the actual experiment, the ekf_localisation node treats the odometry sensor as a single sensor and returns unstable output. Hence the configuration for the ekf_localisation node is reconfigured as the odometry sensor providing linear velocity and imu providing linear acceleration (Table 5.2)[1]. The estimated result is compared with the output trajectory from an ideal

[1]Imu also provides angular velocity state, to allow the proper function of the ekf_localisation node, no

odometry sensor without any noise implemented.

Table 5.1: Executed waypoints for EKF localisation

| T | X | Y | Z | Yaw |
|---|---|---|---|---|
| 1.0 | 0.0 | 0.0 | 1.0 | 0.0 |
| 1.0 | 1.0 | 1.0 | 2.0 | 0.0 |
| 1.0 | 2.0 | 2.0 | 3.0 | 0.0 |
| 1.0 | 3.0 | 3.0 | 4.0 | 0.0 |
| 1.0 | 4.0 | 4.0 | 5.0 | 0.0 |
| 1.0 | 5.0 | 5.0 | 6.0 | 0.0 |
| 1.0 | 6.0 | 6.0 | 7.0 | 0.0 |
| 1.0 | 7.0 | 7.0 | 8.0 | 0.0 |
| 1.0 | 8.0 | 8.0 | 9.0 | 0.0 |
| 1.0 | 9.0 | 9.0 | 10.0 | 0.0 |
| 1.0 | 10.0 | 10.0 | 11.0 | 0.0 |

Table 5.2: EKF sensor input states configuration (T:True F:False)

| | $x$ | $y$ | $z$ | $roll$ | $pitch$ | $yaw$ | $x'$ | $y'$ | $z'$ | $roll'$ | $pitch$' | $yaw'$ | $x''$ | $y''$ | $z''$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| imu config | F | F | F | F | F | F | F | F | F | T | T | T | T | T | T |
| odom config | F | F | F | F | F | F | T | T | T | F | F | F | F | F | F |

Table 5.3 lists seven different scenarios designed to test the performance of ekf_localisation node. those configuration includes:

- fuse single IMU with single odometry
- fuse single IMU with two odometries
- fuse two IMUs with single odometry
- fuse two IMUs with two odometrys

orientation data is evaluated

As EKF works on the principle of trusting more on the input data with less noise, extra experiments were implemented to test the ekf_localisation node's performance with the same sensor combination but different sensor parameters. In case $No.2,3,4$, both IMU0, IMU1 and odometry0, odometry1 are implemented with identical noise covariance, In case $NO.4,5,6$, the IMU and odometry are implemented with different noise covariance. The corresponding noise parameters in each axis are given in Table 5.3, where the imu indicates the linear acceleration noise covariance, and odometry indicates the linear velocity noise covariance.

Table 5.3: Sensor combination for EKF fusion

| **Case** *No* | **Sensor combination** | **Noise covariance** | | | | |
|---|---|---|---|---|---|---|
| | | | | x | y | z |
| 1 | 1 imu 1 odometry | | imu0 | 0.004 | 0.004 | 0.004 |
| 2 | 1 imu 2 odometry | identical | imu1 | 0.004 | 0.004 | 0.004 |
| 3 | 2 imu 1 odometry | | odom0 | 0.0012 | 0.0012 | 0.003 |
| 4 | 2 imu 2 odometry | | odom1 | 0.0012 | 0.0012 | 0.003 |
| 5 | 1 imu 2 odometry | | imu0 | 0.004 | 0.004 | 0.004 |
| 6 | 2 imu 1 odometry | different | imu1 | 0.003 | 0.003 | 0.003 |
| 7 | 2 imu 2 odometry | | odom0 | 0.0012 | 0.0012 | 0.003 |
| | | | odom1 | 0.0016 | 0.0008 | 0.002 |

It takes approximately 18 seconds in simulation for the UAV to reach the target, in which there are approximately 5 seconds for ROS to initialise time, to load the corresponding parameters for the UAV's sub-systems. The total simulation time is approximately 22 seconds for each scenario. The data was collected with rosbag by monitoring two following ROS topics: 1)./firefly/ground_truth/position; 2)./firefly/odometry/filtered. The first one is provided by an ideal odometry sensor, and the second one is the estimated odometry data from the ekf_localisation node. Both topics contain the UAV's executed trajectory with respect to the simulation time in 3D. For each scenario, there are approx-

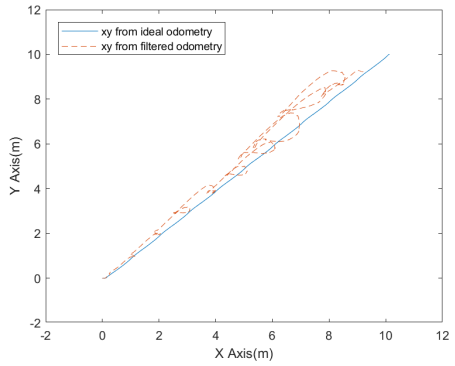
imately 2000 recorded messages for both of the odometry data. Another ROS node was written to monitor both of the odometry data and calculate the difference by substrating the filtered odometry from the ideal odometry in each axis, to evaluate the difference between the ideal odometry and the filtered odometry,

Table 5.4: EKF error

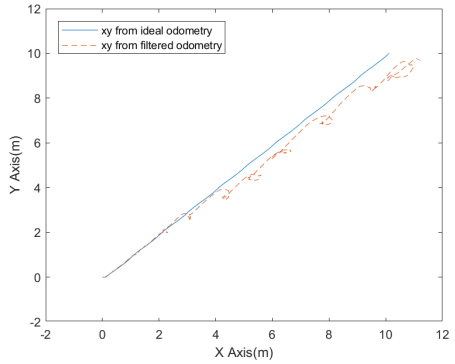
| **CASE** *No.* | **max(m)** **x y z** | **min(m)** **x y z** | **mean(m)** **x y z** | **standard deviation** | **close error(m)** **x y z** |
|---|---|---|---|---|---|
| | 2.56 | -0.06 | 0.58 | 0.68 | 0.95 |
| 1 | 1.82 | -0.53 | 0.37 | 0.47 | 0.77 |
| | 1.04 | -22.29 | -0.35 | 0.67 | -0.77 |
| | 0.75 | -1.79 | -0.22 | 0.52 | -1.074 |
| 2 | 1.30 | -0.67 | 0.23 | 0.35 | 0.303 |
| | 4.62 | -0.54 | 0.74 | 1.07 | 2.197 |
| | 0.67 | -1.45 | -0.28 | 0.55 | -1.280 |
| 3 | 1.76 | -0.23 | 0.41 | 0.50 | 0.961 |
| | 2.03 | -1.11 | 0.27 | 0.52 | 0.929 |
| | 0.40 | -1.85 | -0.49 | 0.73 | -1.705 |
| 4 | 0.30 | -1.74 | -0.41 | 0.55 | -1.244 |
| | 6.13 | 0.00 | 1.65 | 1.98 | 4.831 |
| | 0.43 | -1.57 | -0.23 | 0.42 | -0.872 |
| 5 | 1.42 | -0.47 | 0.27 | 0.41 | 0.826 |
| | 2.46 | -0.79 | 0.26 | 0.44 | 0.426 |
| | 1.81 | -0.50 | 0.31 | 0.44 | -0.038 |
| 6 | 2.35 | -0.03 | 0.51 | 0.57 | 0.906 |
| | 1.13 | -2.20 | -0.18 | 0.51 | 0.112 |
| | 1.56 | -0.95 | 0.23 | 0.44 | 0.776 |
| 7 | 2.31 | -0.66 | 0.51 | 0.75 | 1.793 |
| | 2.38 | -1.94 | -0.17 | 0.681 | -1.167 |

The evaluated results are listed in Table 5.4; it contains the maximum, minimum and mean error in each axis and their corresponding standard deviation. Lastly, As the implemented ekf sensor fusion, both the initial noise covariance and process noise covariance are not tuned, which could cause a delay for the coverging time between filtered odometry

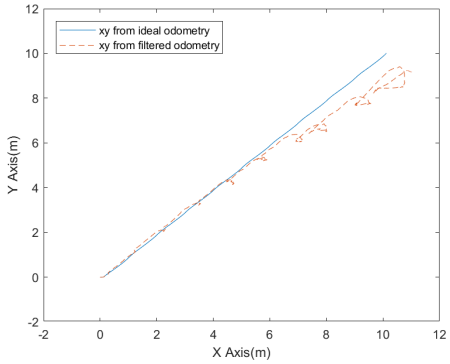
and the real one. Therefore, a close error was introduced to compare the difference between the filtered odometry and ideal dometry at 20 seconds, approximately two seconds after the UAV reached the target. Although, in Table 5.4, one IMU and one odometry do not give the largest close error at 20 seconds. Figure 5.1 and 5.2 shows the 2D trajectory comparison between the ideal and filtered odometry over the whole simulation time, which gives a better evaluation for the estimation to minimise the error caused by delay. By observating Figure 5.1a, 5.1c, 5.2a and 5.2c, it can be seen the ekf_localisation node yields better performance with an increasing number of sensors, as the resulting trajectory from case $No.2,3$ with more sensors are closer to the actually executed trajectory. The combination of two IMU and two odometries gives the best estimation.




(a) 1 IMU 1 Odometry



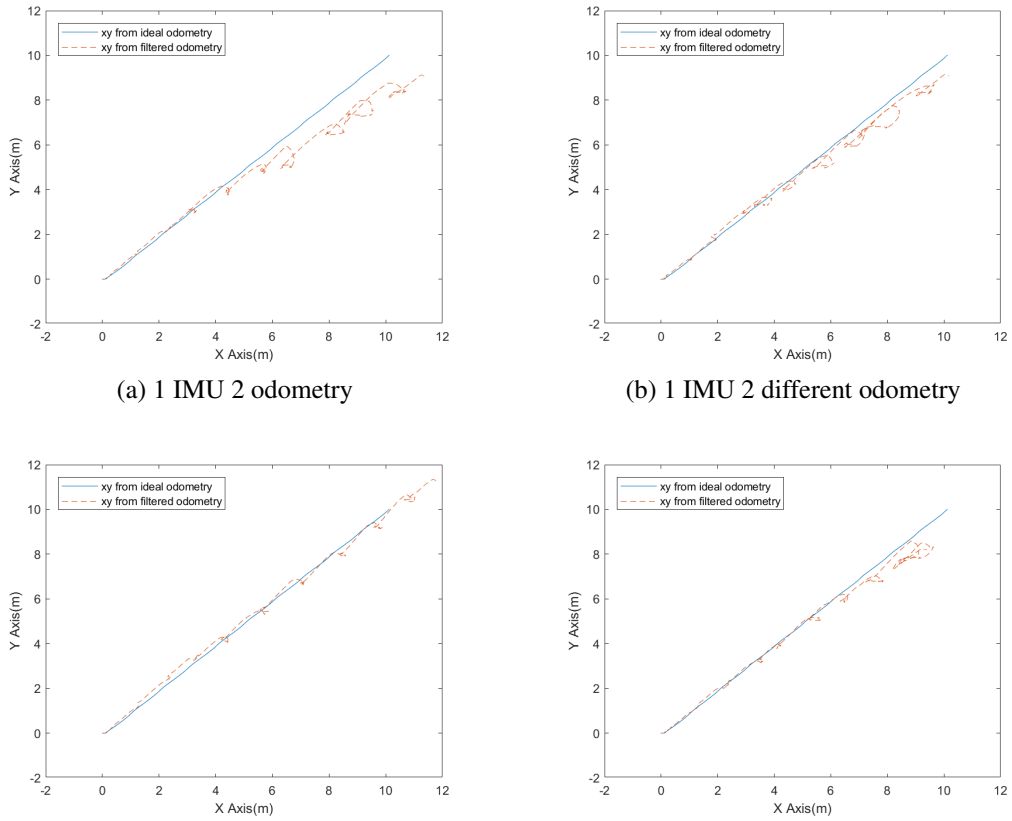

(b) 1 IMU 2 odometry




(c) 1 IMU 2 different odometry

Figure 5.1: Comparison between ideal odometry and filtered odometry in 2D

For the effect of sensor choice between identical and different sensor parameters. In Figure 5.1b, 5.1c and Figure 5.2a, 5.2b, It clearly shows that the combination of different sensor returns better estimation of the executed trajectory. However, in Figure 5.2c, 5.2d,




(a) 1 IMU 2 odometry

(b) 1 IMU 2 different odometry

(c) 2 IMU 2 odometry

(d) 2 different IMU 2 different odometry

Figure 5.2: Comparison between ideal odometry and filtered odometry in 2D

the combination of the different sensor does not yield a better estimation of the executed trajectory, but in Table 5.4, the combination of sensors with different noise parameters gives smaller standard deviation error in both x, z-axis. To investigate, it requires the process of tuning the ekf_localisation node with proper initial noise covariance and process noise covariance, to minimise the odometry error caused by the delay between the estimated position and the actually executed trajectory. This is beyond the scope of this thesis, and further investigation is required to evaluate whether sensors with different noise parameter yield better estimation for EKF fusion.

Table 5.5: Restricted area tested scenarios

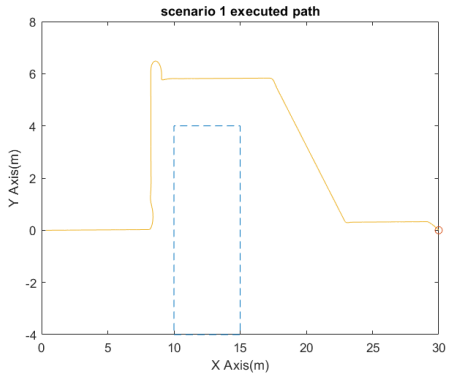
| Case No | Target | | | Restricted Area | | | |
|---|---|---|---|---|---|---|---|
| | $x$ | $y$ | $z$ | $x_{min}$ | $x_{max}$ | $y_{min}$ | $y_{max}$ |
| $No.1$ | 30 | 0 | 2 | 10 | 15 | -4 | 4 |
| $No.2$ | 10 | 3 | 2 | 5 | 15 | -4 | 0 |
| $No.3$ | 20 | 8 | 2 | 10 | 25 | -4 | 4 |
| $No.4$ | 25 | 3 | 2 | 10 | 15 | -4 | 4 |

## 5.2 Restricted area avoidance with ideal odometry

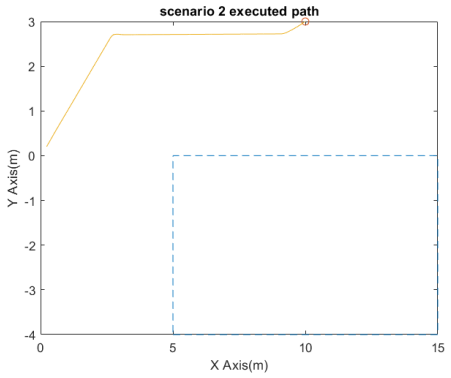
For restricted area avoidance algorithm, The UAV will constantly take input from an ideal odometry sensor without any noise implemented, and compare its current location with the restricted area, then calculate the shortest exit strategy to avoid the area according to the relationship between the potential trajectory and the restricted area. For the implemented navigation system, the navigation system will return an error message if the target location is within the restricted area or lies on any one of the four boundaries. The relationship between the potential trajectory and the restricted is divided into three scenarios:

- the potential trajectory insects with two parallel boundaries from the RA.
- the potential trajectory insects with two adjacent boundaries from the RA.
- the potential trajectory does not intersect with the RA, but the UAV is located on the edge of the RA.

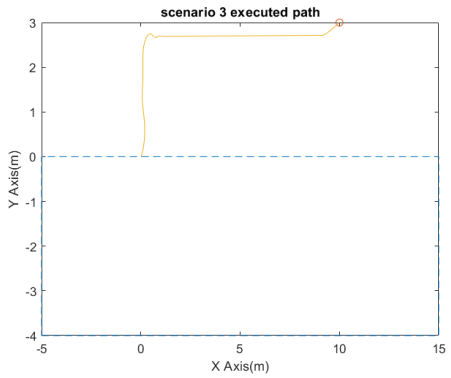
Table 5.5 includes the four scenarios designed to test the UAV's ability to avoid an user-defined restricted area. Each scenario is given with a 3D target point and a restricted rectangle area defined by $x_{min}, x_{max}, y_{min}, y_{max}$, the UAV is expected to avoid the restricted area and reach the target point from the original point with the coordinate of $(0,0,0)$. Case $No.1$ and $No.4$ include the scenario that the potential trajectory intersects with two parallel boundaries of the bounding box, and the difference is the implemented shortest



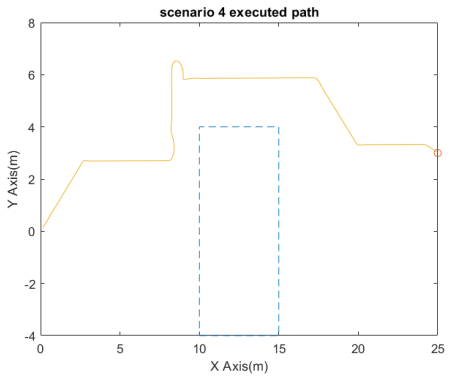

(a) two parallel boundary with equal distance to exit the area




(b) two adjacent boundary




(c) both UAV and target lies outside of the bounding box




(d) two parallel boundaries with a shorter path to exit

Figure 5.3: Restricted Area results in 2D

path exit strategy. In case $No.1$, both the UAV's location and the target is designed to be the same, and it is equal distance for the UAV to avoid the area by taking any of the two corners of the restricted area. In case $No.4$, there is a shorter path to avoid the area, the UAV is expected to calculate the distance from its current location and take the short path to exit the restricted area. Case $No.2$ covers the scenarios that the potential trajectory intersects with two adjacent boundaries, and the UAV is expected to avoid the restricting area by approaching to the corner where those two boundaries intersect. Lastly, case $No.3$ represent the scenario that the UAV is located on the boundary of the restricted area, and the potential trajectory will not intersect with the restricted area, the ideal result would be the UAV to keep a safe distance from the boundary.

The result of restricted area avoidance can be evaluated by recording the UAVs odom-

etry and plot the trajectory has executed. As it is shown in Figure 5.3, the restricted area is represented as the dashed blue line, the orange line represents the executed trajectory in 2D, and the target is marked with the circle. The UAV had successfully reached the target by finding the corrected exit corner when the potential trajectory intersects with two adjacent boundaries (Figure 5.3b), and kept a safe distance from the restricted area when the UAV is located on the edge of the restricted area, and the target location does not pose a potential hazard (Figure 5.3c). For the scenarios when the potential trajectory intersects with two parallel boundaries, the UAV had successfully reached the target by avoiding the restricted area with a safe distance, and the UAV had found the correct corner to exit the restricted by calculating and comparing the distance to each corner (Figure 5.3d).

However, in Figure 5.3a the executed trajectory is slightly different from what is expected as the UAV1 did not go straight from the original point$(0,0)$ to $(8,6)$ and point $(17,6)$ to the target$(30,0)$. Those trajectories are determined purposely to save computer resources in restricted area avoidance and next waypoint calculation stages. For restricted area avoidance algorithm, the UAV is programmed to calculate the next waypoint with respect to the restricted area, and only trigger the potential intrusion allert when it is approaching the resized area, instead of the full navigation process. By implementing this way, it leads to a straight line between $(0,0)$ and $(8,0)$. For the trajectory between $(17,6)$ and $(30,0)$, this is caused by how next waypoint is calculated when there is no obstacle or restricted area. Ideally, to reach a target in an empty world, the shortest trajectory to the target should have the unit vector of:

$$\begin{bmatrix} Dir_x \\ Dir_y \\ Dir_z \end{bmatrix} = \frac{\begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix} - \begin{bmatrix} U_x \\ U_Y \\ U_z \end{bmatrix}}{\sqrt{(D_x-U_x)^2+(D_y-U_y)^2+(D_z-U_z)^2}} \tag{5.1}$$

Where $(D_x, D_y, D_z)$ and $(U_x, U_y, U_z)$ are the coordinate for the target and UAV's current

position respectively, the UAV then should travel along the unit vector until an obstacle is detected. For the implemented path planning algorithm, the next waypoint is calculated by comparing UAV's current position with the target in each axis respectively, the next waypoint$(N_x, N_y, N_z)$ is incremented or decremented from UAV's current position according to the difference in each axis (with details in Section 4.3.2).

Lastly, as it is shown in the Figure 5.3a and 5.3d, the UAV seems to go over the safe distance when the UAV is turning left around 8 in the x-axis, which is possibly caused by system error which is consisted of control and localisation error. Since the UAV's position is provided by an ideal odometry sensor for the implemented restricted area avoidance scenarios. The control system will be evaluated as it happens only when the UAV is involving a manoeuvre. As it is shown in Figure 4.26, the position controller is executed by taking position and yaw command. The UAV is programmed to travel along the boundary which it is adjacent to, by facing in the direction where the trajectory leads to. The trajectory overshot happened around when the UAV is required to change the yaw angle from 90 degrees to 0 degrees. It is reasonable to assume the trajectory overshot is caused by attitude tracking component in the position control system. To test this, change the algorithm to make the UAV travelling with a zero degree yaw angle and plot the trajectory again.

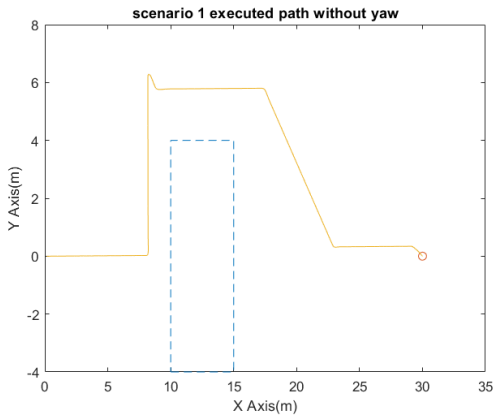



Figure 5.4: Scenarios 1 executed path without yaw

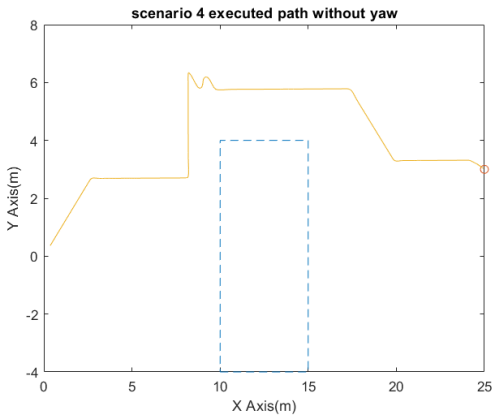



Figure 5.5: Scenarios 4 executed path without yaw

Figure 5.4 and 5.5 shows the 2D overview for the trajectory executed without yaw implementation for case *No*.1 and *No*.4 respectively. The resulting trajectories are still unstable, but the overshooting is much sharper than the executed trajectory with yaw implemented, this is caused by the implemented position controller restricted the initial attitude error less than 90 degrees to obtain the stability of the complete system, but an in-depth explanation would be beyond the scope of this thesis. The reference and overview of the properties of the position controller can be found in [10]. A possible solution to overcome the instability from the position controller is to introduce an intermediate waypoint with smaller turning angle when a sharp manoeuvre is required. However, this restricts the flexibility of the UAV's movement.

## 5.3 Non-axis-aligned restricted area

From the resulting trajectory, it can be seen that the implemented algorithm has successfully found a relatively short path to avoid the restricted area. However, the method is only feasible when the restricted area is rectangle and axis aligned. This section includes the discussion for the proposed implementation when the restricted area is not axis aligned.

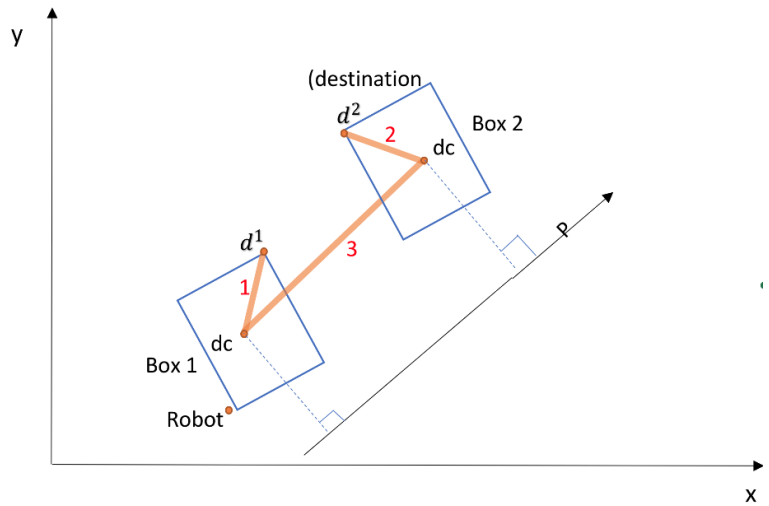



Figure 5.6: Restricted rectangle with rotation

Figure 5.6 shows a rectangle restricted area (box2) which is not axis aligned. Where

the box1 is the rectangle constructed aligned with the same axis with box1, defined by the UAV's position (R) and the target (d1). As it is shown in the Figure, the confliction between potential trajectory and the restricted area can be determined by applying Separating Axis Theorem used in Section 4.3.3. However, now the algorithm needs to find the corresponding distance projected along the axis which the restricted area is aligned to ($\bar{P}$). The methodology can be implemented as follows. Firstly, to identify any point ($d^2$) from the boundary which is close to the target point ($d^1$), and find the central point for box1 ($dc_1$) and box2 ($dc_2$) respectively. Therefore the vector between the centre of each box can be calculated as:

$$\bar{d3} = \begin{bmatrix} dc_x^2 \\ dc_y^2 \end{bmatrix} - \begin{bmatrix} dc_x^1 \\ dc_y^1 \end{bmatrix} \tag{5.2}$$

Apply the same method to get the vector between each box to their edge:

$$\bar{d1} = \begin{bmatrix} d_x^1 \\ d_y^1 \end{bmatrix} - \begin{bmatrix} dc_x^1 \\ dc_y^1 \end{bmatrix} \tag{5.3}$$

$$\bar{d2} = \begin{bmatrix} d_x^2 \\ d_y^2 \end{bmatrix} - \begin{bmatrix} dc_x^2 \\ dc_y^2 \end{bmatrix} \tag{5.4}$$

With vector ($\bar{d1}, \bar{d2}, \bar{d3}$ ) and the axis vector $\bar{P}$, it is possible to get the the distance projection along the axis $\bar{P}$ by applying dot product:

$$\begin{aligned} \bar{dp1} &= \bar{d1} \cdot \bar{P} \\ \bar{dp2} &= \bar{d2} \cdot \bar{P} \\ \bar{dp3} &= \bar{d3} \cdot \bar{P} \end{aligned} \tag{5.5}$$

Hence the gap between box1 and box2 can be calculated by $|\bar{dp3}| - |\bar{dp2}| - |\bar{dp1}|$, and the trajectory is determined to be in a potential collision in $\bar{P}$ axis with the restricted area if the result value is smaller or equal than zero. Any collision along the other axis

can be checked by applying the same method by rotating $\bar{P}$ 90 degrees. The trajectory is concluded to be no collision with the restricted area if there is no overlap on any one of the axes. For even more complicated restricted area shapes, it is possible to segment the restricted area into multiple polygons and apply the same method by constructing a new box for each side of the polygon. The trajectory can be safely concluded to be collision-free if any of all those polygons are determined to be free of any potential collision.

## 5.4 Conclusions

Spatial awareness is the ability to be aware of oneself in space, and it is an organised knowledge of objects concerning oneself regarding space and distance. This chapter includes the evaluation and discussion of the performance of estimating UAV's state with EKF fusion technique. Furthermore, the implementation of restricted area avoidance is also included in this chapter, as the UAV is required to avoid the pre-defined area based on its dynamic position within the environment rather than the detected obstacles.

For the implementation of UAV state estimation with EKF, due to the limitation of the implemented control system, which supports four DoF tracking of the UAV with three positions and one heading direction. UAV's orientation estimation is not evaluated; the experiment is focused on the performance of the position estimation by fusing a various combination of sensors. Two types of sensor are used for the UAV state estimation, IMU sensor and odometry sensor, where the IMU provides input state of angular velocity and linear acceleration, odometry provides linear velocity. The UAV is programmed to execute a user-defined trajectory, and the estimated EKF filtered position data is compared with the position data from ideal odometry without any noise implemented. From the experiment result, it is confident that the ekf_localisation node yields better performance by increasing the number of sensors used. Since EKF works based on trusting more on the input sensor with less noise, another set of experiment is designed to compare the performance between fusing multiple sensors with identical and different noise parame-

ter. Partial results supports fusing different sensor yields better performance than sensors with identical noise parameter; further investigation is required. Improvement can be achieved by determining the correct initial noise covariance and process noise covariance of the sensor system, to reduce the converging time between the estimated data and the actually executed trajectory.

For restricted area avoidance case study, an ideal odometry sensor is used to provide the UAV with its real state. From the result, the UAV is proven to have spatial awareness ability as it keeps a good understanding of its position during the whole navigation process. The implemented algorithm is capable of searching for a safe trajectory by avoiding the axis-aligned restricted area with a relatively short path with low computing requirement. Methodology for an irregular shaped restricted area is discussed if the restricted area is in any other complicated shape. The resulting trajectory is constrained with the limitation the implemented Lee position controller, which requires the initial attitude error less than 90 degrees to obtain the stability of the complete system. Possible solution is to introduce an intermediate waypoint with smaller change of the UAV's state. Furthermore, the restricted area is resized by increasing a fixed error distance, where the actual distance should be identified from the system errors, which consists of the accumulation errors from both controls, localisation systems.

# Chapter 6

# Evaluation of obstacle avoidance

## 6.1 Static obstacle avoidance case study

This section includes the experiments designed to test the performance of the UAV's static obstacle avoidance ability. The UAV is expected to reach a randomly assigned target within a 3D unknown environment. There are four aspects which can affect the UAV's static obstacle avoidance ability: mapping (Chapter 4.1), localisation (Chapter 4.2), obstacle avoidance algorithm in cognition system (Chapter 4.3.2), and control system (Chapter 4.4). For the implemented localisation system, an ideal odometry sensor is used to provide the true position of the UAV. From the result of spatial awareness case study in Chapter 5, the control system is tested to be able to control the UAV at the desired position with an error of $(0.13, 0, -0.2)$ meter in each axis respectively. This chapter will focus on how the mapping system and collision avoidance algorithm affect the performance of UAV's static obstacle avoidance ability.

The mapping system works based on utilising a depth camera to convert the simulated environment into point cloud data. Table 4.1 lists the depth camera's relevant parameter which can affect the mapping performance, such as frame rate, maximum range and horizontal FoV. Where the maximum range and horizontal FoV defines how much the depth camera can detect from the environment, and the frame rate defines the working frequency

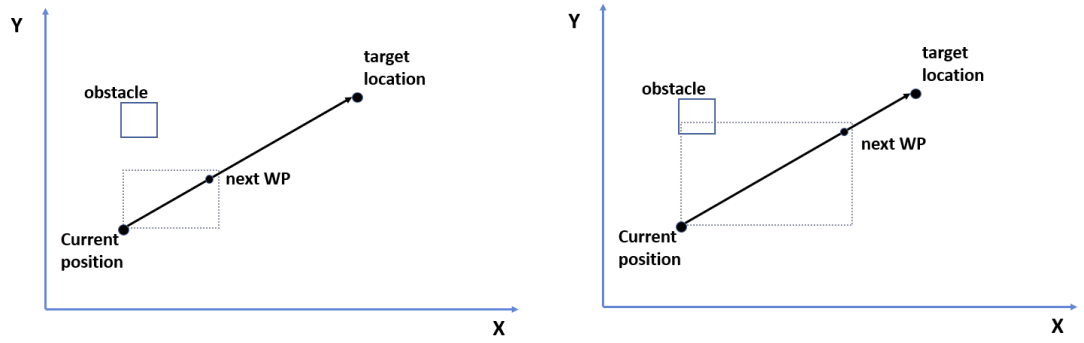
of the depth camera. With the processed point cloud representation of the working environment, the mapping system then converts the point cloud data into octomap representation, where the 3D environment is segmented into multiple spaces, which can be further segmented recursively into eight sub-space until the map reaches the user desired resolution. Higher-resolution(low parameter value) map gives a better representation of the complicated part of the working environment but requires more computational capacity.

The implementation of the algorithm for path planning with static obstacle avoidance can be roughly divided into the following three steps:

- calculate the next intermediate waypoint to the target location
- construct a bounding box based on the UAV's current location and the proposed next intermediate waypoint and evaluate the potential hazard for the 3D area
- search for an alternative waypoint if the initially proposed waypoint is in potential hazard

For the first step, the waypoint is calculated base on the geometry distance from the UAV's current location to the target location. In order for the waypoint to be evaluated for collision checking, the maximum distance from UAV to the next intermediate waypoint is constrained by the depth camera's maximum range minus half length of the bounding box $bbx_l$ described in Equation 4.18. The potential hazard for the next waypoint is evaluated by constructing an axis-aligned bounding box area, any objects detected within the area will trigger the algorithm to the next step, search for an alternative collision-free waypoint to the target. Because the UAV is working in an outdoor environment, it is more likely for the UAV to avoid the obstacle by going over it, the default implemented strategy of searching for the alternative waypoint is to search for collision-free space above the UAV's current location, then dividethe UAV's near space into five different bounding box and order them in a special sequence based on the bounding boxes' distance to the target location. Lastly, the algorithm will evaluate potential hazard those bounding box in the defined the order.

**Next waypint calculation**

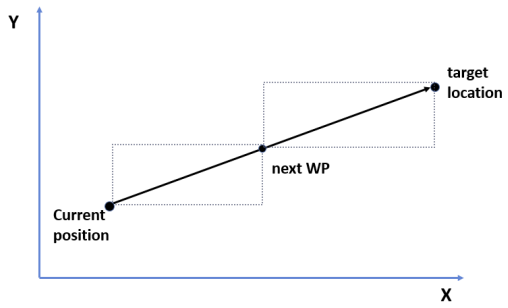



(a) shorter next waypoint distance results in a smaller are that will be evaluated for collision checking

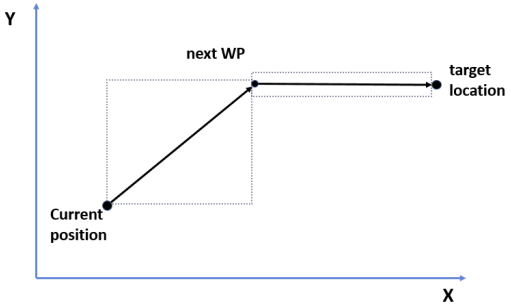
(b) longer next waypoint distance results in a larger are that will be evaluated for collision checking

Figure 6.1: How next waypoint location and size of the bounding box can affect the algorithm

The calculation for the coordinate of next intermediate waypoint impacts the construction of the bounding box used for collision checking. In an open space with fewer obstacles, longer next waypoint distance results in faster converging to the target location, However, in a relatively crowded working environment, as it is shown in Figure 6.1, longer next waypoint distance also results in a larger axis-aligned bounding box area that need to be evaluated by the collision checking system, which will effectively result in a larger unnecessary bounding box area and trigger the algorithm search for alternative waypoint. The resulting large bounding box requires a higher amount of computational capability when the potential trajectory between UAV and the next waypoint is not axis-aligned. A possible solution to improve this is to manipulate the UAV's trajectory to enable the UAV reaching the target location in each axis separately. Therefore, the resulting bounding box's length in the arrived axis is only incremented with the necessary length for the UAV to pass (Figure 6.2). Experiment on the choice of next waypoint step size is proposed to evaluate the processing time required for bounding box iteration and UAV's journey duration.

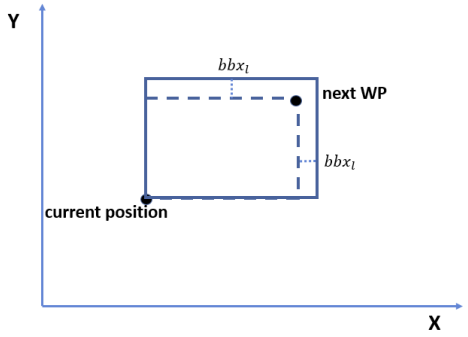



(a) going straight from UAV's current location results in bigger bounding box requires for collision evaluation, require more processing time for bounding box itteration

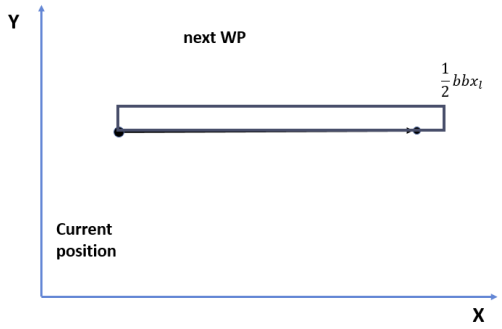



(b) UAV reach an axis first, requires less processing time but longer journey duration

Figure 6.2: How different trajectory affects the bounding box selection with the same target location




(a) when the UAV moves on both x, y axis

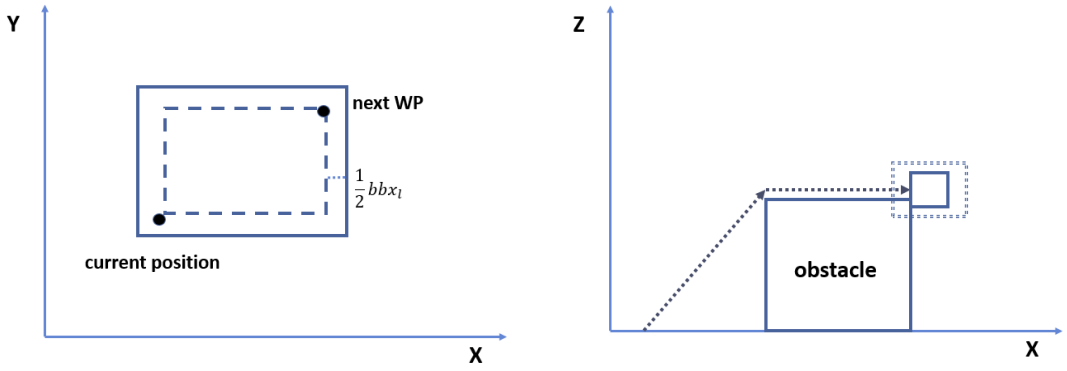



(b) when the UAV moves only on x axis

Figure 6.3: 2D overview of constructing a bounding box for collision checking, based on the UAV's current position and the proposed next waypoint from Equation 4.19

### Bounding box construction

As it is shown in Figure 6.3, the size of the bounding box is constrained by the geometrical relationship between the position of the UAV and the proposed next waypoint, with extended length $bbx_{additional}$ from the UAV towards the next waypoint in each axis. The additional length needs to be greater than $\frac{1}{2}bbx_l$, which works fine under the assumption that UAV is moving on more than one axis (Figure 6.3a), the defined bounding box area covers the minimum space required for any potential rotation of the UAV. However, as it is shown in 6.3b, when the UAV is moving only on x-axis, the resulting bounding box does not contain sufficient mapping information for potential hazard evaluation.



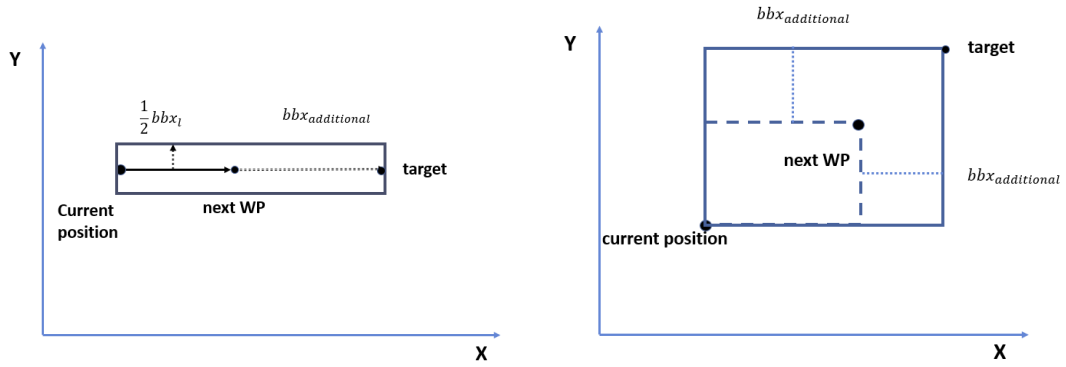

(a) when the UAV moves on both x, y axis

(b) when the UAV moves only on x axis

Figure 6.4: 2D overview of constructing bounding box by extending a fixed length of the UAV's current position and next waypoint

One possible solution is to extend the bounding box area with a length of $\frac{1}{2}bbx_l$ at the minimum and maximum coordinate in each axis (Figure 6.4a). However, in the experiment, the resulting bounding box contains unnecessary obstacle information when is UAV is located precisely by the edge of the obstacle, as it is shown in Figure 6.4b, the UAV is travelling from the left toward the target at right, it avoids the obstacle by going over it vertically. When the UAV has just avoided the obstacle at the position of top right of the obstacle, the consequential bounding box triggers the potential hazard even it is safe for the UAV travel towards the target, This is because the minimum additional length $bbx_l$ contains system errors from the navigation system, which is greater than the geometric length of the UAV. Furthermore, the potential hazard within in the bounding box area is evaluated by leaf_bbx_iterator function provided by octomap, with a given minimum and maximum coordinates, due to rounding and discretisation effects, nodes may be traversed that have float coordinates appearing outside of the float bounding box. Although the algorithm can still find an alternative waypoint after the false collision alert, the algorithm can be improved by constructing a bounding box according to the converging status of the UAV and its target in each axis separately.

Figure 6.5 shows the 2D overview of constructed bounding box based on the converging status in x and y-axis separately. The algorithm will first check if the UAV has arrived at the target location by separating its axes and apply different techniques to construct the




(a) when the UAV moves on both x, y axis

(b) when the UAV moves only on x axis

Figure 6.5: 2D overview of constructing bounding box based on the converging status in x, y axis sperately

bounding box. The algorithm considers the UAV has arrived if the difference between the UAV's position and the target location is less than the proposed step size from the next waypoint calculation. For example, the UAV is considered to have arrived on its x-axis with its current position (5.4, 6.6, 7.6) and target location (5, 6, 7) if the next waypoint is calculated to move by 0.5 meters by each step. In Figure 6.5a, the UAV has arrived at the target location by its y axis, the corresponding bounding box is only extended with a fixed length towards the target in its x axis, with a minimum length of $\frac{1}{2}bbx_l$ extended in both positive and negative y In figure 6.5b, The UAV has not arrived in either x or y axis, the bounding box is extended towards the next waypoint in both x and y axis with the length of $bbx_{additional}$. The construction of the bounding box in z-axis is implemented in the same approach, except that the minimum value is constrained to be 1 meter, to compensate the ground land being detected as obstacle. Ideally, the constrained length should be half of the octomap resolution. Furthermore, as the default alternative waypoint searching strategy is to "look up", it is more preferable for the UAV to go up at an earlier stage than later when the UAV is working in a crowded environment, which will cause the algorithm triggers the collision alert and cost more computational power to search for alternative waypoint. The choice of $bbx_{additional}$ with larger values than the minimum length of $bbx_l$ will result in a larger area that is evaluated for collision checking and bring the UAV to go up at the early stage of the simulation.

### 6.1.1 Experiment design

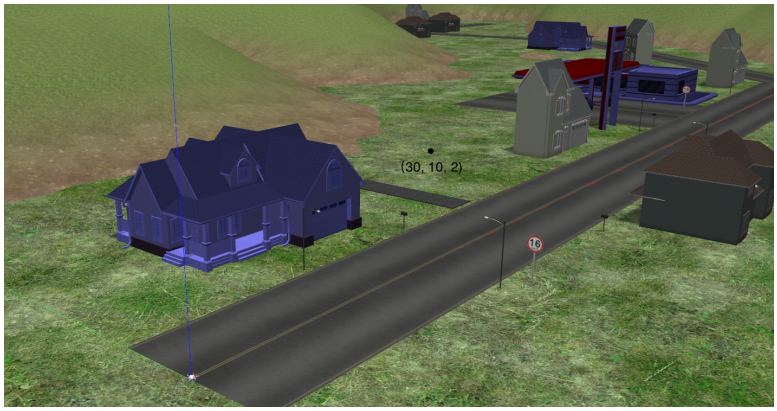



Figure 6.6: 3D map with static obstacles and user-defined target

This section contains the evaluation of the feasibility of static obstacle avoidance ability of the algorithm, and how the previously discussed factors can affect the performance of the algorithm. Figure 6.6 illustrated a 3D outdoor environment, the UAV is expected to find a collision-free trajectory to the user-defined target $(30, 10, 2)$. The experiments are designed as follows, with the according case studies given in Table 6.1.

- evaluation of how octomap resolution can affect the performance of the algorithm
- evaluation of how next waypoint distance calculation can affect the algorithm
- evaluation of how bounding box construction can affect the algorithm

Table 6.1: Static obstacle avoidance tested scenarios

| **Case No** | **octomap resolution (m)** | **waypoint step x, y, z (m)** | $bbx_{additional}$ **x, y, z (m)** |
|---|---|---|---|
| 1 | *various* | $0.5, 0.5, 0.5$ | 4, 4, 4 |
| 2 | 0.5 | *various* | 4, 4, 4 |
| 3 | 0.5 | $0.5, 0.5, 0.5$ | various |

### 6.1.2 Result evaluation

#### Octomap resolution

To evaluate how octomap resolution can affect the algorithm's performance, The frame rate of the depth camera is set to be 30 Hz, the next intermediate waypoint is calculated with the step size of 0.5 meters in both x, y, z-axis, with resulting distance from UAV's current position to the next waypoint in the range of $[0.5m - \sqrt{0.75}m]$, the bounding box for collision checking is constructed by adding an additional length $bbx_{additional}$ of 4 meters in both x, y, z-axis. For example, if the UAV is travelling from (4, 4, 4) to (4.5, 4.5, 4.5), the resulting bounding box area would be a cubical sapce with minimum value and maximum value of (2, 2, 2) and (8.5, 8.5, 8.5). The performance of the algorithm is evaluated by recording the total flight duration consumed for the UAV to reach the target location with various octomap resolution, and the results are given in Table 6.2. The results show that the algorithm's performance improved significantly when the map resolution changed from 0.05 meter to 0.1 meters, at the resolution of 0.05 meter with a cubical bounding box with a length of 4 meters, there are $80^3 = 512000$ iterations required for collision checking, by changing the resolution to 0.1 meter, the iteration required is $40^3 = 64000$. This is probably due to the system halt caused by the high amount of iteration since the performance only improves slightly by keep increasing the map resolution until it reaches the resolution of 0.5 meters.

At the resolution of three and five meters, the algorithm failed to find a trajectory to the target location. More precisely, the algorithm failed to find an initial waypoint at the beginning of the simulation. This is because the perception unit in mapping system detects the ground land and treat it as an obstacle, even the UAV is in an open space, the resulting octomap with low resolution (large resolution number) from the ground covers the UAV, hence no initially waypoint found at the beginning the simulation. Figure 6.7 shows the resulting bounding box with the resolution of five meters. An ideal solution would be the improvement in the mapping system, which segments the ground land from

the rest of the static obstacles.

Table 6.2: Algorithm performance with different octomap resolution for outdoor environment

| Case No | octomap resolution (m) | flight duration (MM:SS) |
|---|---|---|
| 1.1 | 0.05 | 4:57 |
| 1.2 | 0.1 | 1:05 |
| 1.3 | 0.15 | 0:53 |
| 1.4 | 0.2 | 0:51 |
| 1.5 | 0.5 | 0:48 |
| 1.6 | 1.0 | 0:48 |
| 1.7 | 1.5 | 0:48 |
| 1.8 | 2 | 0:49 |
| 1.9 | 3 | failed |
| 1.10 | 5 | failed |

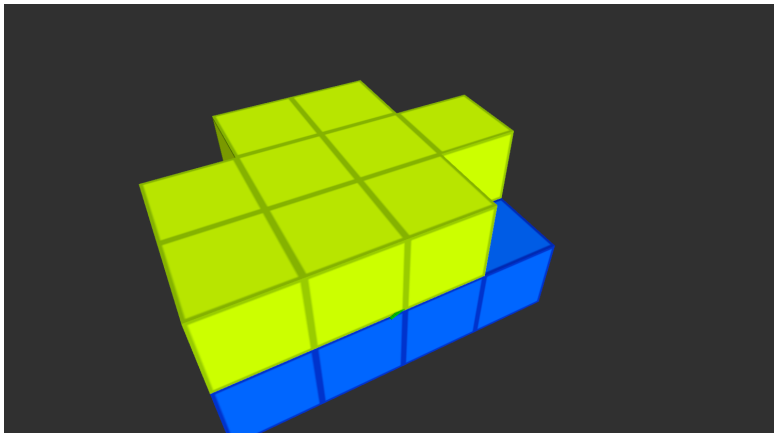


Figure 6.7: The constructed bounding box when the UAV at its initial position with the resolution of five meters

**Next waypoint step size and** $bbx_{additional}$

The sum of next waypoint step size, and additional length $bbx_{additional}$ is constrained by the maximum sensor range from the depth camera, consider the step size and additional

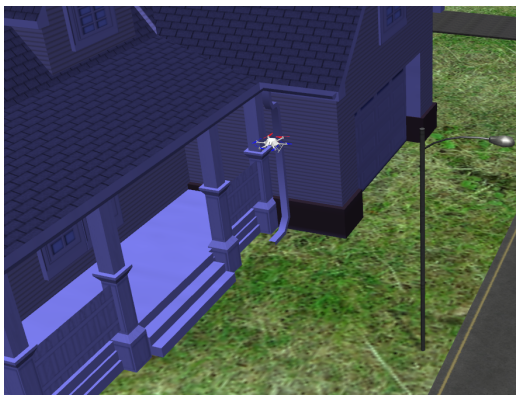
length are same in x, y, z-axis. The maximum sum $sum_{max}$ can be calculated by:

$$\begin{aligned} \sqrt{3{sum_{max}}^2} &= 10 \\ sum_{max} &= \sqrt{\frac{100}{3}} \\ &\approx 5.77 \end{aligned} \tag{6.1}$$

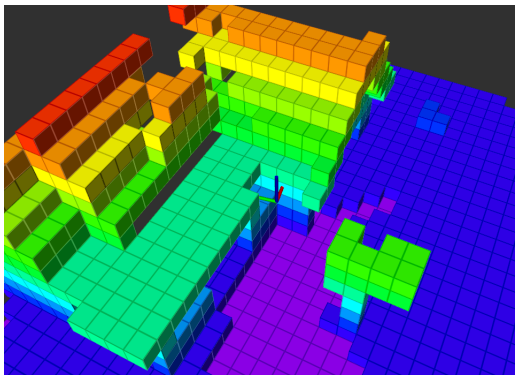
The choice of next waypoint step size and additional length $bbx_{additional}$ are highly interrelated and will be evaluated together with octomap resolution of 0.5 meter, Table 6.3 shows the experiments designed and the corresponding result, The results show that the choice of the additional length $bbx_{additional}$ has minimum effects on the algorithm's performance, the step size of the next waypoint calculation affects significantly on the converging time of the algorithm, However, by observing the 3D simulation, it is noted that the UAV tends to oscillate more with the increasing the step size, this is probably caused by the implemented control system, when the distance traversed approaching the threshold of the maximum available range, further investigation is required to identify the cause of the oscillation. The algorithm failed to find a trajectory when the step size is 0.1 meter, this is due to the step size is way smaller than the additional length $bbx_{additional}$ and octomap resolution, possible improvement is to reduce the additional length and construct a smaller bounding box then look for alternative waypoint, Figure 6.8 shows where the UAV has stuck in gazebo simulated environment and the corresponding octomap representation.

Table 6.3: Algorithm performance with different next waypoint step size and additional bounding box length in outdoor environment

| **Case No** | **waypoint step x, y, z (m)** | $bbx_{additional}$ **x, y, z (m)** | flight duration (MM:SS) |
|---|---|---|---|
| 2.1 | 0.1, 0.1, 0.1 | 4, 4, 4 | failed |
| 2.2 | 0.5, 0.5, 0.5 | 4, 4, 4 | 0:47 |
| 2.3 | 1, 1, 1 | 4, 4, 4 | 0:29 |
| 2.4 | 1.5, 1.5, 1.5 | 4, 4, 4 | 0:23 |
| 3.1 | 0.5, 0.5, 0.5 | 3, 3, 3 | 0:47 |
| 3.2 | 0.5, 0.5, 0.5 | 3.5, 3.5, 3.5 | 0:47 |
| 3.3 | 0.5, 0.5, 0.5 | 4.5, 4.5, 4.5 | 0:47 |



(a) simulated environment for case *No*.2.1, where the algorithm failed search for a waypoint



(b) octomap for case *No*.2.1,where the algorithm failed search for a waypoint

Figure 6.8: where the algorithm failed to find a solution for case *No*.2.1

### Executed trajectory

The experiment result can be evaluated by plotting 3D scatter for the executed waypoints, and compare the value according to the size and location of the obstacles defined in the world file. Moreover, the executed waypoint also contains information about the UAV's orientation, which is difficult to represent the full information within a single plot. Rviz (Ros Visualisation) is employed to display sensor data and state information from ROS. It is possible to display a live representation of the UAV's odometry (position and orientation) and the mapping information within a single window. Figure 6.9 illustrates the

3D visualisation taken from RViz. Where the UAV's odometry is represented by the red arrow pointing to the direction which the UAV is facing. The rest is the visualisation for the obstacles detected by the mapping system. As it is shown in the picture, the UAV has successfully reached the target and avoided the obstacle by going over the building. However, as described in Section 4.3.2, the algorithm's default solution to avoid any obstacle is to set a trajectory by going over the obstacle. Hence, it is essential to test the algorithm when the UAV is in a roofed environment.

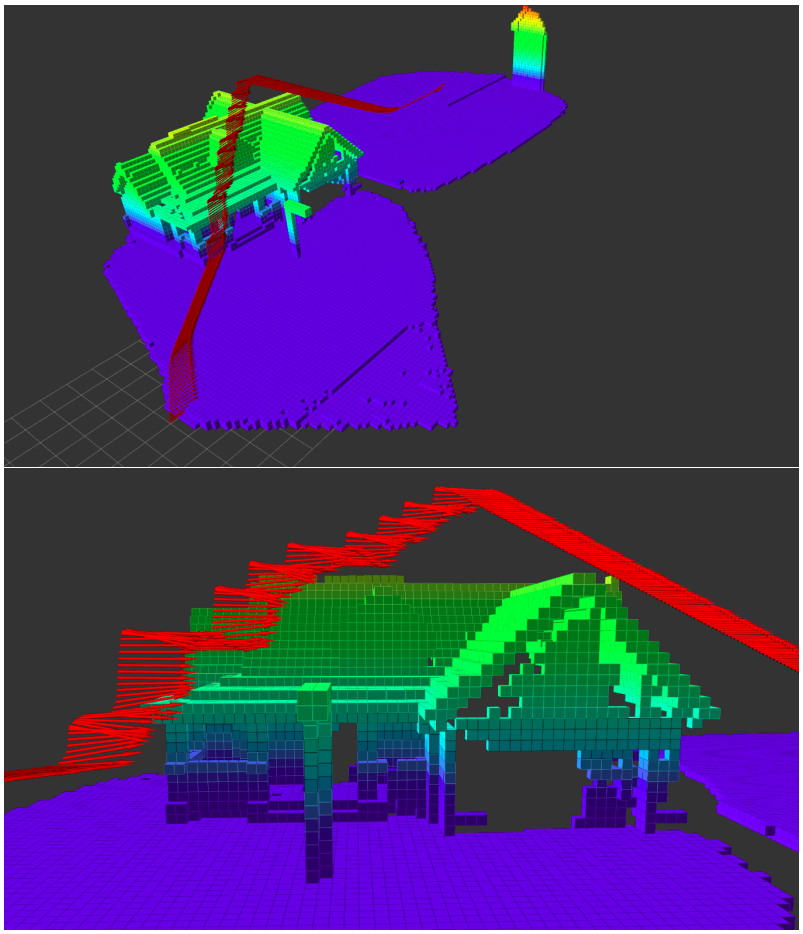


Figure 6.9: The executed trajectory viewed from different angle.

Figure 6.10 shows the resulting trajectory has been exected within a roofed environment. The overall trajectory is concluded to be collision-free as the UAV has successfully detected and avoided the surrounding obstacles during the navigation. However, there are still unexpected aspects of the experiment result. As it is shown in Figure 6.10, the trajec-

tory went up and down while the UAV is trying to avoid the roofed area. This is caused by how the alternative waypoint is determined, as the UAV will only check collision with a small bounding box instead of the whole available map data.

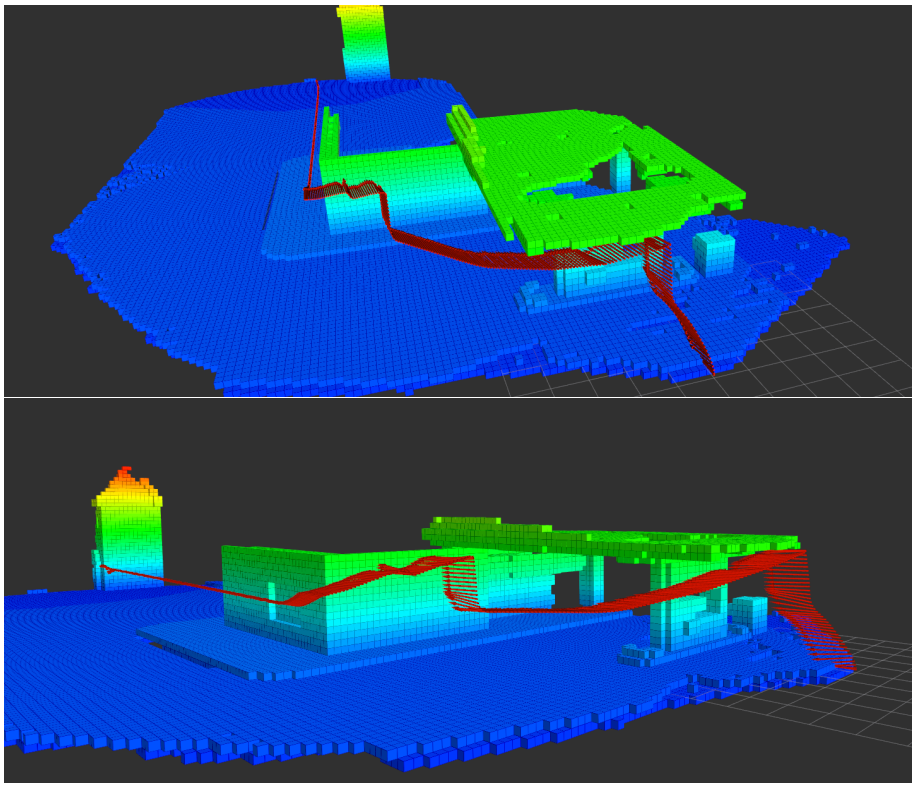


Figure 6.10: The executed trajectory viewed from different angle in a roofed envrionment.

### 6.1.3 Conclusions

This chapter contains the discussion and evaluation for the performance of the static obstacle avoidance algorithm. The algorithm starts by calculating an intermediate next waypoint based on the UAV's current position and the user-defined target location, then evaluate the potential hazard for the proposed waypoint by constructing an axis-aligned bounding, any obstacles detected within that area will trigger the algorithm search for an alternative collision-free waypoints. Different techniques of constructing the bounding box are discussed and evaluated, to contains minimum but sufficient areas for collision checking. The next waypoint is calculated by separating axis, which enables the UAV

to arrive on one axis separately, it minimises the size of the resulting bounding box, and reduces the iteration required for collision checking.

From the RViz 3D visualisation and recorded experiment data, the algorithm is tested to be able to find a collision-free trajectory in a static environment, with the corrected parameters for octomap resolution, next intermediate waypoint calculation and additional length to construct the bounding box used for evaluating potential hazard. Different scenarios were designed to investigate how these three parameters affect the performance of the algorithm.

From the experiment result, the next waypoint calculation impacts most significantly on the performance of the algorithm, less time is required for longer next waypoint step size. However, the longer next waypoint calculation also causes the UAV to oscillate with unstable trajectories. This is probably caused by the implemented position controller, with longer distance reaching to the maximum threshold of the supported movement range. Further study is required to investigate the cause of oscillation of the trajectory.

The octomap resolution affects the algorithm heavily at the range between 0.05 - 0.15 meters; the navigation simulation reduces significantly by increasing the map resolution. However, the experiment results do not differ between the range of 0.15 - 2 meters. This is probably caused by the iteration required for the potential hazard checking. Further investigation is required to investigate if longer processing time is caused by system halt from the high map resolution (with smaller value). With octomap resolution greater than two meters, the algorithm failed to find any waypoint, and this is due to the implemented depth camera detecting the land and treat it as obstacle, which covers the space of the surrounding environment of the UAV.

Lastly, the experiment results show that the additional length extended by the bounding box has minimal effects on the converging time, a possible explanation is that the additional length is constrained by the maximum sensor range of the depth camera of 10 meters. Further investigation on how additional length affect the performance of the algorithm can be evaluated with a longer range depth camera.

## 6.2 Dynamic obstacle avoidance case study

This section includes the experiments designed to test the performance of the algorithm's dynamic obstacle avoidance ability. The UAV is expected to operate within a 3D unknown environment contains a single moving obstacle, and the algorithm should find a randomly assigned target location by avoiding the dynamic obstacle without prior knowledge of the movement trajectory. In order to do this, the UAV needs to constantly be aware of the position of the moving obstacle and hence chose an appropriate path by predicting the obstacle's future location based on its historical movement trajectory.

In a real-world, dynamic obstacle avoidance involves the process of objects segmentation and classifying static and dynamic objects based on their change of detected obstacles' position, which is beyond the scope of this thesis. The implemented experiments are desired to test the algorithm's feasibility for avoiding a single dynamic obstacle. The dynamic obstacle is simulated as a second UAV by executing a pre-defined trajectory, and feeding its odometry to the navigation system, where the navigation system will determine an appropriate trajectory directly from the second UAV's odometry data, instead of from a comprehensive mapping system with objects segmentation. The implementation of the algorithm can be roughly divided into three steps:

- obstacle future prediction
- potential collision checking
- alternative waypoint searching

The algorithm will first assign an initial waypoint base on only the UAV's current position and target location, then evaluate potential collision by predicting the obstacle's historical trajectory, lastly either chose to execute the initially proposed waypoint or search for an alternative waypoint based on the feedback from the dynamic collision checking system. The potential collision with the moving obstacle is evaluated with Separating Axis theorem, which has minimal effects on the performance of the algorithm. The re-

maining of the section will explain how the obstacle's position prediction and alternative waypoint searching are affecting the performance of the algorithm.

### 6.2.1 Obstacle position prediction

To predict the obstacle's future position, the algorithm firstly need to estimate the obstacle's velocity in each axis from its historical displacement in the past duration $t$, then predict the obstacle's future position range by constructing a cubic bounding box, with the estimated obstacle's velocity and the time $t^u$ required for the UAV to reach the next waypoint with an additional length in each axis to allow the UAV's rotation and obstacle's change of velocity. These two steps are highly interrelated due to the limitation of the implemented position controller for the navigation system, which has no direct control of the UAV's velocity and thus the time $t_u$. Lower $t$ value yields a better representation of the obstacle's instantaneous velocity, which is preferable when the obstacle is accelerating, where the resulting bounding box is larger and gives a better prediction of the obstacle's position range. However, when the obstacle is decelerating, the resulting bounding box will contain unnecessary space for the collision checking, therefore slower converging to the target location.

Table 6.4 shows the results for the experiment designed to evaluate how the duration of $t$ affects the accuracy of obstacle velocity estimation. The moving obstacle is simulated with a second UAV, and the obstacle is travelling only on the y-axis, with the coordinates of $[0, 1, 3, 6, 10, 15, 25]$. They are chosen due to the limitation of no direct control of the navigation velocity, and each distance is double compared to the last one. To allow broader range velocity estimation. Additionally, after the obstacle arrives at each waypoint, it will stay at the corresponding position for 1 second, where the obstacle needs to decrease its velocity to hover at each position. The obstacle's velocity is estimated by broadcasting obstacle's transform (position) into a tracking frame, then utilising the lookupTwist function within ROS tf listener, to estimate the obstacle's velocity by comparing the tracking frame with respect to the observation world frame with various

duration $t$ to average. The estimated velocity is compared with the velocity measured from an ideal odometry sensor. The accuracy of the estimated velocity is evaluated, the difference calculated by subtracting the estimated velocity by the ideal velocity, with the corresponding maximum, minimum, and standard deviation for each duration $t$ used.

From the early paragraph, it states the hypothesis that when the obstacle is accelerating, shorter duration of $t$ yields a better estimation of the obstacle's velocity. To evaluate this, as it is shown in the table, the maximum measured velocity from ideal odometry sensor is approximately $11m/s$, by observing the maximum estimated velocity for different $t$, it can be seen the maximum estimated velocity starts to drop from its true value when the duration is greater than $0.521s$ as it is expected. However, the estimated maximum velocity is extremely larger than its true value when the duration is less than $0.035s$, with the maximum estimate velocity of $384.01m/s$ when the duration is $0.001s$. A possible explanation would be the duration $t$ exceeds the minimum limit of the time $T_p$ required to process the velocity estimation. Which means the computer requires a longer time to update the obstacle's position, and the corresponding displacement during the time $T_p$ is actually larger than what is expected. Therefore the estimated velocity would be:

$$V_e = \frac{v \times T_p}{t} \quad when \quad t < T_p \tag{6.2}$$

$$V_e = \frac{v \times t}{t} \quad when \quad t \geq T_p \tag{6.3}$$

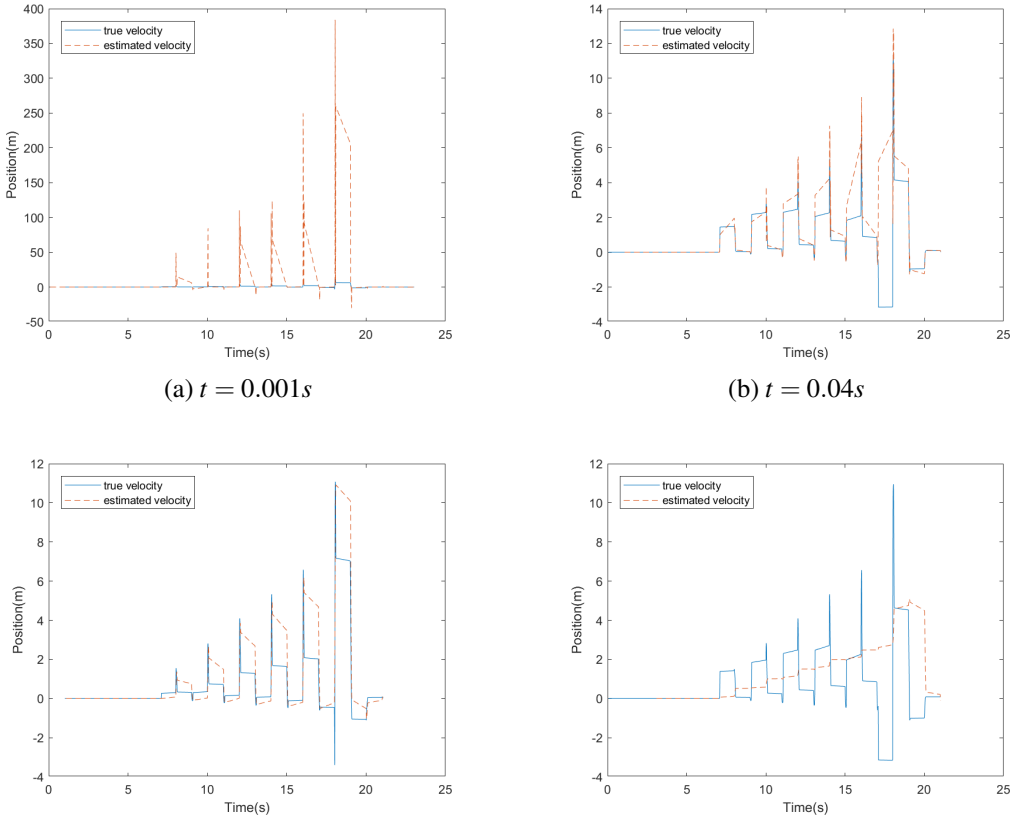
Where $v$ denotes the measured velocity from the ideal odometry sensor, it is possible to estimate the $T_p$ by substituting the maximum velocity and maximum estimated velocity with the corresponding duration $t$ in Equation 6.2. The results are given in $T_p$ column in Table 6.4, the estimated results are approximately within the range of$[0.35 \rightarrow 0.5]s$ until the duration $t$ is larger than $0.04s$, which satisfies the hypothesis in Equation 6.2, 6.2. Therefore, to yield a better estimation of the obstacle's velocity, the duration $t$ used to average obstacle's velocity should be within the range of $[0.04 \rightarrow 0.512s]$. The algorithm

will choose the duration of 0.04$s$ to estimate the obstacle's velocity.

Table 6.4: Velocity difference between ideal odometry sensor and estimated velocity

| $t$ (s) | **difference max (m/s)** | **difference min (m/s)** | **difference SD** | **estimated max (m/s)** | **estimated min (m/s)** | **max (m/s)** | **min (m/s)** | $T_p$ (s) |
|---|---|---|---|---|---|---|---|---|
| 0.001 | 29.26 | -373.29 | 44.67 | 384.01 | -30.16 | 11.08 | -3.23 | 0.035 |
| 0.002 | 8.53 | -174.76 | 20.87 | 185.45 | -9.11 | 11.18 | -3.43 | 0.033 |
| 0.004 | 10.78 | -99.43 | 11.74 | 110.34 | -11.90 | 11.05 | -3.29 | 0.040 |
| 0.008 | 8.76 | -38.25 | 4.88 | 44.95 | -3.11 | 11.17 | -3.11 | 0.032 |
| 0.016 | 8.94 | -28.22 | 4.41 | 34.39 | -2.10 | 11.16 | -3.18 | 0.049 |
| 0.032 | 7.97 | -6.61 | 1.24 | 17.45 | -1.49 | 10.89 | -3.19 | 0.051 |
| 0.033 | 6.47 | -8.84 | 1.41 | 18.32 | -1.57 | 11.21 | -3.19 | 0.054 |
| 0.035 | 8.53 | -12.00 | 1.68 | 14.14 | -1.63 | 11.15 | -3.37 | 0.044 |
| 0.037 | 8.44 | -10.20 | 1.44 | 13.78 | -1.32 | 11.26 | -3.15 | 0.037 |
| 0.04 | 8.87 | -9.88 | 1.25 | 12.85 | -1.27 | 11.33 | -3.16 | 0.045 |
| 0.064 | 1.00 | -9.64 | 1.30 | 15.45 | -1.60 | 11.34 | -3.03 | 0.087 |
| 0.128 | 2.12 | -7.84 | 0.88 | 11.80 | -1.25 | 10.91 | -3.26 | 0.138 |
| 0.256 | 2.54 | -8.75 | 0.95 | 12.62 | -1.29 | 11.15 | -3.39 | 0.290 |
| 0.384 | 4.73 | -5.88 | 1.26 | 11.65 | -1.16 | 11.27 | -3.10 | 0.387 |
| 0.512 | 5.56 | -5.13 | 1.37 | 10.92 | -1.04 | 11.08 | -3.40 | 0.495 |
| 1.024 | 8.43 | -6.48 | 1.94 | 8.99 | -0.75 | 11.19 | -3.24 | 0.822 |
| 1.536 | 8.96 | -7.07 | 2.15 | 6.88 | -0.50 | 11.26 | -3.12 | 0.938 |
| 2.048 | 7.89 | -6.16 | 2.07 | 5.09 | -0.11 | 10.95 | -3.15 | 0.952 |
| 2.560 | 8.06 | -5.31 | 1.87 | 4.49 | -0.28 | 11.28 | -3.24 | 1.020 |
| 3.072 | 8.18 | -4.82 | 1.98 | 4.79 | 0.00 | 10.76 | -3.10 | 1.369 |

The potential collision is evaluated by constructing the bounding box represented the range of obstacle's future position. Ideally, the range is calculated by $v_e^o \times t^u$, where $v_e^o$ denotes the estimated obstacle's velocity, $t^u$ denotes the time required for the UAV to reach the next proposed waypoint. However, there is no direct control of velocity for the implemented position controller. From Figure 6.11b, the maximum velocity is approximate $1.8m/s$ when the obstacle UAV is travelling from 0 to 1 in y-axis. Therefore, in dynamic obstacle avoidance, the maximum step size for waypoint calculation in each axis is restricted to 1 meter, with approximately $t^u \simeq 0.6s$.




(a) $t = 0.001s$ (b) $t = 0.04s$

(c) $t = 0.512s$ (d) $t = 2.048s$

Figure 6.11: Comparison between true velocity and the estimated velocity with various $t$

Furthermore, as it is stated in Chapter 4.3.3, the bounding box should be extended with extra length to allow UAV's rotation and compensate the estimation error caused by the change of obstacle's velocity. The experiment results in Figure 6.11b shows that with a duration of $t = 0.04s$, the algorithm can sufficiently keep track of any acceleration.

However, there is still error for the estimated velocity, to minimise the effects caused by the estimation. The velocity $v_e^o$ used to predict obstacle's future position should be the addition of estimated velocity and the maximum velocity difference in Table 6.4. For duration $t = 0.04$, the maximum difference is $8.87m/s$ occurs approximately at $18.25s$ of the simulation, with the true velocity $10.51m/s$ and estimated velocity $1.64m/s$[1]. The difference differs significantly because the implemented obstacle is travelling from 15 to 25, the 10-meter displacement has reached the threshold of the maximum traversing distance supported by the position controller, which cause instability of the obstacle UAV's state. Hence for the implemented obstacle UAV, the maximum traversing step is restricted to 5 meters, with maximum velocity difference of $2.20m/s$.

### 6.2.2 Alternative waypoint searching

When the initally proposed waypoint is considered to be in potential collision with the moving obstacle. the algorithm will search for an alternative collision-free trajectory by assigning a waypoint towards one of the four corners of the constructed bounding box in 2D. Figure 6.12 shows the 2D overview of the scenarios that the potential trajectory potentially collides with the dynamic obstacle.

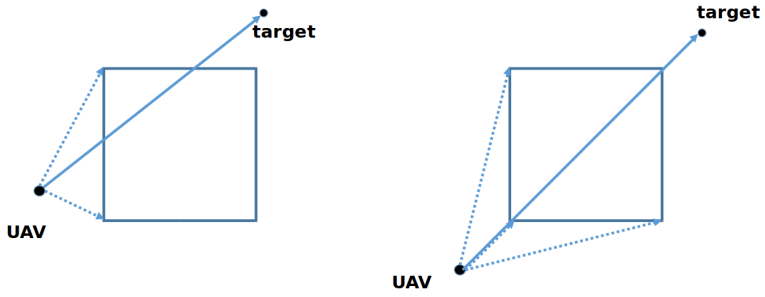



Figure 6.12: 2D overview of potential collision scenarios

As it is illustrated in the Figure, there are two or three corners are safe for the UAV to travel without colliding with the obstacle, which can be determined by calculating the distance from UAV's current position to each of the four corners, where the shortest

[1] the number does not add up because the estimated velocity is taken at $t = 18.17$

two are the safe options. For simplification of the algorithm, it will check if there are two corners lie on the line between UAV's current position and the target location, for the special case when the potential trajectory intersects with two diagonal corners of the square bounding box, which is relatively rare in a real-world scenario.

With the two safe options determined, the algorithm need determine the appropriate one based on the displacement from UAV's current position to each of these two corners, which are denoted by $[d_x^{bbx1}, d_y^{bbx1} d_x^{bbx2}, d_y^{bbx2}]$. Although it is possible for the UAV to locate exactly on one or two (at the corner) of the four boundaries, this special scenario is relatively rare. Furthermore, the accuracy for the implemented position controller is on the level of $1 \times 10^{-20}$, which makes it nearly possible for the UAV to locate on the bounding box's boundary, the algorithm will drive the UAV into hovering mode, to stay at its current location, wait for change of position from the moving obstacle. The implemented algorithm consider only for the scenarios that the UAV locates outside of the bounding box.

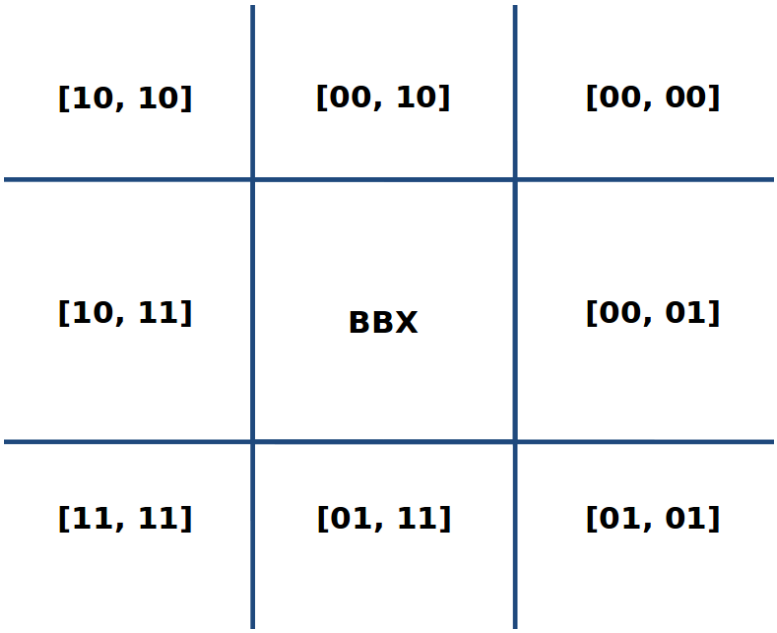
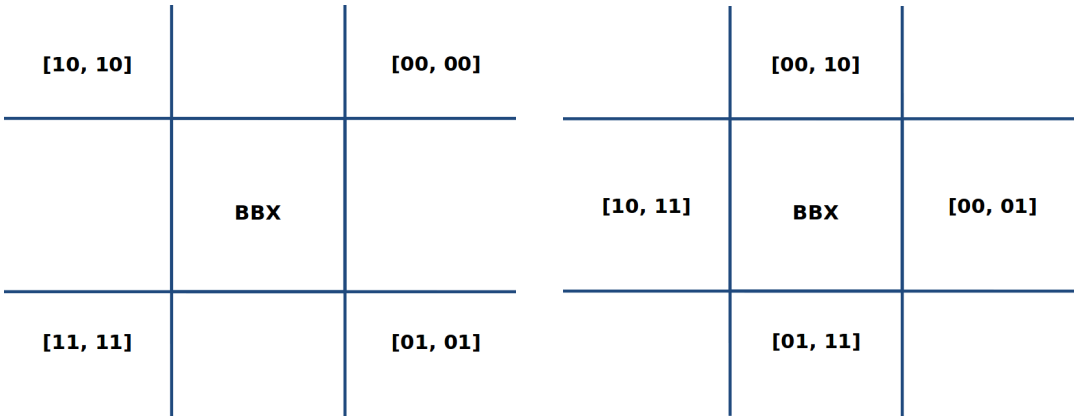



Figure 6.13: Possible displacement combination for $[d_x^{bbx1} d_y^{bbx1}, d_x^{bbx2} d_y^{bbx2}]$ in 2D,where 1 denotes positive value and 0 denotes negative value

Figure 6.13 shows there are eight possible combinations of how the UAV locate out-

side of the bounding box, where 01 denotes the direction from the UAV to two of the shortest corner, with 0 representing the negative direction and 1 representing a positive direction. Because the UAV is located outside of the bounding box, $[d_x^{bbx1} d_y^{bbx1}]$ and $[d_x^{bbx2} d_y^{bbx2}]$ will at least share one direction by evaluating each axis separately and maximum two same direction. Figure 6.14 groups the UAV's position with respect to the bounding box base on the number of the same direction.

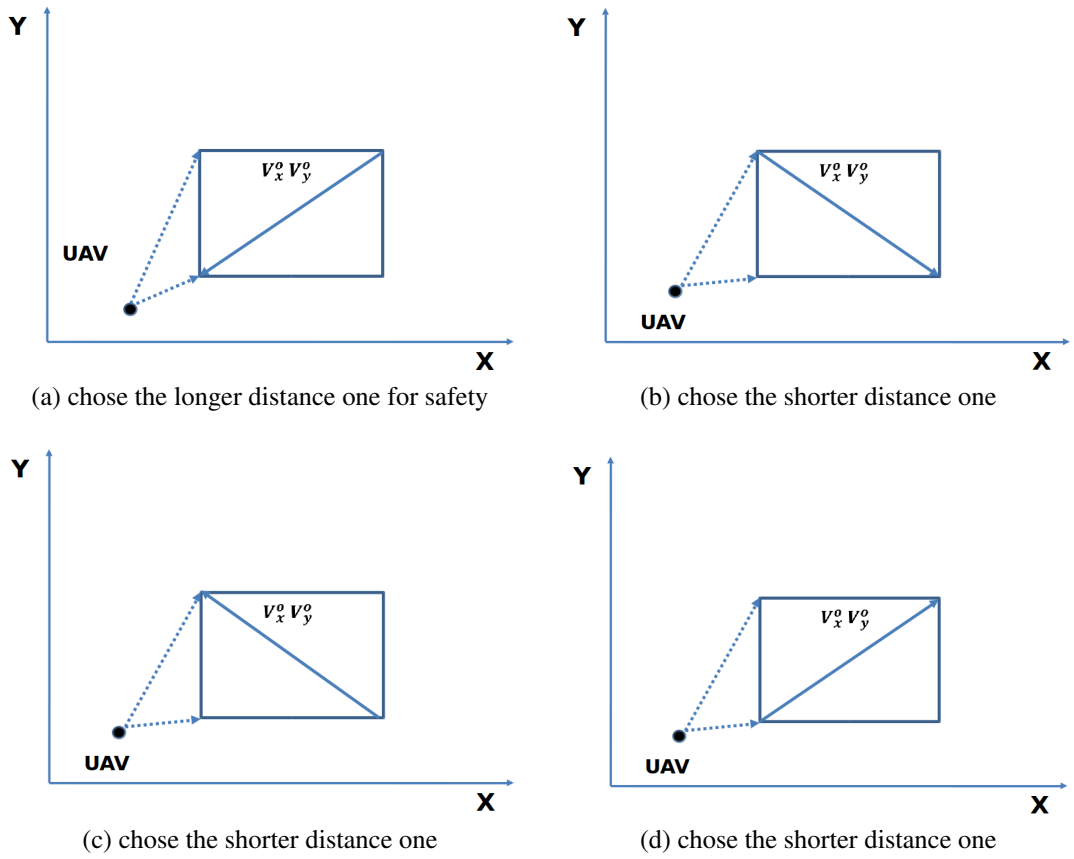



(a) sharing two directions (b) sharing one direction

Figure 6.14: Classification for $[d_x^{bbx1} d_y^{bbx1}, d_x^{bbx2} d_y^{bbx2}]$ base on their direction

**Alternative waypoint searching: two same directions**

When both $d_x^{bbx1}, d_x^{bbx2}$ and $d_y^{bbx1}, d_y^{bbx2}$ share the same direction, the algorithm will compare the distance from these two corners to the target location, where the corner with a shorter distance to the target is chosen to be the next alternative waypoint. However, when the both the estimated velocity $v_x^o, v_y^o$ have the opposite sign compared to the direction vector, as it is shown in Figure 6.15a, the corner with shorter distance to the target will result in the UAV and obstacle traversing toward each other, the corner with longer distance will be chosen to be the next alternative waypoint to minimise the potential collision.




(a) chose the longer distance one for safety

(b) chose the shorter distance one

(c) chose the shorter distance one

(d) chose the shorter distance one

Figure 6.15: Determine the direction for the alternative next waypoint when the direction to both corner share the same direction

**Alternative waypoint searching: one same direction**

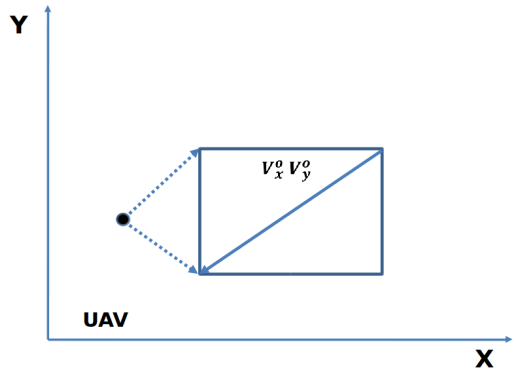
When both $d_x^{bbx1}, d_x^{bbx2}$ and $d_y^{bbx1}, d_y^{bbx2}$ share only one same direction, the algorithm determines the next alternative waypoint base on the number of the opposite direction those two vector share with the estimated obstacle's velocity. The prefered corner would have two opposite direction, the second preferable choice is the one with only one opposite sign. This is chosen to allow the UAV to travel in the opposite direction of the moving obstacle to minimise the potential collision. Figure 6.16 illustrates the scenarios when there is only one same direction, and the explanation of determined corner.

**Summary**

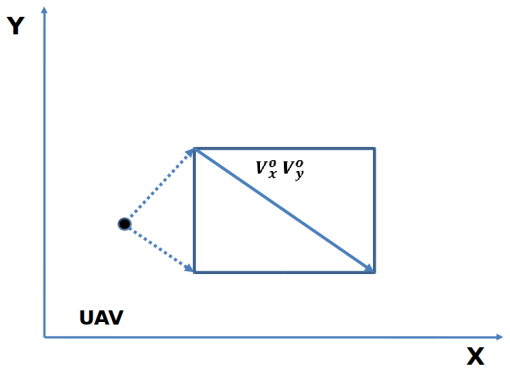
The alternative waypoint is determined based on the relationship between the estimated obstacle 2D velocity and the direction from UAV's current position to two of the safe corners of the bounding box. Although it is a relatively rare case when the estimated velocity is zero in any of the 2d axes, it can be improved by repeat the described process by only comparing the direction and estimated velocity in 1D.

For the case when the obstacle moves only in the z-axis, the algorithm will search for the alternative waypoint with the shortest 2D distance to the target location, and avoid the dynamic obstacle by going the opposite direction of the estimated z-axis velocity.

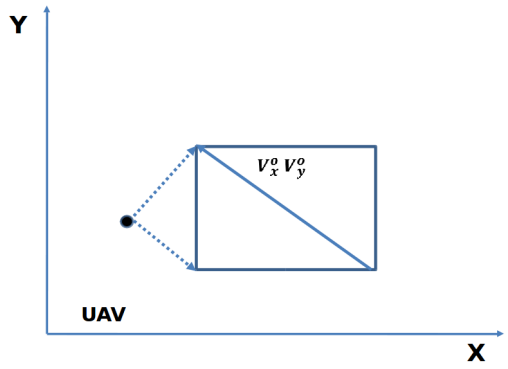
Furthermore, the proposed alternative waypoint searching technique only provides a 2D coordinate, the value for next alternative waypoint in z-axis can be either implemented toward the target location for fast converging or towards the opposite direction of estimated obstacle velocity to minimise potential risk. For the implemented algorithm, the z value for the next alternative waypoint is chosen to be the same as the UAV's current position.



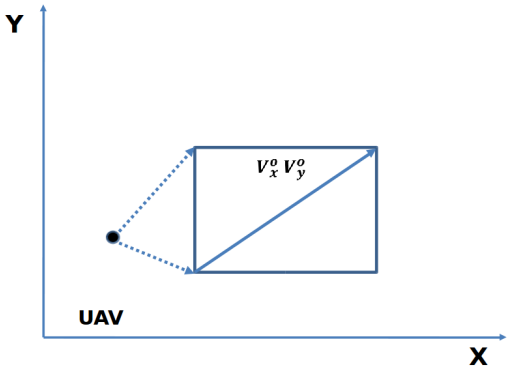

(a) chose the top-left corner with two opposite direction compared to the obstacle




(b) chose the top-left corner with one opposite direction compare to the obstacle




(c) chose the bottom-left corner with two opposite direction compare to the obstacle




(d) chose the bottom-left corner with one opposite direction compare to the obstacle

Figure 6.16: Determine the direction for the alternative next waypoint when only one direction to both corner share the same direction

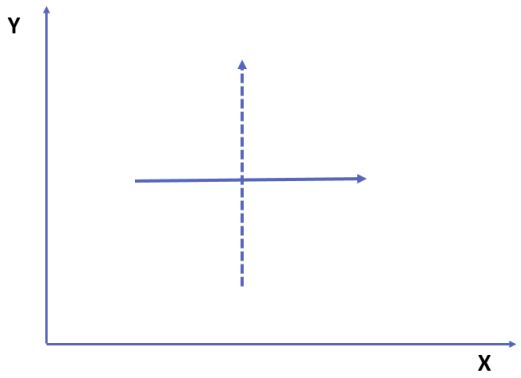
### 6.2.3 Experiment design

Table 6.5 shows the start and finish coordinate of UAV and the moving obstacle, for the five-set experiments designed to test the algorithm's feasibility of dynamic obstacle avoidance. The UAV is expected to find a path from the proposed start position to the target location by avoiding the dynamic obstacle. The obstacle's trajectory is designed to be in different relative motions compared to the UAV's potential trajectory, which includes:

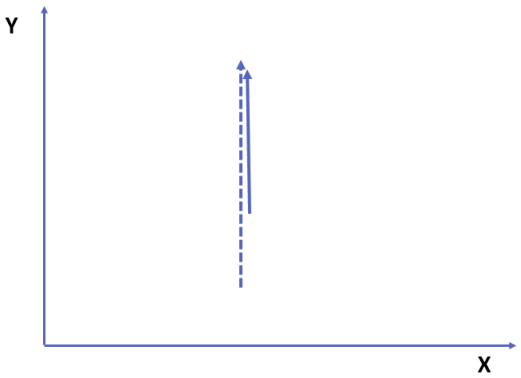
- Case 1, the dynamic obstacle's trajectory is perpendicular to the UAV's potential trajectory.
- Case 2, the dynamic obstacle's trajectory is parallel and same direction to the UAV's potential trajectory.
- Case 3, the dynamic obstacle's trajectory is parralle and opposite to the UAV's potential trajectory.
- Case 4, the dynamic obstacle's trajectory is parallel to the UAV's potential trajectory with a turning angle.
- Case 5, the dynamic obstacle's trajectory is opposite to the UAV's potential trajectory with a turning angle.

Table 6.5: Cases designed for dynamic obstacle avoidance with start and finish coordinate for both UAV and moving obstacle

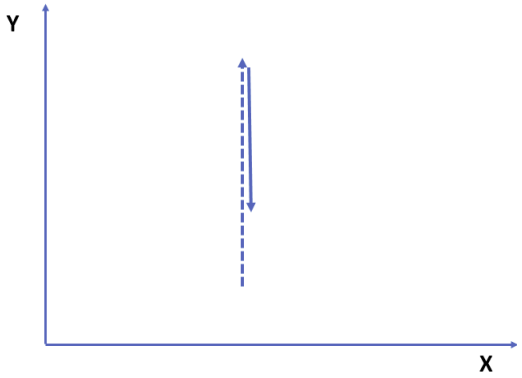
| **CASE** *No.* | **UAV start** **x, y, z** (*m*) | **UAV target** **x, y, z** (*m*) | **obstacle start** **x, y, z** (*m*) | **obstacle target** **x, y, z** (*m*) |
|---|---|---|---|---|
| 1 | [0, -6, 2] | [0, 4, 2] | [-6, 0, 2] | [6, 0, 2] |
| 2 | [0, -6, 2] | [0, 10, 2] | [0, 0, 2] | [0, 6, 2] |
| 3 | [0, -6, 2] | [0, 10, 2] | [0, 6, 2] | [0, 0, 2] |
| 4 | [0, -10, 2] | [0, 10, 2] | [-3, -3, 2] | [3, 3, 2] |
| 5 | [0, -10, 2] | [0, 10, 2] | [3, 3, 2] | [-3, -3, 2] |



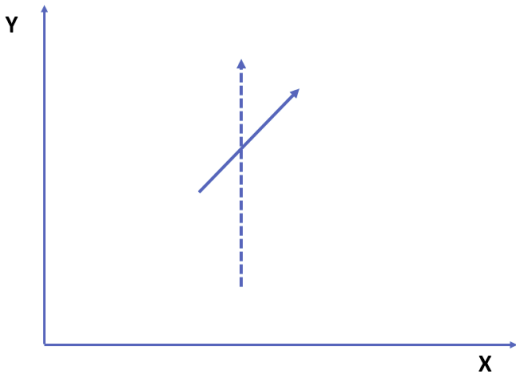

(a) Case.1, the dynamic obstacle's trajectory is perpendicular to the UAV's potential trajectory.



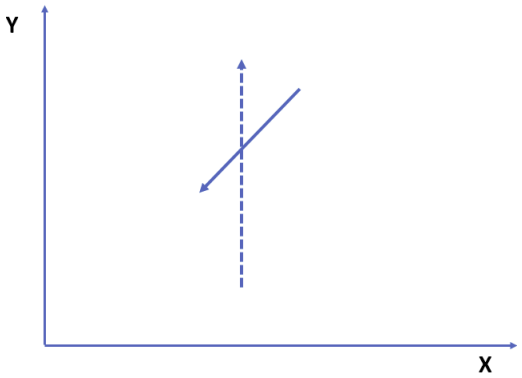

(b) Case 2, the dynamic obstacle's trajectory is parallel and same direction to the UAV's potential trajectory.




(c) Case 3, the dynamic obstacle's trajectory is parralle and opposite to the UAV's potential trajectory.




(d) Case 4, the dynamic obstacle's trajectory is parallel to the UAV's potential trajectory with a turning angle.




(e) Case 5, the dynamic obstacle's trajectory is opposite to the UAV's potential trajectory with a turning angle.

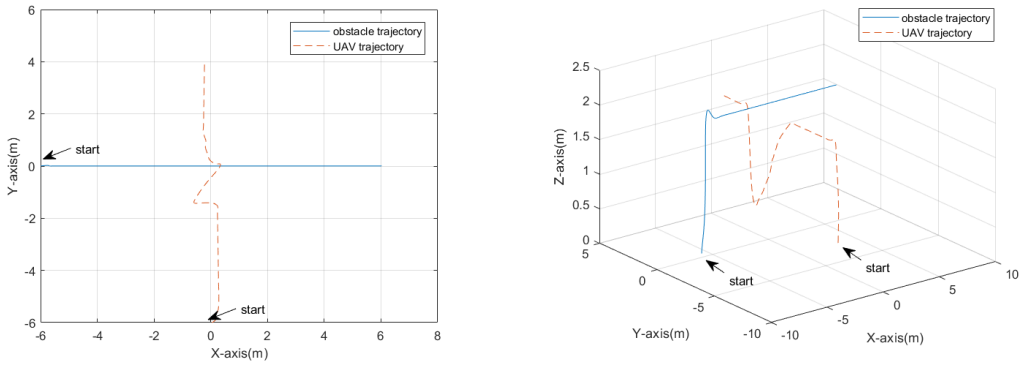
Figure 6.17: 2D overview of the obstacle's trajectory and UAV's potential trajectory, where the dashed arrow denotes the UAV's potential trajectory and solid arrow denotes the obstacle's trajectory

### 6.2.4 Result evaluation

This section includes the evaluation for comparison between the UAV executed trajectory and the obstacle's trajectory in both 2D and 3D for each dynamic obstacle avoidance case studies.

#### Case 1

As it is shown in Figure 6.18, the obstacle is travelling in positive x-axis direction from -6 to +6; the algorithm find a trajectory to avoid the obstacle by going to the opposite direction of the obstacle's trajectory.



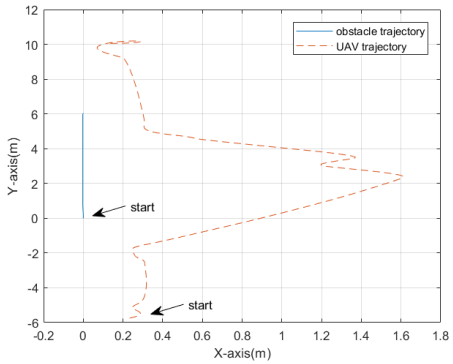

(a) 2D trajectory comparison for case 1

(b) 3D trajectory comparison for case 1

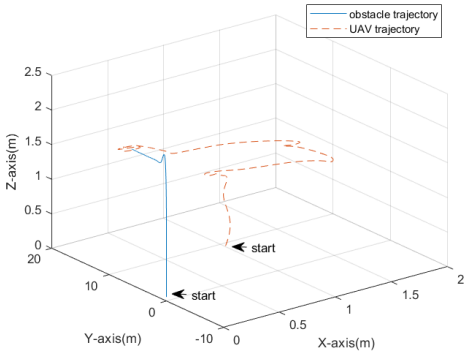
Figure 6.18: Case 1, obstacle and UAV's executed trajectory in both 2D and 3D

**Case 2 3**

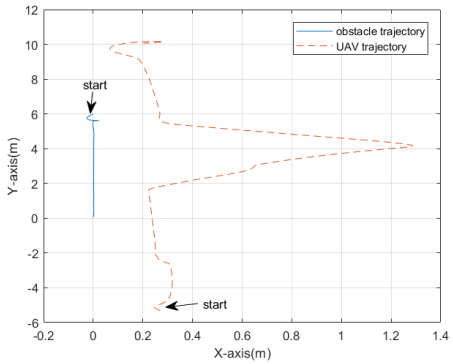
As it is shown in Figure 6.19, For case 2, 3, the obstacle is travelling from 0 to 6 and 6 to 0 in the y-axis, respectively. The algorithm determines the suitable corner of the bounding box by comparing the direction between the UAV's potential trajectory to one of the bounding box's corner and the obstacle's velocity and chose the corner with a higher number of the shared opposite direction. For case 2 and 3, both the two shortest distance corner have the same number of opposite direction (1 for case 2 and 0 for case 3), the algorithm then chose the corner base on the distance to the corner, hence, avoid the obstacle by going in the positive x-axis direction.



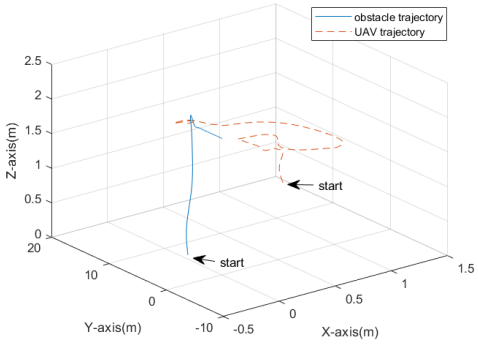

(a) 2D trajectory comparison for case 2




(b) 3D trajectory comparison for case 2




(c) 2D trajectory comparison for case 3

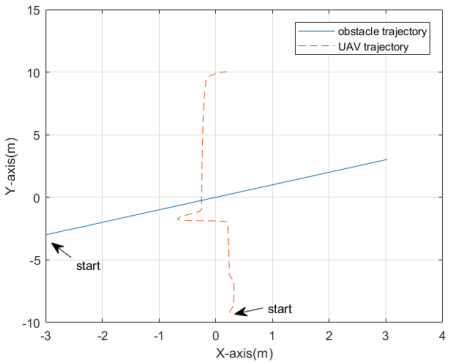



(d) 3D trajectory comparison for case 3

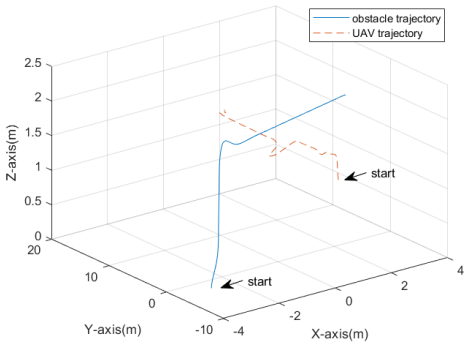
Figure 6.19: Case 2 3, obstacle and UAV's executed trajectory in both 2D and 3D

**Case 4 5**

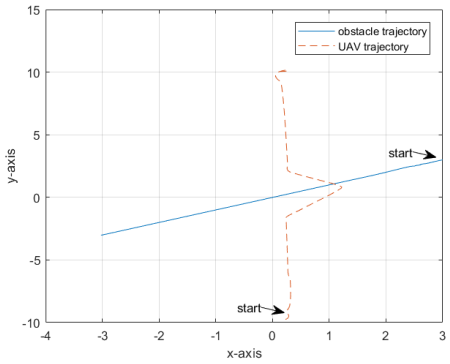
As it is shown in Figure 6.20, For case 2, 3, the obstacle is travelling from [-3, -3] to [3, 3] and [3, 3] to [-3, -3] respectively. Similarly to case 2 and 3, the UAV avoids the obstacle by going to one of the bounding box corners with a higher number of opposite direction compared to the obstacle's velocity. For case 4, the obstacle is travelling in the positive x-axis direction and positive y-axis direction, the algorithm finds a collision-free path by going to the negative direction on both x and y axes. For case 5, the obstacle is travelling in the x-axis direction and negative y-axis direction, the algorithm finds a collision-free path by going to the positive direction in both x and y axes.



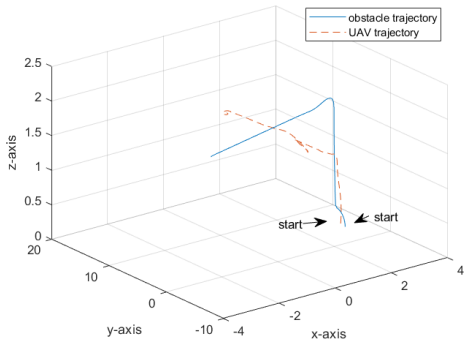

(a) 2D trajectory comparison for case 4




(b) 3D trajectory comparison for case 4




(c) 2D trajectory comparison for case 5




(d) 3D trajectory comparison for case 5

Figure 6.20: Case 4 5, obstacle and UAV's executed trajectory in both 2D and 3D

### 6.2.5 Conclusions

This section includes the discussion and evaluation for the performance of the dynamic obstacle ability, where the algorithm is expected to find a collision-free trajectory within a 3D unknown environment with one dynamic obstacle. The implemented algorithm focuses only on the feasibility of the dynamic obstacle avoidance ability, provided with the historical position of the dynamic obstacle.

The algorithm works on the basis of predicting the obstacle's future range of position by estimating the obstacle's velocity from its historical movement trajectory within the past duration of $t$. The choice of $t$ affects the accuracy of velocity estimation. Generally, smaller $t$ yields a better estimation of the obstacle's velocity where there is a change of the obstacle's velocity. However, when the $t$ is extremely small, the estimated velocity differs significantly from its real velocity, this is due to the $t$ exceeds the minimum duration required for the system to process data. From the experiments result, the ideal range of $t$ is determined to be within $[0.04 \rightarrow 0.512s]$.

The potential collision between the UAV and the dynamic obstacle is determined with SAT techniques, which states that if there is a straight line between two polygons, these two polygons do not intersect. These two polygons are constructed as cubical bounding boxes with the UAV and the predicted obstacle's future displacement in each axis.

In order to search for the alternative waypoint when the initial one could potentially collide with the dynamic obstacle, the algorithm will propose the next alternative waypoint towards the direction of one of the four corner of the constructed bounding box in 2D, which is calculated base on the directions between the estimated obstacle's velocity, and the displacement from the UAV's current position. The solution in the z-axis is configurable, either with the opposite direction of the estimated obstacle velocity for maximum safety, or the same direction to the target location for fast converging.

From the tested result, the UAV is proven to be able to avoid the potential collision for dynamic obstacle by predicting its future range of position from its historical trajectory. However, there are still several aspects that can be improved.

Firstly, the implemented algorithm is only tested within a single dynamic obstacle, and there are no other static obstacles within the navigation environment. The obstacles size and location can be easily acquired by searching the octomap data. For the cases when there are more than one moving obstacles or static obstacles, it will not be sufficient information for the UAV to predict each obstacles' position. Which could be improved by introducing a segmentation and feature tracking system for the received sensory data, to classify and identify between static and dynamic objects, treat them as static obstacles if there is a small or no change in the obstacles' position. For the scenarios where there is more than one dynamic obstacle, it is possible to apply the same method to check the collision by introducing multiple bounding boxes.

Secondly, the alternative trajectory is determined by assigning a waypoint towards one of the four corners of the bounding box constructed by the obstacle's future range of position, this work fine under the assumption that the UAV is only equipped with forward-viewing perception, where the obstacle will always reside within the range between the UAV and its target location. However, for the cases when the UAV is equipped with panoramic-view perception. It is possible that the resulting bounding box is located in the opposite direction from the UAV to its target location, by assigning a waypoint towards the bounding box corner would result in the low converging time to the target. To improve this, one possible solution is to determine whether the obstacle is approaching or departing away from the potential trajectory between the UAV's current position and target point, calculate a relative far waypoint but safe from the obstacle's trajectory when it is approaching, and shorter waypoint to the target location when the obstacle is moving away or in parallel with the potential trajectory. Furthermore, the implemented position which has no direct control of the UAV's velocity, the bounding box is predicted base on the estimated UAV's velocity, which is larger than the actual value to minimise the potential collision, the performance of the algorithm can be improved by manipulating the UAV's velocity.

# Chapter 7

# Conclusion and future works

## 7.1 Conclusion

The goal of the thesis is to design a collision-free autonomous navigation system with spatial awareness within a comprehensive simulation framework. While spatial awareness is the UAV's ability to acknowledge its position and orientation in relation to the surrounding environment, collision-free navigation means the UAV is required to find a trajectory to the target by avoiding any obstacles on the way, including both static and dynamic one, without any prior map information about the environment. The system should be real-world compatible and can be easily transferred to a real UAV. The implemented simulation framework consists of the necessary components required for the cognition tasks, which includes mapping, localisation, cognition and control system. The cognition system makes execution command based on the input data from mapping and localisation system, with information about the position information of the obstacle and UAV within a reference world frame. The mapping and localisation systems are modelled with noise to simulate real-world scenarios. The simulation framework is implemented in a modular way to test different strategies for control, mapping or localisation systems.

In order to test algorithms for autonomous UAV to work in a real environment, it is essential to build a comprehensive simulation framework with details as close as possible to

its real-world counterparts. The simulation framework is developed based on Rotors simulator package with the integration of Gazebo and ROS. ROS provides libraries and tools to facilitate the software development for the UAV autonomous navigation, and Gazebo is used for simulation, which includes a robust physical engine, high-quality graphics and convenient programmatic and graphical interface.

The development of the mapping system is divided into two steps: environment perception and map construction. The perception step utilises a depth camera to convert the simulated environment from Gazebo into ROS compatible point cloud data. Octomap is used for the map construction step, and it converts the processed point cloud data into a map representation. The environment is described with octree representation to identify the free, occupied and unknown subspaces.

For localisation system, an ideal odometry sensor with perfect ground truth data is used for the high-level tasks in the cognition system. However, experiments are designed to evaluate the performance of state estimation ability with EKF sensor fusion. The experiment focuses on the performance of the position estimation by fusing different combinations of IMU and odometry sensors. The UAV is programmed to execute a user-defined trajectory, and the estimated EKF filtered position data is compared with an ideal odometry sensor. The experiments results show that ekf_localisation node yields better performance by increasing the number of sensors used.

The control system is responsible for the decision made from the cognition system is executed, which is driving the UAV to a randomly assigned target with a prefered orientation. Each rotor from the UAV is modelled with motor dynamics, which accounts for the most critical effect for the control system. The implemented control system supports four DoF tracking of the UAV with the position in 3D and one heading direction.

The implementation for the cognition system is split into three parts with case studies for real-life scenarios, which are restricted area avoidance, static obstacle avoidance and dynamic obstacle avoidance.

Firstly, restricted area avoidance is implemented to examine the UAV's spatial aware-

ness ability. As the UAV is required to avoid a pre-defined rectangle-shaped restricted area by only taking input from the localisation system, the UAV needs accurate information about its location, therefore to exit the restricted area and approach to the target. Furthermore, As UAV adoption grows throughout the world, UAV operation is restricted from certain areas, such as an airport. Therefore, restricted area avoidance serves a fundament function for the autonomous UAV navigation system. The methodology for the implemented algorithm contains three steps.

- Perceive the UAV's localisation information, which includes the size and location of the restricted area.
- Check collision between the initially proposed trajectory and the restricted area data, However, it is futile and a waste of computer resource for the UAV to run the recursive function for the whole navigation process. A condition is given to the system so that the collision checker is only executed when the UAV is relatively close to the restricted area, to allow a fast false return. The collision checking result is compensated with the combined system error with additional buffer safe zone to maximise UAV's safety.
- Search for the alternative trajectory if the initial trajectory conflicts with the restricted area. The algorithm will search for the shortest exit strategy to avoid the restricted by comparing the UAV's position and target location.

By evaluating the experiment result, the UAV has successfully avoided the area by taking a relative shot path. However, the resulting trajectory found a limitation with the implemented Lee position controller, which will cause an unstable trajectory if the yaw change is too significant, which requires the initial attitude error less than 90 degrees to obtain the stability of the complete system, possible solution is to introduce an intermediate waypoint with smaller change the UAV' state. Methodology for other irregular shaped restricted area is discussed and could be implemented without a large amount of algorithm modification.

Secondly, for static obstacle avoidance, the UAV is required to avoid any static obstacles while reaching to the target without any prior map information. The algorithm is implemented similarly as the restricted area. The UAV will check collision for the perceived obstacle and search for an alternative trajectory if the initial one is determined to be dangerous. Different criteria are used for collision checking and alternative trajectory searching. For collision checking, instead of taking the predefined restricted area parameter from the user, the UAV will construct a bounding box according to the UAV's location and the initially proposed waypoint. Where the size of the bounding box is the minimum space required to allow the UAV to pass through, the bounding box space is also compensated with system error to simulated the uncertainties of real-life sensors. In order to search for the alternative trajectory, the UAV will again construct six bounding boxes but with only the UAV's position. The six bounding boxes are up, down, left, right, forward and back of the UAV's near-space. The algorithm will check the 'up' space by default, as the UAV is working within an outdoor environment, it is more likely to avoid the obstacle by going over it. The algorithm will then find the safe and close bounding box to exit if the 'up' space is unavailable.

From the RViz 3D visualisation and recorded experiment data from rosbag package, the algorithm is tested to be able to find a collision-free trajectory in a static environment, with the corrected parameter for octomap resolution, next waypoint calculation and additional length required to constructed the bounding box. Different experiments are designed to test how the next waypoint step size and bounding box construction can affect the converging time to the target location. From the experiment results, the next waypoint calculation impacts more significantly on the performance of the algorithm, less time is required for next waypoint with longer distance. However, by increasing the step size of the next waypoint, it causes the UAV to oscillate with unstable trajectory, which is probably caused the longer traversing distance has reached the maximum supported threshold of the implemented control system. The octomap map resolution affects the converging time heavily at the range between $[0.05 \rightarrow 0.15 meters]$, the navigation simulation reduces

significantly by increasing the map resolution. However, larger map resolution value does not provide a comprehensive representation of the environment when the UAV is working within a relatively crowded environment, the choice of map resolution should be determined according to the UAV's working environment. Furthermore, the algorithm failed to find an initial waypoint when the octomap resolution parameter is higher than three meters, which is caused the mapping system detects the ground land and treat it as an obstacle, and the UAV is contained by the occupied space when the resolution parameter is higher than the UAV's geometrical size. Lastly, the results show that the additional length extended has minimal effects on the converging time, which is possibly caused by the additional length is constrained by the maximum sensor range of the depth camera in the mapping system, which is only 10 meters, further investigation is required to investigate with a longer range depth camera.

Lastly, for the dynamic obstacle avoidance, the cognition system will first estimate the obstacle's trajectory by comparing the obstacle's current position and historical position with the past duration of $t$. The choice of the past duration $t$ affects on the accuracy of velocity estimation, where small value less than the processing time required for the ROS node to run will result in the estimated velocity differs significantly from its true value, and larger $t$ do not yield a good representation when there is any rapid change of the obstacle's velocity. From the experiment results, the duration $t$ is determined to be within the range of $[0.04 \rightarrow 0.512]$ to sufficiently estimate the obstacle's velocity. The algorithm then predicts the obstacle's future position range in each axis with the estimation of the obstacle's velocity and time required for the UAV to reach the next waypoint. However, as the implemented control system operates by tracking the UAV's position and orientation, with no direct control of velocity, the UAV's next waypoint size is restricted to a fixed size. Separating Axis Theorem is used to check collision between the UAV and dynamic obstacle. The theorem states that two polygons do not collide if it is possible to draw a line to separate them, the polygons are constructed with the measured maximum UAV velocity for that particular step size and the estimated obstacle velocity in each axis

respectively, with the measured time required for the UAV to travel for that particular step size. The maximum UAV velocity is selected due to the consequence of constructing a larger bounding box used for collision checking, therefore, minimise potential collision the dynamic obstacle. The algorithm evaluates and compares the gap distance between those two bounding boxes in each axis, and triggers potential collision if there is no gap in any of those axes.

When there is any potential collision detected between the UAV and the dynamic obstacle, the algorithm will search for a collision-free trajectory by choosing two of the shortest distance from the UAV to the corners of the bounding box constructed by the obstacle's velocity in 2D, then determines the appropriate one based on the direction between the estimated obstacle's velocity and the direction from UAV's current position toward the corresponding corner. The solution for the value in the z-axis is configurable, either with the opposite direction of the estimated velocity for maximum safety, or the same direction towards the target location. From the tested result, the algorithm is able to estimate the moving obstacle's velocity and avoid it by predicting its position. The algorithm is sufficient when there is only one obstacle. The future work will be the full implementation for multiple dynamics obstacles and developing a new position controller with direct control of UAV's velocity.

## 7.2 Contribution to knowledge

This thesis has presented a comprehensive simulation framework to test algorithms for the autonomous UAV navigation system. The navigation system consists of four sub-systems: mapping, localisation, cognition and control systems. The contribution includes the test of sensor fusion algorithm for localisation system and design of cognition algorithms for three case studies. Experiments are designed to evaluate the performance of state estimation by fusing different sensor combinations, and the results are concluded that EKF sensor fusion yields a better performance by increasing the number of sensors. For

cognition system, three cases were designed to test the feasibility of the algorithms to avoid a restricted area, static obstacle and dynamic obstacle. The experiments on the three cases have been conducted, and the UAV is able to reach the target under all the three cases of environments. All simulated components were designed to be analogous to their real-world counterpart. Ideally, it can be transferred to a real UAV without any changes. The simulation system provides a platform for future robotic research. As it is implemented in a modular way, it is easier to debug. Hence, the system has good reliability. Moreover, the system has good readability, maintainability and extendability.

## 7.3 Future works

The simulation framework for UAV autonomous navigation can be improved in several aspects. For the low-level task, the depth camera in mapping system detects and treat ground land as a static obstacle when the UAV is travelling in a low attitude, which is inefficient and waste of computational resource for the collision checking system to constantly trigger the collision alert, and caused algorithm failed to find the initial waypoint with low-resolution map(large resolution parameter). The mapping system could be improved by providing the ability to classify the ground land from the other obstacles. Furthermore, the dynamic obstacles avoidance is implemented as a stand-alone scenario, with only one dynamic obstacle and no static obstacles, which is not the case in real-world scenarios. Likewise, the mapping system should be able to classify the detects objects base on certain features, identify the static and dynamic obstacles based on their change of position. Future work on the mapping system includes segmentation and feature tracking algorithm to classify the type of objects.

The control system can be improved with the following three aspects. The implemented controller supports only four DoF tracking of the UAV's state, namely three positions and one heading direction, which is not sufficient for the case when the UAV involves orientation change. Secondly, the UAV tends to oscillate when there is a large change of

its state, due to the implemented position controller requires initial attitude error less than 90 degrees. Lastly, for the implemented dynamic obstacle avoidance, the obstacle's future position is predicted by the estimated velocity and the duration required for the UAV to reach the next waypoint, the implemented position controller has no direct control over the UAV's velocity, the duration is set a larger period to minimise the risk which could contain unnecessary space that being check. Furthermore, with direct control of UAV's velocity, the dynamic obstacle avoidance could be improved by manipulating the UAV's velocity rather than its position.

For localisation system, an ideal odometry sensor with perfect ground truth data is used for high-level tasks. In real-world scenarios, the sensor reading contains error and affects the accuracy of the UAV's state estimation. Possible improvement for localisation system would be fusing different sensor reading to get a better estimation of the UAV's state. This thesis includes the experiment design to estimate UAV's state with EKF sensor fusion; the accuracy of the estimation could be improved by identifying the initial noise covariance and process noise covariance of the sensor system.

For high-level tasks in cognition system, the implemented algorithm for restricted area avoidance only works properly when the restricted area is an axis-aligned rectangle-shaped area. The algorithm could be improved to deal with the restricted area which is not axis-aligned (discussed in Chapter 5.3) or any other shapes.

For high-level tasks in cognition system, the implemented algorithm for restricted area avoidance only works properly when the restricted area is an axis-aligned rectangle-shaped area. The algorithm could be improved to deal with the restricted area that is not axis-aligned (discussed in Chapter 5.3) or any other shapes.

For static obstacle avoidance algorithm, the experiment results indicate that the additional length extended for collision checking system has minimal effects on the converging time, which is possibly caused by the additional length is constrained by the maximum range of the depth camera from the mapping system, which is only 10 meters. Further investigation can be carried with a longer sensor range depth camera. Secondly,

the experiment results indicate that the map resolution has minimal effects when the resolution is greater than 0.15 meter and affects significantly when the resolution is smaller than 0.05 meter. A possible explanation is that the computational power required has reached the maximum power the machine is able to provide when the resolution is under 0.5 meter. Further study can be carried out by repeating the experiment on a real UAV to identify how the map resolution affects the algorithm's performance since real UAV, where the ob-board CPU power is much lower than the machine used for simulation. If there is evidence that map resolution affects significantly, it is preferable for the navigation to adapt its mapping resolution dynamically according to the working environment, with higher resolution for the crowded environment and low resolution for the environment with more open space. Furthermore, the algorithm utilises the map-less-based approach for static obstacle avoidance, which gives a better performance when the UAV is working in an unknown environment with relatively more open airspace. When the UAV is congested within a relatively complicated environment, the algorithm searches for an alternative waypoint by identifying the open space of its surrounding environment. The performance of the algorithm can be improved by switching to a map-using-based approach for the congested scenarios.

Lastly, for the dynamic obstacle avoidance, the algorithm is tested with only one dynamic obstacle. The algorithm can be improved to deal with multiple dynamic obstacles, with the improvement of segmentation and feature tracking algorithms from the mapping system. Secondly, the algorithm searches for the collision-free trajectory by assigning a waypoint towards one of the four corners of the bounding box by obstacle's movement. This technique is guaranteed to find a relatively short path to the destination when the UAV is equipped with a forward-view perception system, where the obstacle always resides within the range between the UAV and its target location. However, for the systems that are equipped with panoramic-view perception systems, it is possible that the resulting bounding box is located in the opposite direction from the UAV to its target location. Assigning a waypoint towards the bounding box corner would result in a low converging

time to the target location. The algorithm can be improved by comparing both the obstacle and UAV's position, velocity with respect to the target location, and avoid the obstacle by manipulating the UAV's velocity and the corresponding two displacements.

# References


[1] BAA Training, "Uav types: How to choose yours?" 2015, [Accessed 23-February-2019]. [Online]. Available: https://www.baatraining.com/uav-types-how-to-choose-yours/

[2] Brandon Bailey, "Amazon's new robot army is ready to ship," 2014, [Accessed 26-February-2019]. [Online]. Available: https://www.washingtontimes.com/news/2014/dec/1/amazons-new-robot-army-is-ready-to-ship/

[3] National Coordination Office, "Gps accuracy," [Online; accessed 23-February-2019]. [Online]. Available: http://www.gps.gov/systems/gps/performance/accuracy/

[4] L. Paull, S. Saeedi, M. Seto, and H. Li, "Auv navigation and localization: A review," *IEEE Journal of Oceanic Engineering*, vol. 39, no. 1, pp. 131–149, Jan 2014.

[5] F. Aurenhammer and R. Klein, "Voronoi diagrams," *Handbook of computational geometry*, vol. 5, pp. 201–290, 2000.

[6] T. Yu, J. Tang, L. Bai, and S. Lao, "Collision Avoidance for Cooperative UAVs with Rolling Optimization Algorithm Based on Predictive State Space," *Applied Sciences*, vol. 7, no. 4, p. 329, 2017. [Online]. Available: http://www.mdpi.com/2076-3417/7/4/329

[7] J. W. Park, H. D. Oh, and M. J. Tahk, "UAV collision avoidance based on geometric approach," in *2008 SICE Annual Conference*, aug 2008, pp. 2122–2126.

[8] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013, software available at http://octomap.github.com. [Online]. Available: http://octomap.github.com

[9] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Cham: Springer International Publishing, 2016, ch. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-26054-9_23

[10] T. Lee, M. Leoky, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *Decision and Control (CDC), 2010 49th IEEE Conference on*. IEEE, 2010, pp. 5420–5425.

[11] C. Ilas, "Electronic sensing technologies for autonomous ground vehicles: A review," in *2013 8TH INTERNATIONAL SYMPOSIUM ON ADVANCED TOPICS IN ELECTRICAL ENGINEERING (ATEE)*, May 2013, pp. 1–6.

[12] M. P. Groover, *Fundamentals of modern manufacturing: materials processes, and systems*. John Wiley & Sons, 2007.

[13] ——, "Automation," Aug 2018, [Accessed 23-February-2019]. [Online]. Available: https://www.britannica.com/technology/automation

[14] Marcusredden, "Bulk food dispenser bulk food dispenser holds gallons of food in clear container bulk food dispensers wholesale," 2019, [Accessed 23-February-2019]. [Online]. Available: http://marcusredden.com/pict/

[15] Andreas Bauer, "Automation — Wikipedia, the free encyclopedia," 2019, [Accessed 23-February-2019]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Automation&oldid=883858544

[16] L. R. Newcome, *Unmanned aviation: a brief history of unmanned aerial vehicles*. American Institute of Aeronautics and Astronautics, 2004.

[17] B. Yenne, *Attack of the drones: A history of unmanned aerial combat*. Zenith Imprint, 2004.

[18] J. Buckley, *Air power in the age of total war*. Routledge, 2006.

[19] T. Shima and S. Rasmussen, *UAV cooperative decision and control: challenges and practical approaches*. SIAM, 2009.

[20] DJI, "Dji phantom 4 pro v2.0 professional drone dji," [Accessed 26-February-2019]. [Online]. Available: https://www.dji.com/phantom-4-pro-v2

[21] Jennifer Harbster, "Civil war aeronautics," 2011, [Accessed 25-February-2019]. [Online]. Available: https://blogs.loc.gov/inside_adams/2011/07/civil-war-aeronautics/

[22] David Darling, "Kettering bug," [Accessed 26-February-2019]. [Online]. Available: http://www.daviddarling.info/encyclopedia/K/Kettering_Bug.html

[23] Carousell, "Parrot ar drone 1.0, electronics on carousell," [Accessed 26-February-2019]. [Online]. Available: https://sg.carousell.com/p/parrot-ar-drone-1-0-23134133/

[24] iRobot, "Robot lawn mower," [Accessed 26-March-2019]. [Online]. Available: https://www.irobot.com/about-irobot/company-information/robot-lawn-mower

[25] Sammy Jones, "Self-driving pod trials officially underway in milton keynes," 2018, [Accessed 26-February-2019]. [Online]. Available: https://www.miltonkeynes.co.uk/news/self-driving-pod-trials-officially-underway-in-milton-keynes-1-8423883

[26] amazon, "Amazon prime air," [Accessed 26-February-2019]. [Online]. Available: https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011

[27] Brian Barrett, "Inside the olympics opening ceremony world-record drone show," 2018, [Accessed 26-February-2019]. [Online]. Available: https://www.wired.com/story/olympics-opening-ceremony-drone-show/

[28] C. Xia, "Intelligent Mobile Robot Learning in Autonomous Navigation," Ph.D. dissertation, Ecole Centrale de Lille, 2015. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01298608

[29] A. Finn and S. Scheding, *Developments and challenges for autonomous unmanned vehicles*. Springer, 2012.

[30] A. H. Al-Kaff, "Vision-Based Navigation System for Unmanned Aerial Vehicles," Ph.D. dissertation, Charles III University of Madrid, 2017.

[31] M. Campbell, M. Egerstedt, J. P. How, and R. M. Murray, "Autonomous driving in urban environments: approaches, lessons and challenges," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 368, no. 1928, pp. 4649–4672, 2010. [Online]. Available: http://rsta.royalsocietypublishing.org/cgi/doi/10.1098/rsta.2010.0110

[32] A. Foka, "Predictive Autonomous Robot Navigation," Ph.D. dissertation, University of Crete, 2002.

[33] J. A. Adams, "Unmanned vehicle situation awareness: A path forward," in *Human systems integration symposium*, 2007, pp. 31–89.

[34] L. Chen, S. Wang, K. McDonaldMaier, and H. Hu, "Towards autonomous localization and mapping of auvs: a survey," *International Journal of Intelligent Unmanned Systems*, vol. 1, no. 2, pp. 97–120, 2013. [Online]. Available: https://doi.org/10.1108/20496421311330047

[35] M. Knudson and K. Tumer, "Adaptive navigation for autonomous robots," *Robotics and Autonomous Systems*, vol. 59, no. 6, pp. 410–420, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889011000248

[36] F. Bonin-Font, A. Ortiz, and G. Oliver, "Visual navigation for mobile robots: A survey," *Journal of Intelligent and Robotic Systems*, vol. 53, no. 3, p. 263, May 2008. [Online]. Available: https://doi.org/10.1007/s10846-008-9235-4

[37] J. Li, Y. Bi, M. Lan, H. Qin, M. Shan, F. Lin, and B. M. Chen, "Real-time simultaneous localization and mapping for uav: a survey," in *Proc. of International micro air vehicle competition and conference*, 2016, pp. 237–242.

[38] J. Kim, "Autonomous Navigation for Airborne Applications," Ph.D. dissertation, The University of Sydney, 2004.

[39] Z. Tao and W. Lei, "Sins and gps integrated navigation system of a small unmanned aerial vehicle," in *2008 International Seminar on Future BioMedical Information Engineering*. IEEE, 2008, pp. 465–468.

[40] National Coordination Office, "Gps accuracy," 2017, [Accessed 23-February-2019]. [Online]. Available: http://www.gps.gov/systems/gps/performance/accuracy/

[41] G. Schillaci and V. V. Hafner, "Random movement strategies in self-exploration for a humanoid robot," in *Proceedings of the 6th international conference on Human-robot interaction*. ACM, 2011, pp. 245–246.

[42] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, jul 1968. [Online]. Available: http://dx.doi.org/10.1109/TSSC.1968.300136

[43] E. Z. Gómez, "Map-building and planning for autonomous navigation of a mobile robot," Ph.D. dissertation, Center for Research and Advanced Studies of the National Polytechnic Institute, 2015.

[44] Argam Artashyan, "Xiaomi mi robot vacuum review," 2016, [Accessed 26-February-2019]. [Online]. Available: https://www.xiaomitoday.com/xiaomi-mi-robot-vacuum-review/

[45] T. Shima and S. Rasmussen, *UAV cooperative decision and control: challenges and practical approaches*. SIAM, 2009.

[46] H. Chen, X. m. Wang, and Y. Li, "A Survey of Autonomous Control for UAV," in *2009 International Conference on Artificial Intelligence and Computational Intelligence*, vol. 2, nov 2009, pp. 267–271.

[47] J. Craighead, R. Murphy, J. Burke, and B. Goldiez, "A survey of commercial open source unmanned vehicle simulators," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, April 2007, pp. 852–857.

[48] C. A. Theilmann, "Integrating autonomous drones into the national aerospace system," Ph.D. dissertation, University of Pennsylvania, 2015.

[49] A. G. N. C. dos Santos, "Autonomous Mobile Robot Navigation using Smartphone," Ph.D. dissertation, University of Lisbon, 2008.

[50] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[51] B. Grelsson, "Global pose estimation from aerial images: Registration with elevation models," Ph.D. dissertation, Linköping University Electronic Press, 2014.

[52] A. Amor-Martinez, A. Ruiz, F. Moreno-Noguer, and A. Sanfeliu, "On-board real-time pose estimation for uavs using deformable visual contour registration," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 2595–2601.

[53] Y. Wang, "An efficient algorithm for uav indoor pose estimation using vanishing geometry." in *MVA*, 2011, pp. 361–364.

[54] R. E. Bicking, "Features-fundamentals of pressure sensor technology-this tutorial addresses the basic physics of pressure, three types of pressure measurement, seven pressure sensor technologies, and three," *Sensors-the Journal of Applied Sensing Technology*, vol. 15, no. 11, pp. 30–43, 1998.

[55] C. Kanellakis and G. Nikolakopoulos, "Survey on computer vision for uavs: Current developments and trends," *Journal of Intelligent & Robotic Systems*, vol. 87, no. 1, pp. 141–168, Jul 2017. [Online]. Available: https://doi.org/10.1007/s10846-017-0483-z

[56] S. Gupte, P. I. T. Mohandas, and J. M. Conrad, "A survey of quadrotor unmanned aerial vehicles," in *2012 Proceedings of IEEE Southeastcon*, March 2012, pp. 1–6.

[57] C. Toth and G. Jóźków, "Remote sensing platforms and sensors: A survey," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 115, pp. 22–36, 2016.

[58] F. Remondino, "Heritage recording and 3d modeling with photogrammetry and 3d scanning," *Remote Sensing*, vol. 3, no. 6, pp. 1104–1138, 2011.

[59] K. Mth and L. Buoniu, "Vision and control for uavs: A survey of general methods and of inexpensive platforms for infrastructure inspection," *Sensors*, vol. 15, no. 7, pp. 14 887–14 916, 2015. [Online]. Available: http://www.mdpi.com/1424-8220/15/7/14887

[60] G. P. Stein, A. D. Ferencz, and O. Avni, "Estimating distance to an object using a sequence of images recorded by a monocular camera," Dec. 29 2015, uS Patent 9,223,013.

[61] M. Liang and X. Hu, "Recurrent convolutional neural network for object recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3367–3375.

[62] D. Hultqvist, J. Roll, F. Svensson, J. Dahlin, and T. B. Schn, "Detecting and positioning overtaking vehicles using 1d optical flow," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, June 2014, pp. 861–866.

[63] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, "Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerial vehicle," *Journal of Field Robotics*, vol. 33, no. 4, pp. 431–450, 2016.

[64] Y. Lu, Z. Xue, G.-S. Xia, and L. Zhang, "A survey on vision-based uav navigation," *Geo-spatial Information Science*, vol. 21, no. 1, pp. 21–32, 2018. [Online]. Available: https://doi.org/10.1080/10095020.2017.1420509

[65] J. C. Andersen, "Mobile robot navigation," Ph.D. dissertation, technical university of denmark, 2007.

[66] G. Reina, D. Johnson, and J. Underwood, "Radar sensing for intelligent vehicles in urban environments," *Sensors*, vol. 15, no. 6, p. 1466114678, Jun 2015. [Online]. Available: http://dx.doi.org/10.3390/s150614661

[67] S. Wu, S. Decker, P. Chang, T. Camus, and J. Eledath, "Collision sensing by stereo vision and radar sensor fusion," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 4, pp. 606–614, 2009.

[68] S. Kim, Z. J. Chong, B. Qin, X. Shen, Z. Cheng, W. Liu, and M. H. Ang, "Cooperative perception for autonomous vehicle control on the road: Motivation and experimental results," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov 2013, pp. 5059–5066.

[69] P. M. Møst, "Visual navigation of an autonomous drone," Master's thesis, Institutt for datateknikk og informasjonsvitenskap, 2014.

[70] Y. Li, H. Chen, M. J. Er, and X. Wang, "Coverage path planning for uavs based on enhanced exact cellular decomposition method," *Mechatronics*, vol. 21, no. 5, pp. 876 – 885, 2011, special Issue on Development of Autonomous Unmanned Aerial Vehicles. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957415810001893

[71] S. Chutia, N. M. Kakoty, and D. Deka, "A review of underwater robotics, navigation, sensing techniques and applications," in *Proceedings of the Advances in Robotics*. ACM, 2017, p. 8.

[72] B. Li, C. Mu, and B. Wu, "A survey of vision based autonomous aerial refueling for unmanned aerial vehicles," in *2012 Third International Conference on Intelligent Control and Information Processing*, July 2012, pp. 1–6.

[73] G. N. DeSouza and A. C. Kak, "Vision for mobile robot navigation: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 2, pp. 237–267, 2002.

[74] M. S. Gzel, "Autonomous vehicle navigation using vision and mapless strategies: A survey," *Advances in Mechanical Engineering*, vol. 5, p. 234747, 2013. [Online]. Available: https://doi.org/10.1155/2013/234747

[75] S. Atiya and G. D. Hager, "Real-time vision-based robot localization," *IEEE Transactions on Robotics and Automation*, vol. 9, no. 6, pp. 785–800, 1993.

[76] Y. Matsumoto, M. Inaba, and H. Inoue, "Memory-based navigation using omni-view sequence," in *Field and Service Robotics*, A. Zelinsky, Ed. London: Springer London, 1998, pp. 171–178.

[77] S. Edelkamp and S. Schrödl, *Route Planning and Map Inference with Global Positioning Traces*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 128–151. [Online]. Available: https://doi.org/10.1007/3-540-36477-3{_}10

[78] H. P. Moravec, "The stanford cart and the cmu rover," CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, Tech. Rep., 1983.

[79] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte *et al.*, "Minerva: A second-generation museum tour-guide robot," in *Robotics and automation, 1999. Proceedings. 1999 IEEE international conference on*, vol. 3. IEEE, 1999.

[80] M. V. Srinivasan, "An image-interpolation technique for the computation of optic flow and egomotion," *Biological Cybernetics*, vol. 71, no. 5, pp. 401–415, Sep 1994. [Online]. Available: https://doi.org/10.1007/BF00198917

[81] J. Santos-Victor, G. Sandini, F. Curotto, and S. Garibaldi, "Divergent stereo for robot navigation: Learning from bees," in *Computer Vision and Pattern Recognition, 1993. Proceedings CVPR'93., 1993 IEEE Computer Society Conference on*. IEEE, 1993, pp. 434–439.

[82] C. Zhou, Y. Wei, and T. Tan, "Mobile robot self-localization based on global visual appearance features," in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 1. IEEE, 2003, pp. 1271–1276.

[83] D. Kim and R. Nevatia, "Recognition and localization of generic objects for indoor navigation using functionality," *Image and Vision Computing*, vol. 16, no. 11, pp. 729–743, 1998.

[84] ——, "Symbolic navigation with a generic map," *Autonomous Robots*, vol. 6, no. 1, pp. 69–88, 1999.

[85] B. J. a. N. Guerreiro, "Sensor-based Control and Localization of Autonomous Vehicles in Unknown Environments," Ph.D. dissertation, UNIVERSIDADE DE LISBOA, 2013.

[86] P. E. Hart, N. J. Nilsson, and B. Raphael, "Correction to: "A Formal Basis for the Heuristic Determination of Minimum Cost Paths"," *SIGART Bull.*, no. 37, pp. 28–29, dec 1972. [Online]. Available: http://doi.acm.org/10.1145/1056777.1056779

[87] Y.-h. Qu, Q. Pan, and J.-g. Yan, "Flight path planning of UAV based on heuristically search and genetic algorithms," in *31st Annual Conference of IEEE Industrial Electronics Society, 2005. IECON 2005.*, nov 2005, pp. 5 pp.—-.

[88] P. Bhattacharya and M. L. Gavrilova, "Roadmap-based path planning - using the voronoi diagram for a clearance-based shortest path," *IEEE Robotics Automation Magazine*, vol. 15, no. 2, pp. 58–66, June 2008.

[89] C. Goerzen, Z. Kong, and B. Mettler, "A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance," *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1, p. 65, nov 2009. [Online]. Available: https://doi.org/10.1007/s10846-009-9383-1

[90] J. Krozel and M. Peters, "Strategic conflict detection and resolution for free flight," in *Proceedings of the 36th IEEE Conference on Decision and Control*, vol. 2, dec 1997, pp. 1822—-1828 vol.2.

[91] H. Y. Ong and M. J. Kochenderfer, "Short-term conflict resolution for unmanned aircraft traffic management," in *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, sep 2015, pp. 5A4—-1—-5A4—-13.

[92] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[93] F. Kendoul, "Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems," *Journal of Field Robotics*, vol. 29, no. 2, pp. 315–378, 2012. [Online]. Available: http://dx.doi.org/10.1002/rob.20414

[94] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, 2014.

[95] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986. [Online]. Available: https://doi.org/10.1177/027836498600500106

[96] J. van Tooren, M. Heni, A. Knoll, and J. Beck, "Development of an autonomous avoidance algorithm for UAVs in general airspace," in *Proceedings of First CEAS European Air and Space Conference*. Citeseer, 2007.

[97] R. V. Kulkarni and G. K. Venayagamoorthy, "Bio-inspired algorithms for autonomous deployment and localization of sensor nodes," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 6, pp. 663–675, Nov 2010.

[98] A. Nguyen and B. Le, "3d point cloud segmentation: A survey," in *2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, Nov 2013, pp. 225–230.

[99] B. Renfro, "An analysis of global positioning system (gps) standard positioning system (sps) performance for 2016," 2017.

[100] L. Serrano, D. Kim, R. B. Langley *et al.*, "A gps velocity sensor: How accurate canitbe," *A First Look*, 2004.

[101] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, July 2014.

[102] L. DAlfonso, W. Lucia, P. Muraca, and P. Pugliese, "Mobile robot localization via ekf and ukf: A comparison based on real data," *Robotics and Autonomous Systems*, vol. 74, pp. 122 – 127, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889015001517

[103] P. Martin and E. Salaun, "The true role of accelerometer feedback in quadrotor control," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1623–1629.

[104] M. W. Achtelik, "Advanced closed loop visual navigation for micro aerial vehicles," Ph.D. dissertation, ETH Zurich, 2014.

# Appendix A

# Appendix

## A.1 Development environment setup

This section describes the setup process for the prerequisites for the experiment, which include the integration between ROS and gazebo and the installation of rotors package. At the end of the chapter, a brief overview is given to describe the basic usage to work under ROS-Gazebo environment.

### A.1.1 ROS Gazebo integration

The Robot Operating System (ROS) is a flexible framework for writing robot software. It provides libraries and tools to help software developer to design complex robot applications within different robotic platforms. Currently, Ros only operates on Unix-based operating systems.This simulation framework has only been tested under ROS kinetic under Ubuntu 16.04. There are many different libraries and tools in ROS, and The most straightforward way is to install the full configuration provided by ROS which includes ROS, rqt, RVIZ, robot-generic libraries, 2D/3D simulators, navigation, and 2D/3D perception.

```
$ sudo apt-get install ros-kinetic-desktop-full
```

Gazebo is a well-designed 3D indoor/outdoor multi-robot simulator, complete with dynamic and kinematic physics, and a pluggable physics engine. It offers the ability to accurately and efficiently simulate the robot population in a complex environment and allows users to rapidly test algorithms using realistic scenarios without access to expensive hardware.

```
$ curl -sSL http://get.gazebosim.org | sh
```

### A.1.2 Basic concept and usage

**Package** Software in ROS is organized in packages. A package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module.

**ROS node** A node is a process that performs the computation. Nodes are combined into a graph and communicate with one another using streaming topics, services, and the parameter server. A robot control system usually comprises many nodes. For example, a node takes input from the sensor and convert the data into relevant mapping data, or a node takes position command and controls the robot's wheel motor. It provides additional fault tolerance as crashes are isolated to individual nodes. All running nodes are allocated with graph resource name which that uniquely identifies them to the rest of the system.

**ROS topic** Topic is named buses over which nodes exchange messages. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption. In general, nodes are not aware of who they are communicating with. Instead, nodes that are interested in data subscribe to the relevant topic; nodes that generate data publish to the relevant topic. There can be multiple publishers and subscribers to topics

**ROS message** Node communicates with each other by publishing messages to topics. A message is a simple data structure, comprising typed fields, Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays.

**ROS service** ROS uses a simplified service description language (”srv”) to describe ROS service types. It is built directly upon the ROS message formation to enable request/response communication between nodes. The service description is stored in .srv file of a package.

**ROS bag** A bag is a file format in ROS for storing ROS message data. It serves an important role as it allows the user to store, process, analyse and visualise them.

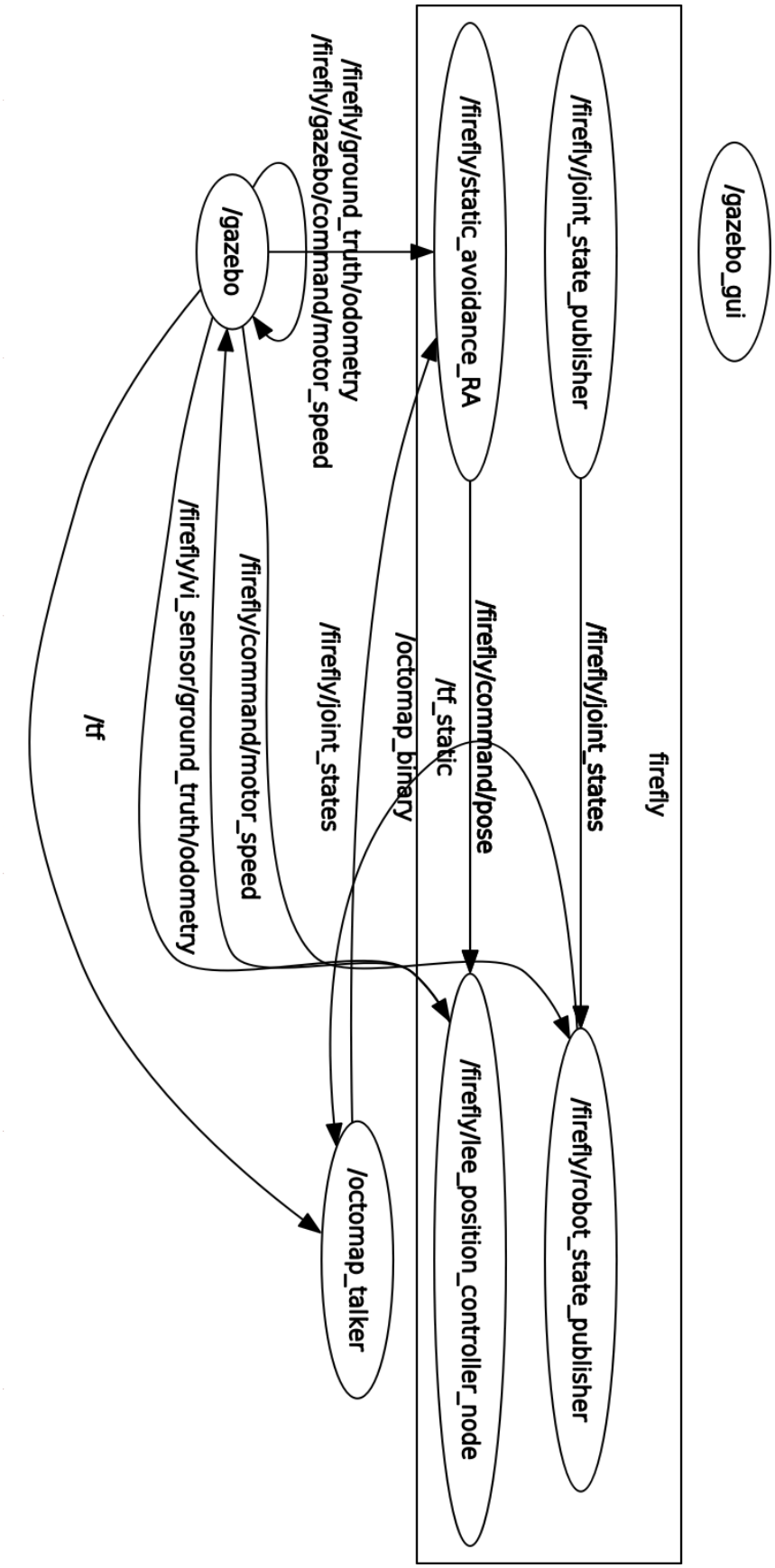
**ROS publisher** ros publisher is a ROS node that broadcasting a ROS message.

**ROS subscriber** ros subscriber is a ROS node that broadcasting a ROS message.

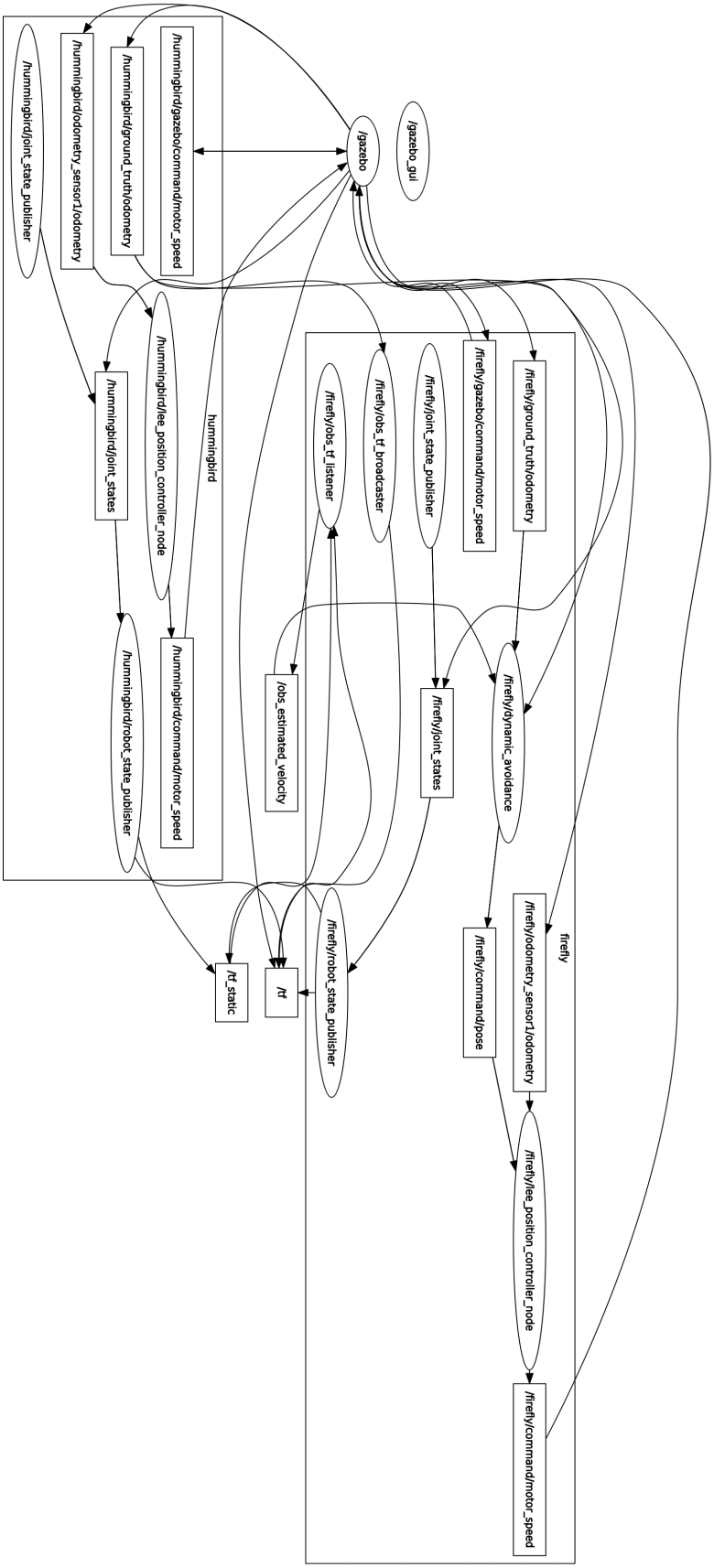
**Utilising Launch files** Launch files are scripts which describes how the nodes should be executed, parameters that are supposed to be set, along with other attributes of launching multiple ROS nodes. The roslaunch package contains the roslaunch tools. The roslaunch package introduces a simple way to launch multiple ROS nodes within a single file, as well as identifying parameters on the Parameter Server by reading in one or more xML configuration files (with the .launch extension) that specify the parameters to set and nodes to launch, as well as the machines that they should be run on. These parameters will be stored on the Parameter Server before any nodes are launched.
to launch a lauch file, entering the following command.

```
$ roslaunch PACKAGE_NAME LAUNCH_FILE.launch
```

## A.2 ROS node graph for static obstacle avoidance with restricted area avoidance

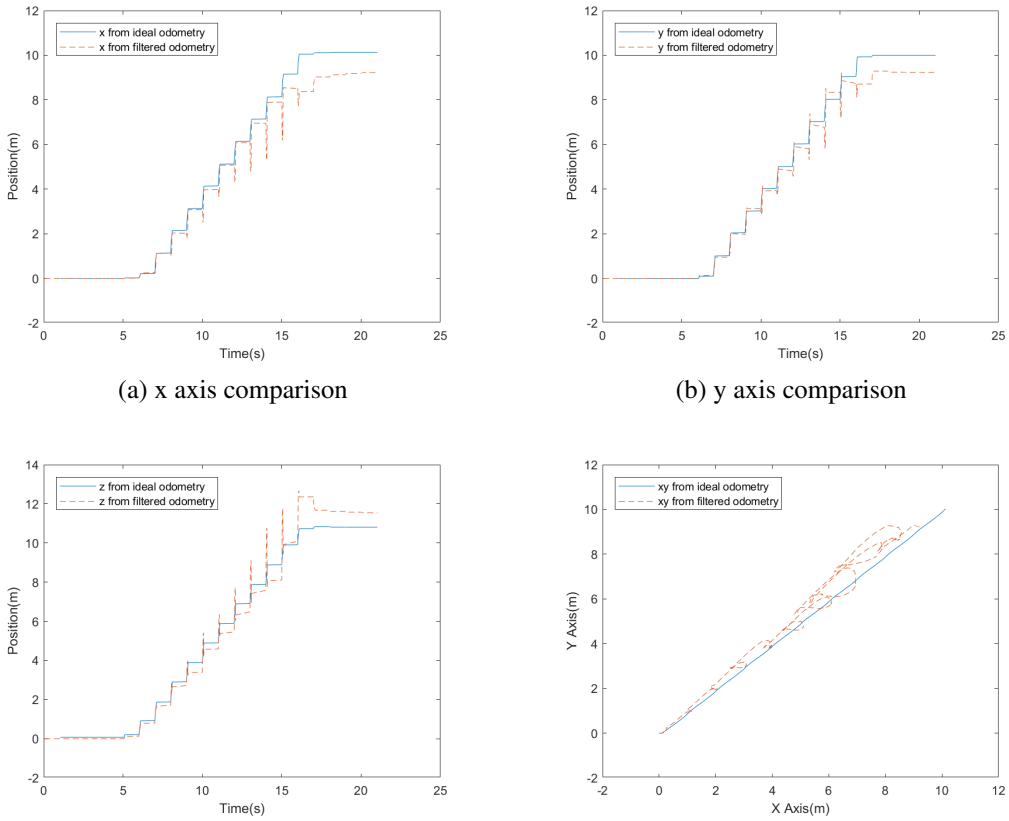


## A.3 ROS node graph for dynamic obstacle avoidance



## A.4 Localisation results with EKF sensor fusion

### A.4.1 1 imu 1 odometry



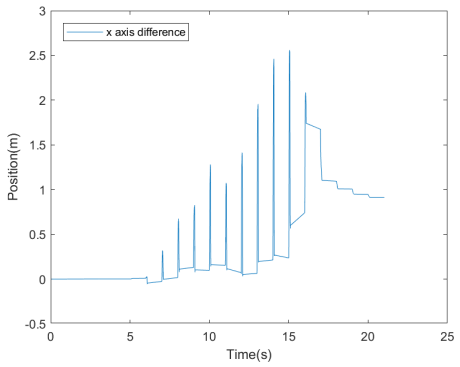

(a) x axis comparison

(b) y axis comparison

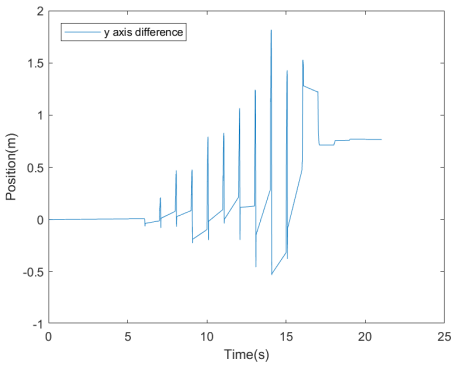
(c) z axis comparison

(d) Executed trajectory comparison in 2D

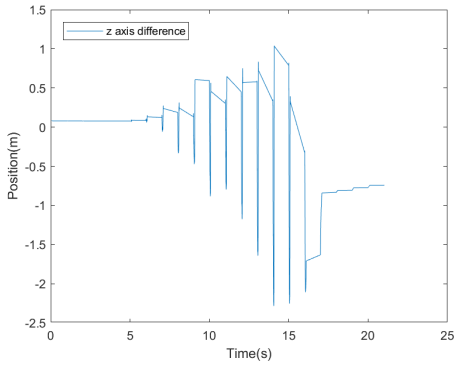
Figure A.1: Comparison between ideal odometry and filtered odometry from 1 imu and 1 odometry




(a) x axis difference



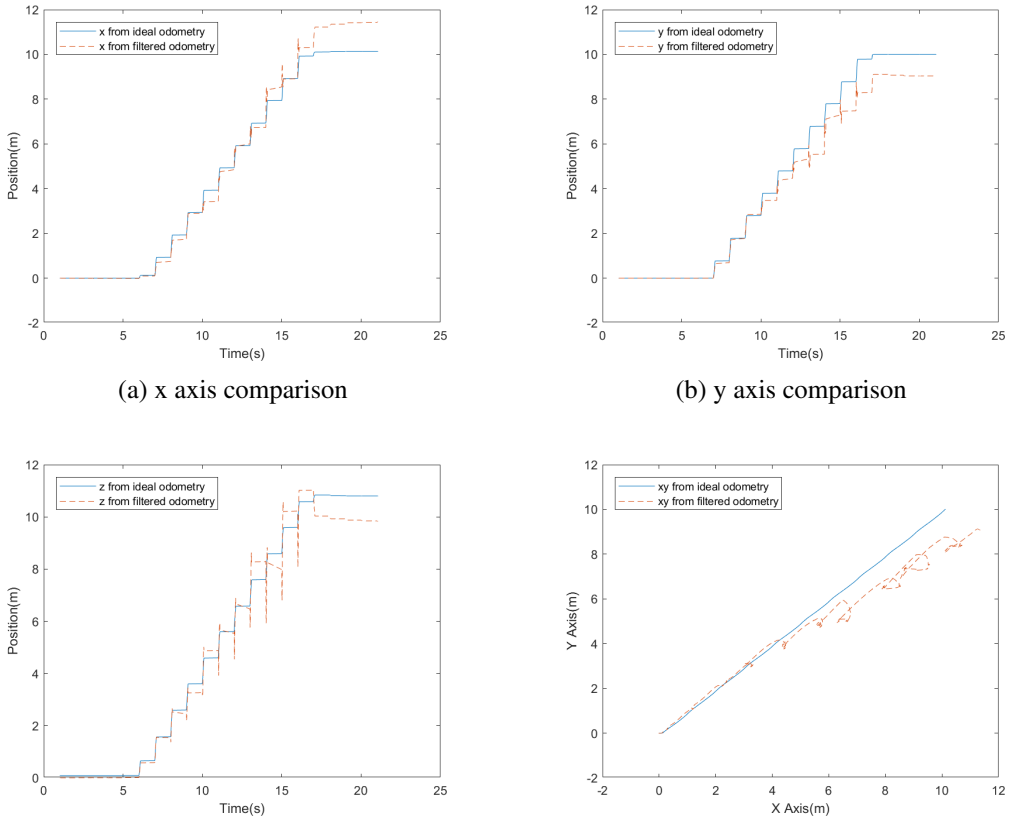

(b) y axis difference




(c) z axis difference

Figure A.2: difference between ideal odometry and the filtered result from 1 imu and 1 odometry

### A.4.2 2 identical imu 1 odometry



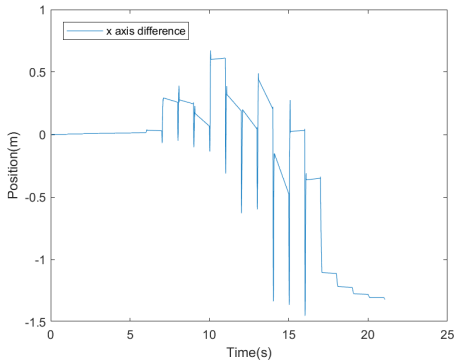

(a) x axis comparison

(b) y axis comparison

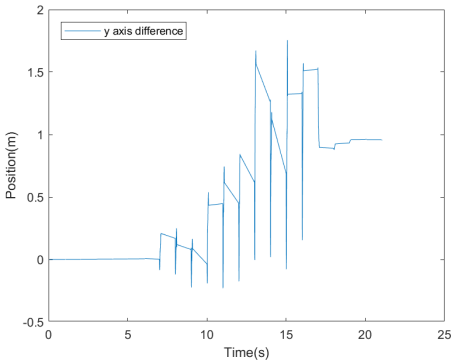
(c) z axis comparison

(d) Executed trajectory comparison in 2D

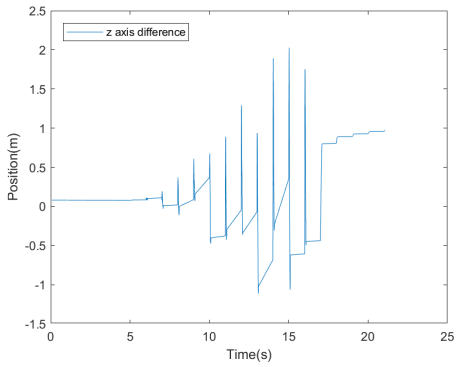
Figure A.3: Comparison between ideal odometry and filtered odometry from 2 identical imu and 1 odometry




(a) x axis difference



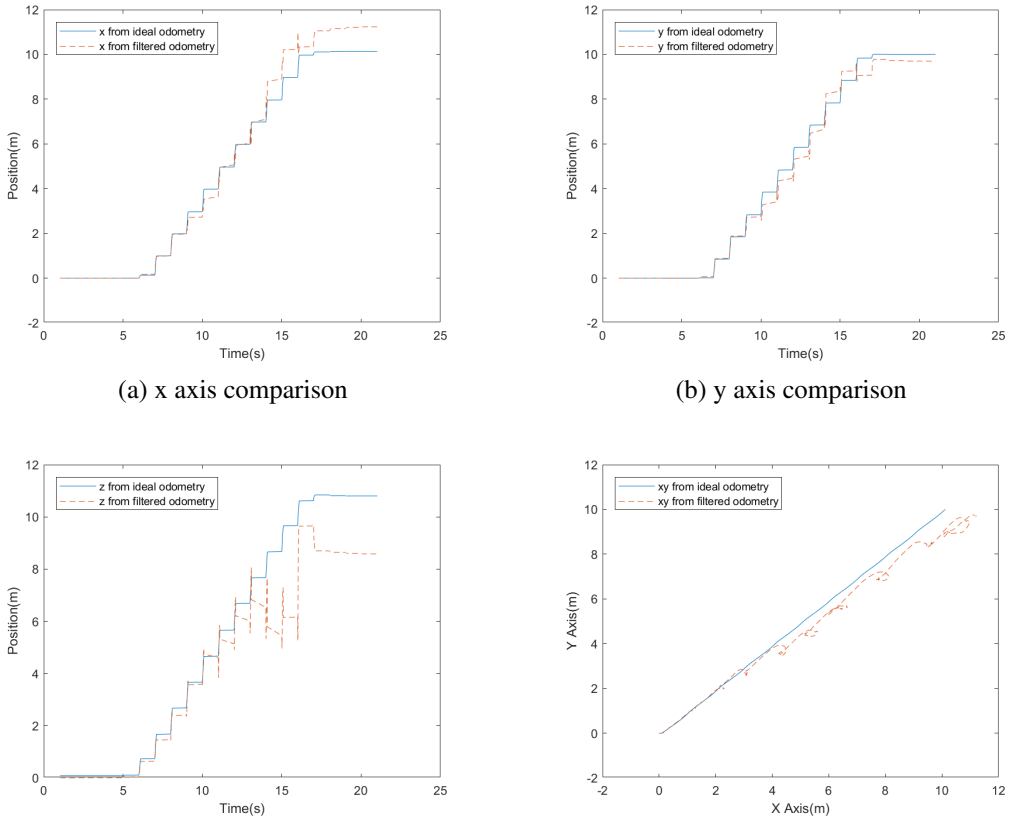

(b) y axis difference




(c) z axis difference

Figure A.4: difference between ideal odometry and the filtered result from 2 identical imu and 1 odometry

### A.4.3 1 imu 2 identical odometry



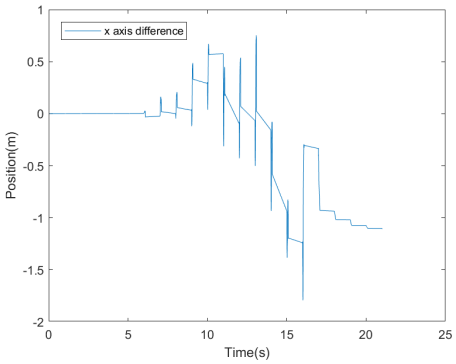

(a) x axis comparison

(b) y axis comparison

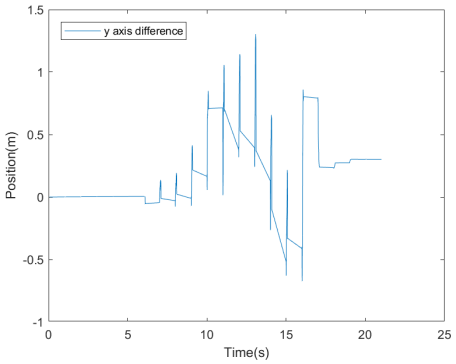
(c) z axis comparison

(d) Executed trajectory comparison in 2D

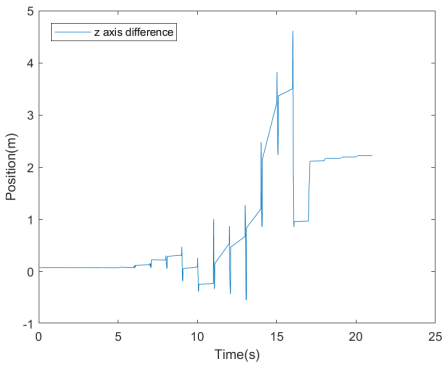
Figure A.5: Comparison between ideal odometry and filtered odometry from 1 imu and 2 identical odometry




(a) x axis difference



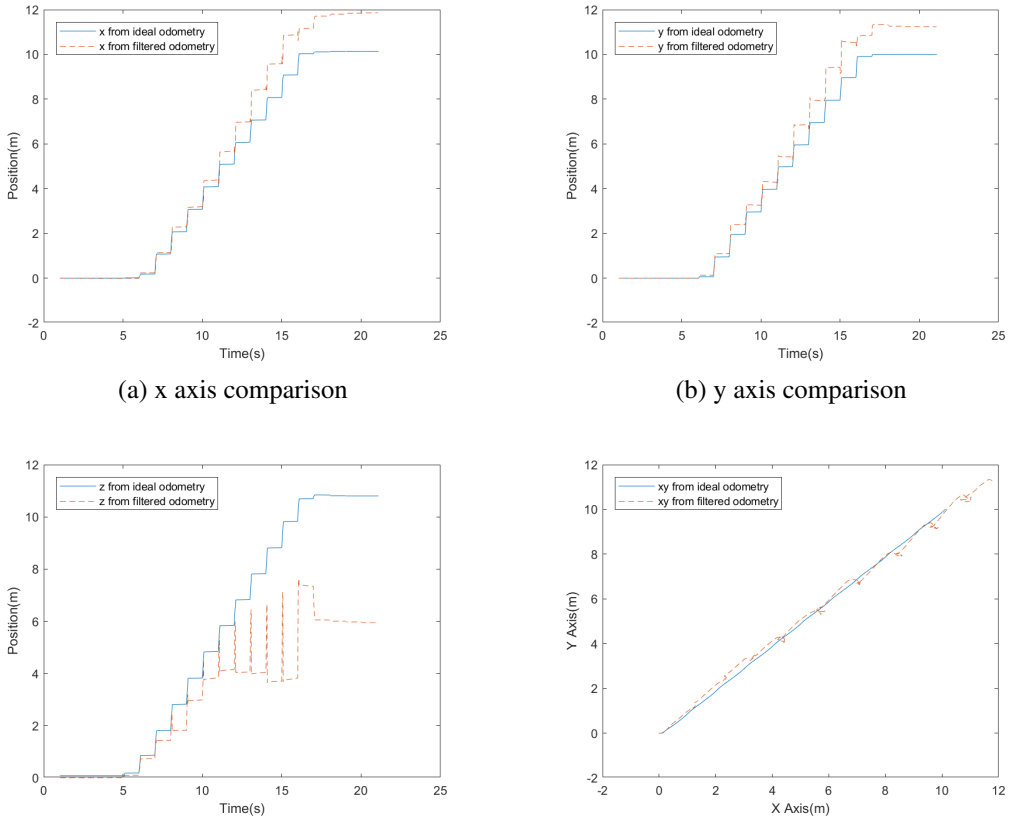

(b) y axis difference




(c) z axis difference

Figure A.6: difference between ideal odometry and the filtered result from 1 imu and 2 identical odometry

### A.4.4 2 identical imu 2 identical odometry



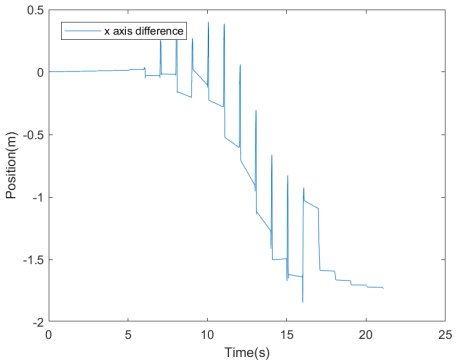

(a) x axis comparison

(b) y axis comparison

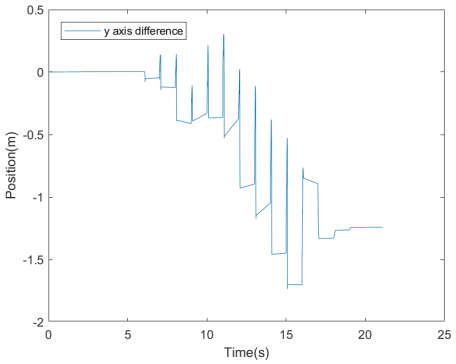
(c) z axis comparison

(d) Executed trajectory comparison in 2D

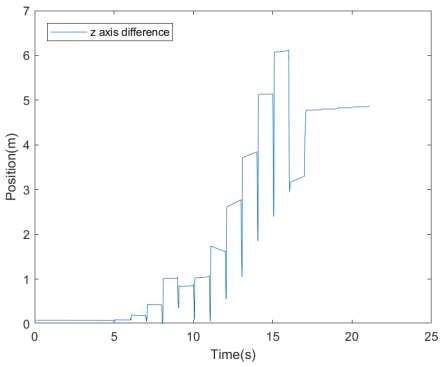
Figure A.7: Comparison between ideal odometry and filtered odometry from 2 identical imu and 2 identical odometry




(a) x axis difference



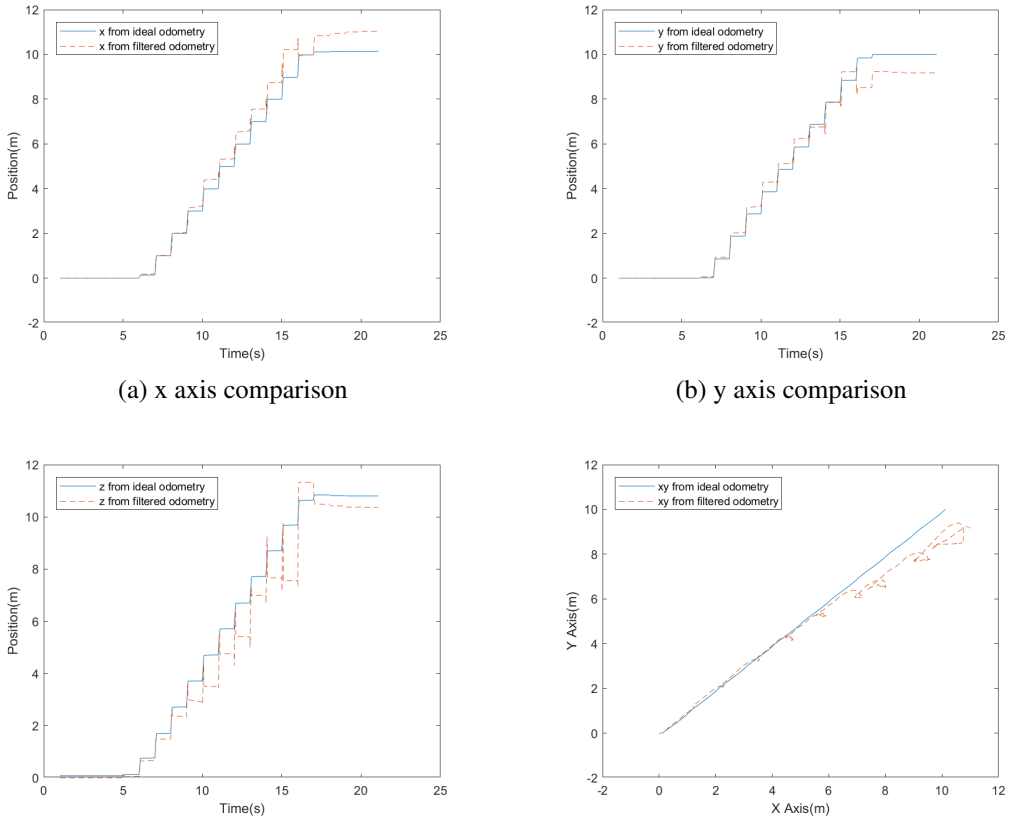

(b) y axis difference




(c) z axis difference

Figure A.8: difference between ideal odometry and the filtered result from 2 identical imu and 2 identical odometry

### A.4.5 1 imu 2 different odometry



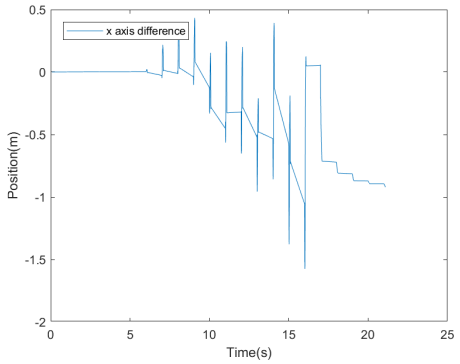

(a) x axis comparison

(b) y axis comparison

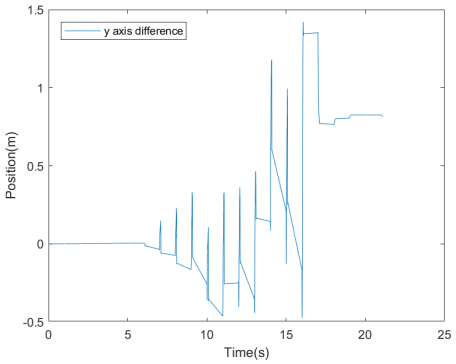
(c) z axis comparison

(d) Executed trajectory comparison in 2D

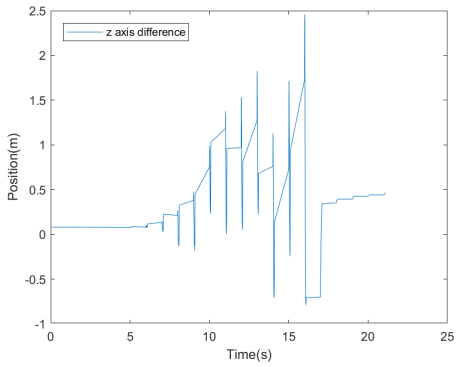
Figure A.9: Comparison between ideal odometry and filtered odometry from 1 imu and 2 different different odometry




(a) x axis difference



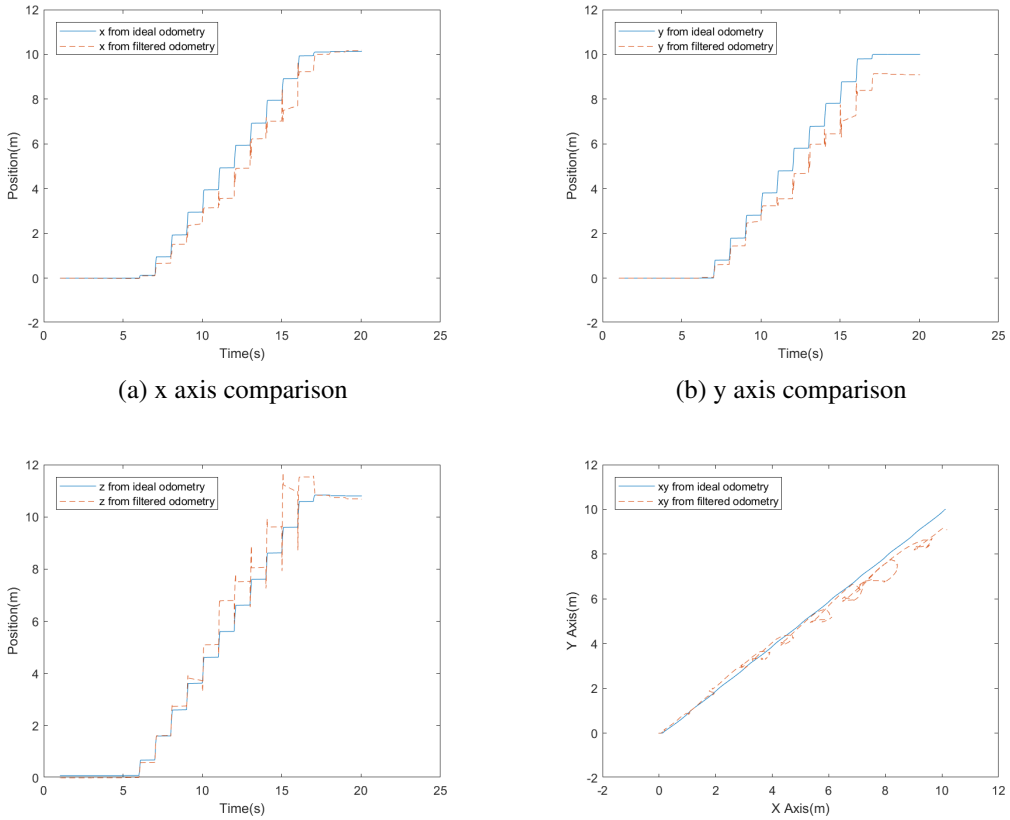

(b) y axis difference




(c) z axis difference

Figure A.10: difference between ideal odometry and the filtered result from 1 imu and 2 different different odometry

### A.4.6 2 different imu 1 odometry



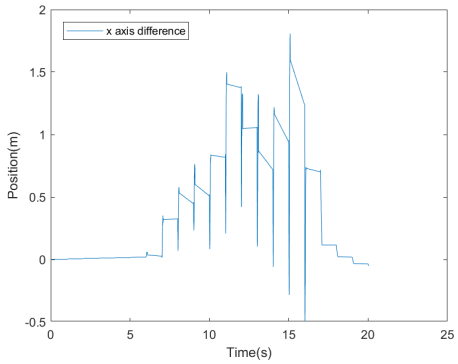

(a) x axis comparison

(b) y axis comparison

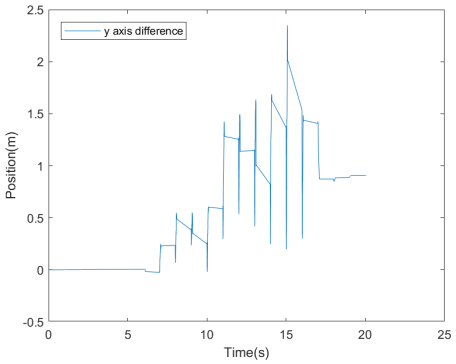
(c) z axis comparison

(d) Executed trajectory comparison in 2D

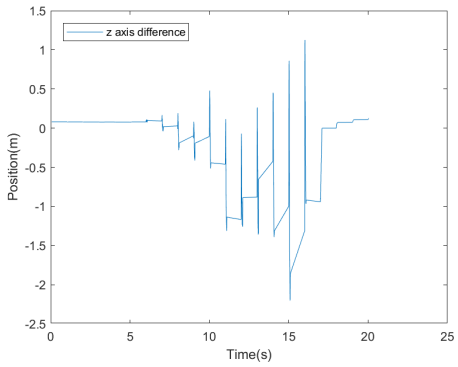
Figure A.11: Comparison between ideal odometry and filtered odometry from 2 different imu and 1 odometry




(a) x axis difference



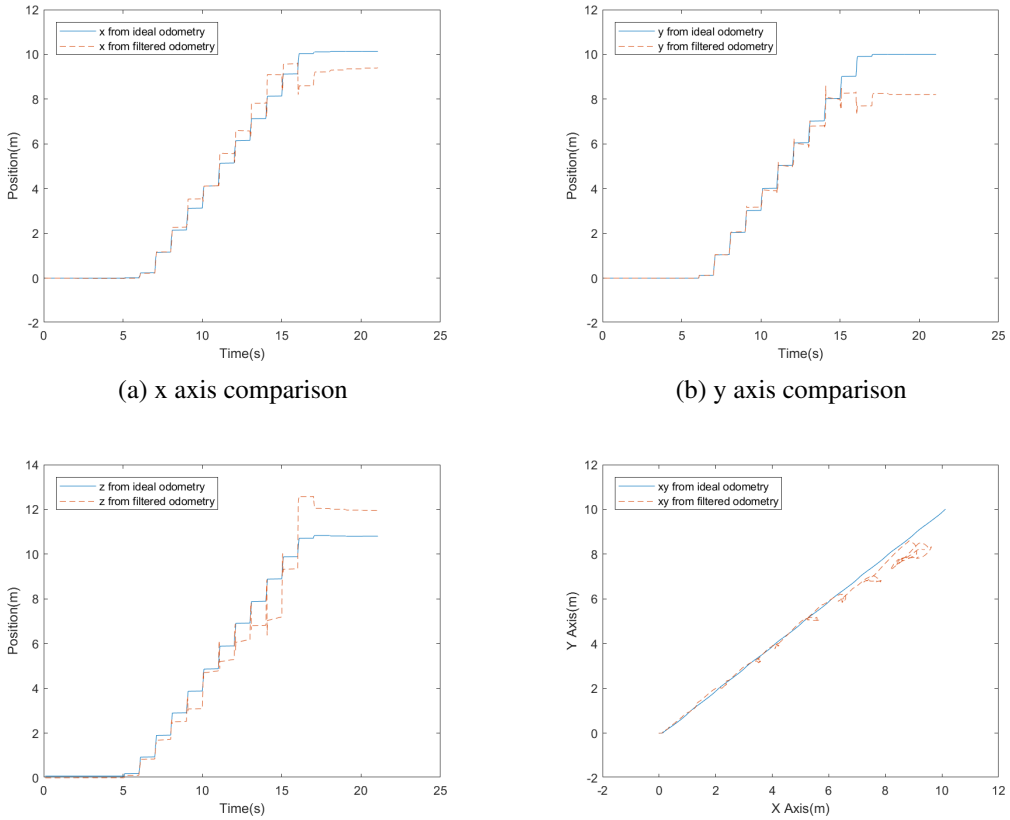

(b) y axis difference




(c) z axis difference

Figure A.12: difference between ideal odometry and the filtered result from 2 different imu and 1 odometry

### A.4.7 2 different imu 2 different odometry



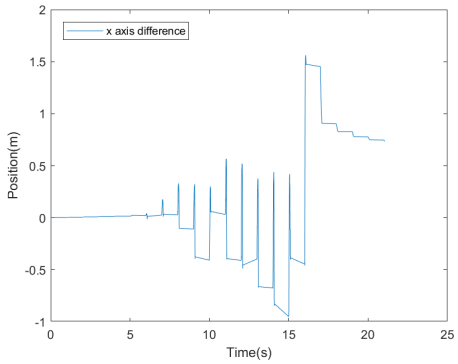

(a) x axis comparison

(b) y axis comparison

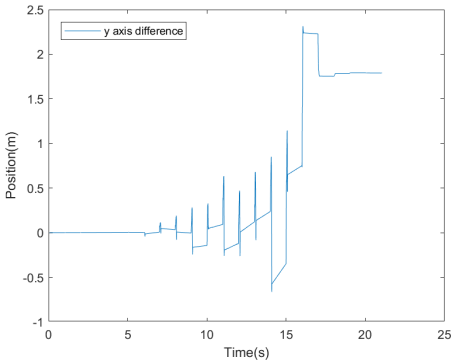
(c) z axis comparison

(d) Executed trajectory comparison in 2D

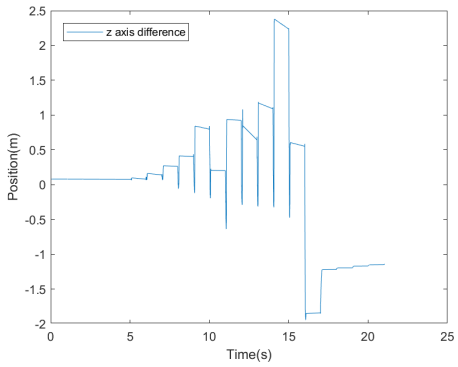
Figure A.13: Comparison between ideal odometry and filtered odometry from 2 different imu and 2 different odometry




(a) x axis difference



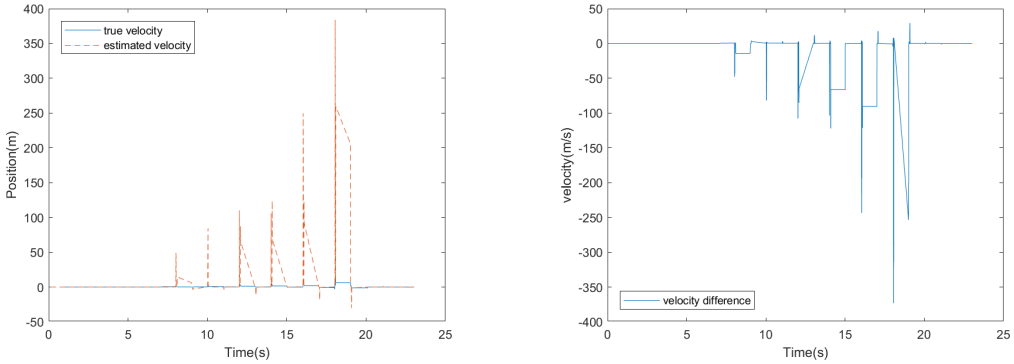

(b) y axis difference




(c) z axis difference

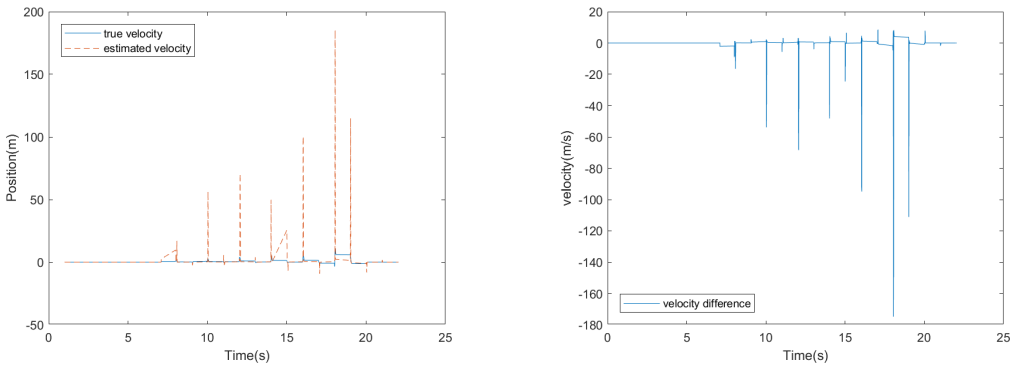
Figure A.14: difference between ideal odometry and the filtered result from 2 different imu and 2 different odometry

## A.5 Velocity estimation with different duration step



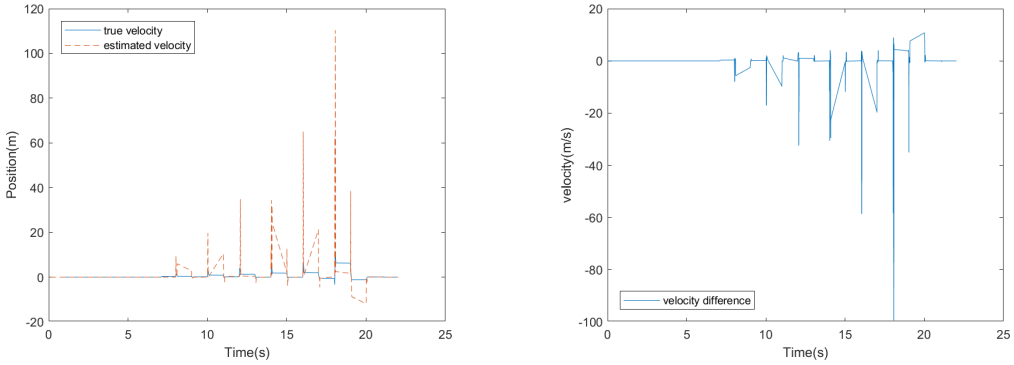

(a) Velocity comparison (b) Velocity difference

Figure A.15: Comparison between true velocity and the estimated velocity in y axis with step duration of 0.001 s



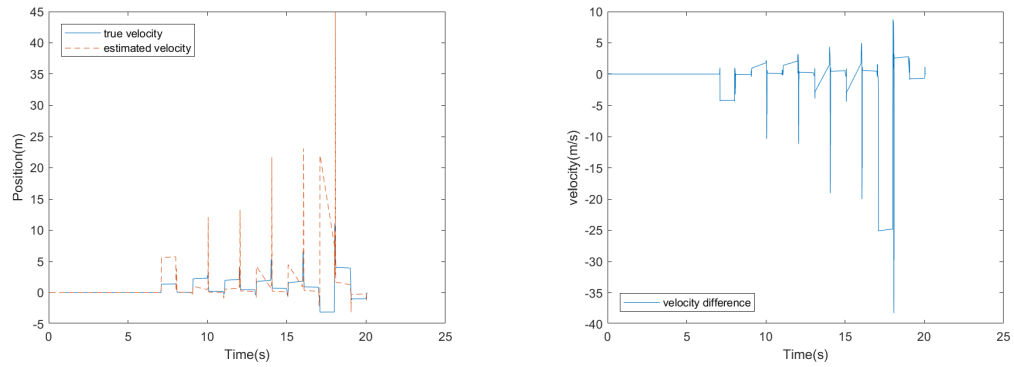

(a) Velocity comparison (b) Velocity difference

Figure A.16: Comparison between true velocity and the estimated velocity in y axis with step duration of 0.002 s



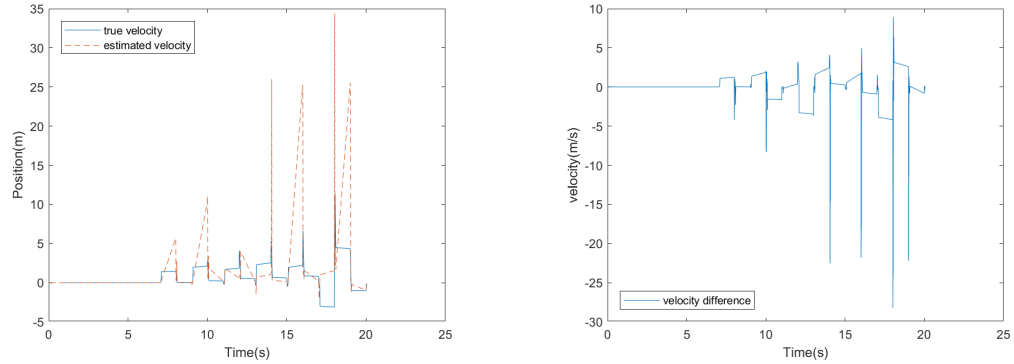

(a) Velocity comparison (b) Velocity difference

Figure A.17: Comparison between true velocity and the estimated velocity in y axis with step duration of 0.004 s



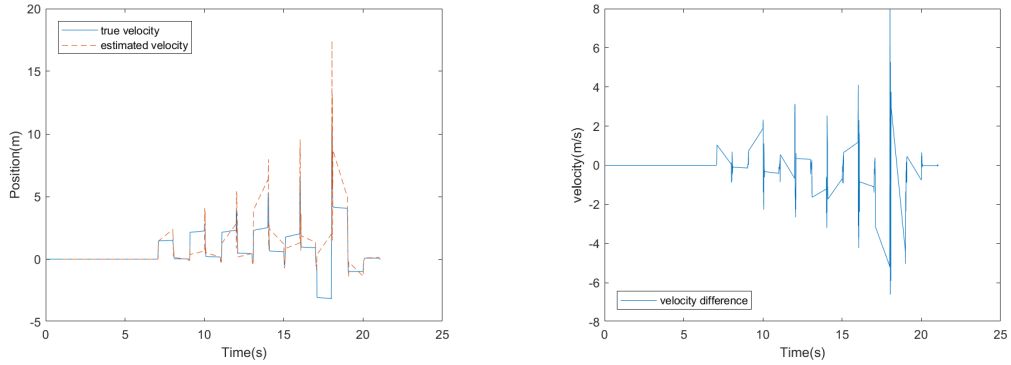

(a) Velocity comparison (b) Velocity difference

Figure A.18: Comparison between true velocity and the estimated velocity in y axis with step duration of 0.008 s



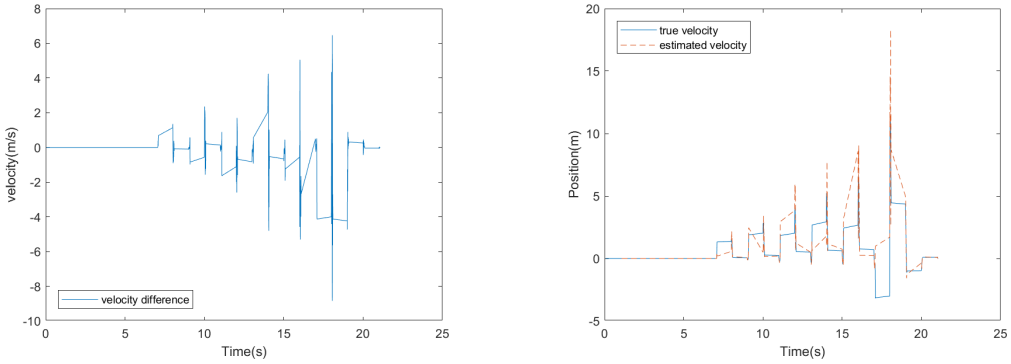

(a) Velocity comparison (b) Velocity difference

Figure A.19: Comparison between true velocity and the estimated velocity in y axis with step duration of 0.016 s



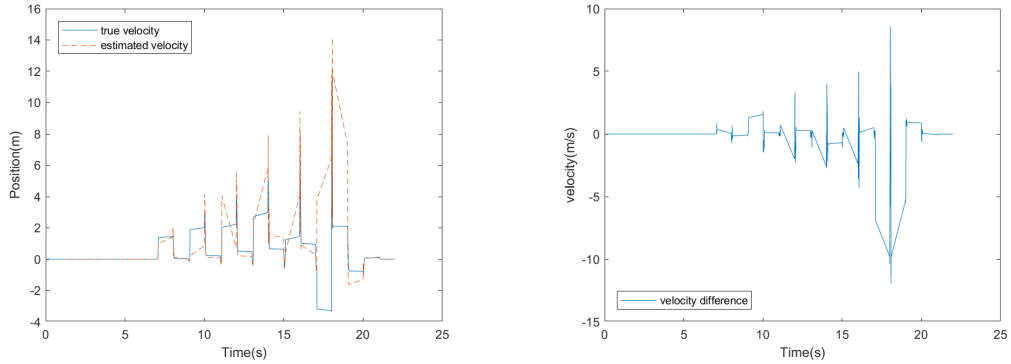

(a) Velocity comparison (b) Velocity difference

Figure A.20: Comparison between true velocity and the estimated velocity in y axis with step duration of 0.032 s



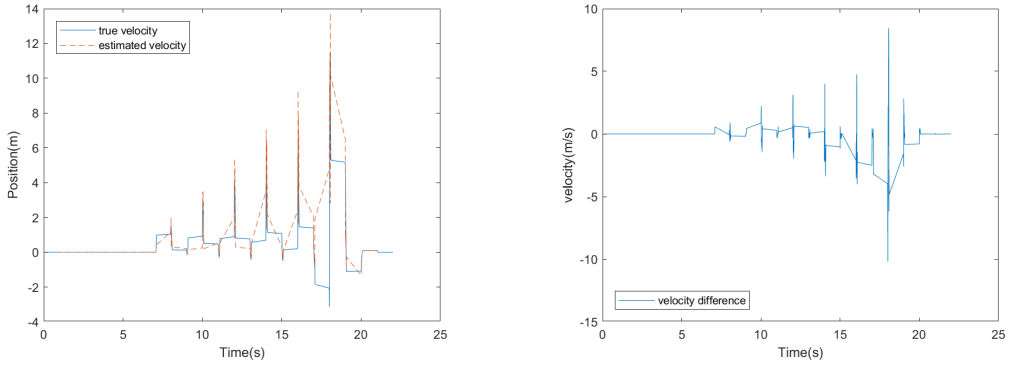

(a) Velocity comparison (b) Velocity difference

Figure A.21: Comparison between true velocity and the estimated velocity in y axis with step duration of 0.033 s



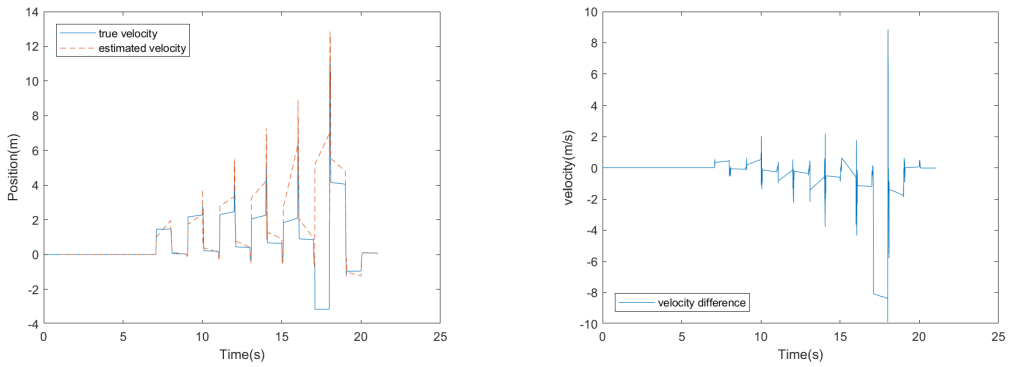

(a) Velocity comparison (b) Velocity difference

Figure A.22: Comparison between true velocity and the estimated velocity in y axis with step duration of 0.035 s



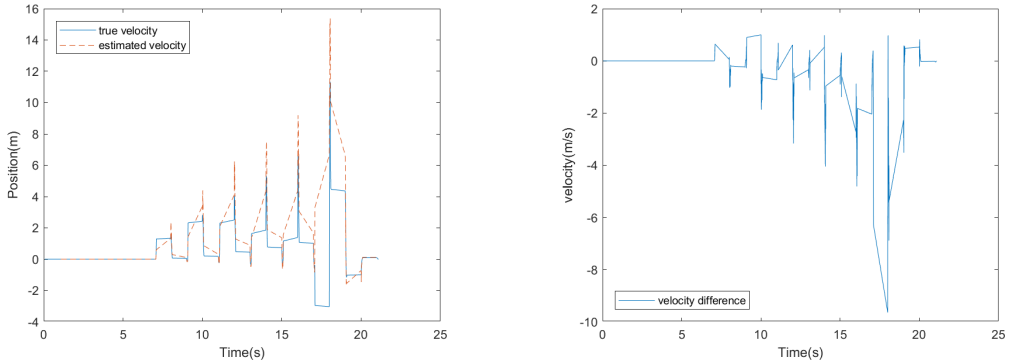

(a) Velocity comparison (b) Velocity difference

Figure A.23: Comparison between true velocity and the estimated velocity in y axis with step duration of 0.037 s



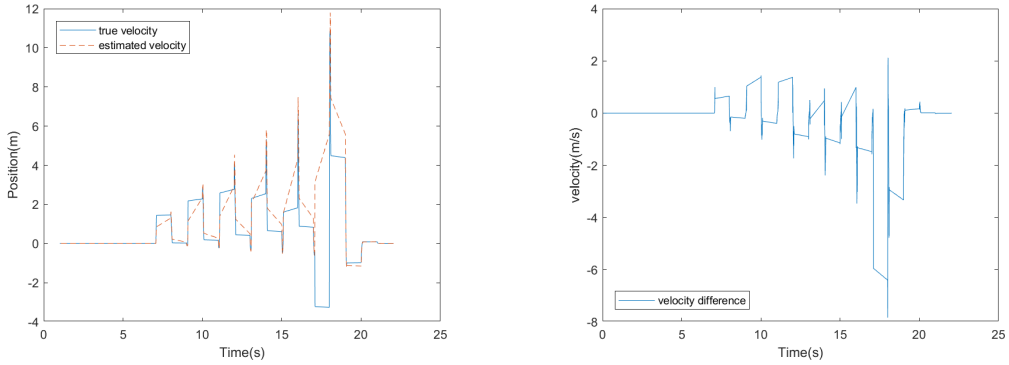

(a) Velocity comparison (b) Velocity difference

Figure A.24: Comparison between true velocity and the estimated velocity in y axis with step duration of 0.04 s



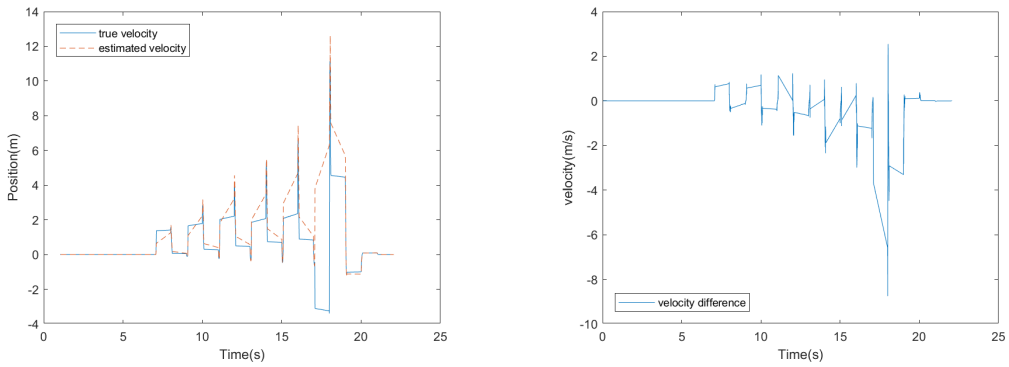

(a) Velocity comparison (b) Velocity difference

Figure A.25: Comparison between true velocity and the estimated velocity in y axis with step duration of 0.064 s



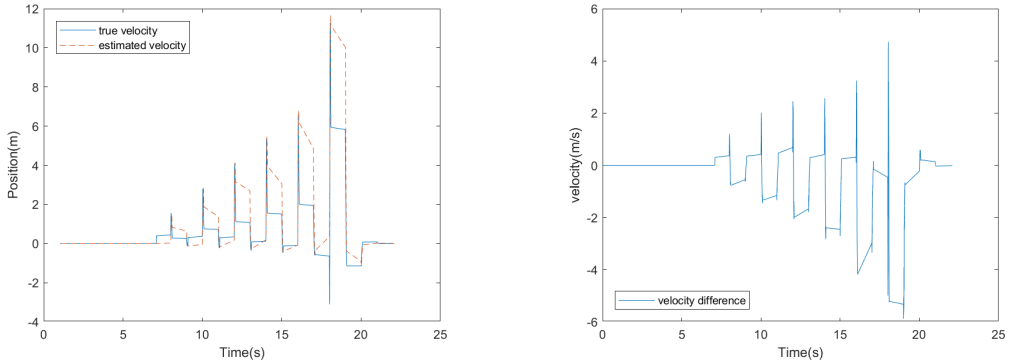

(a) Velocity comparison (b) Velocity difference

Figure A.26: Comparison between true velocity and the estimated velocity in y axis with step duration of 0.128 s



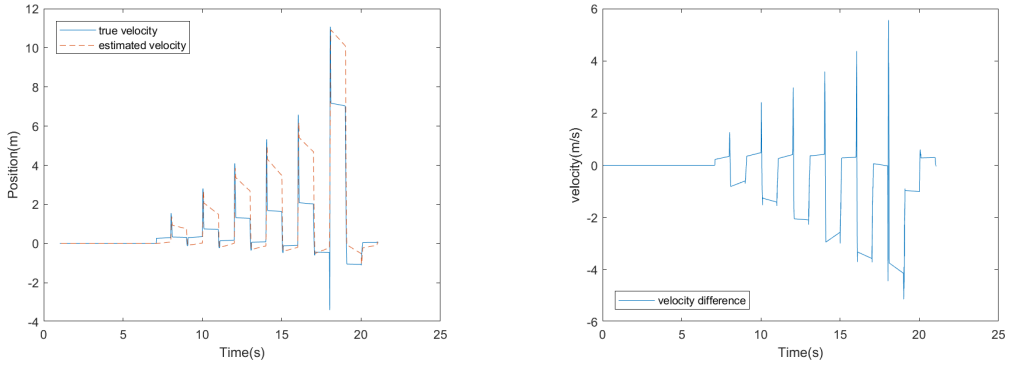

(a) Velocity comparison (b) Velocity difference

Figure A.27: Comparison between true velocity and the estimated velocity in y axis with step duration of 0.256 s



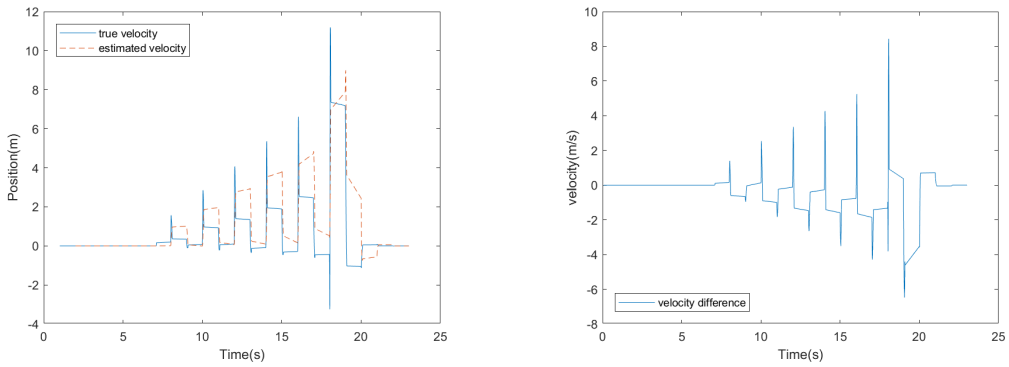

(a) Velocity comparison (b) Velocity difference

Figure A.28: Comparison between true velocity and the estimated velocity in y axis with step duration of 0.384 s



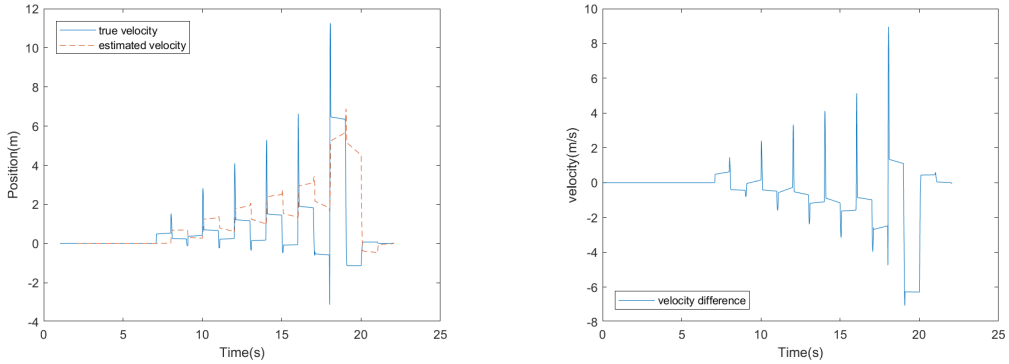

(a) Velocity comparison (b) Velocity difference

Figure A.29: Comparison between true velocity and the estimated velocity in y axis with step duration of 0.512 s



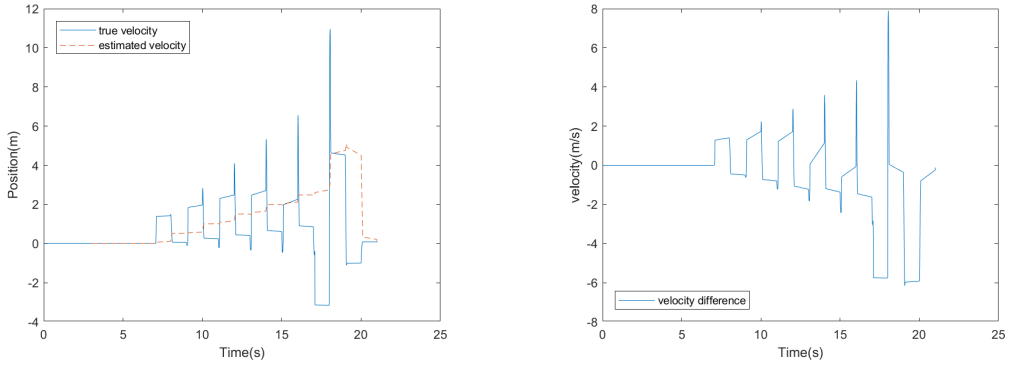

(a) Velocity comparison (b) Velocity difference

Figure A.30: Comparison between true velocity and the estimated velocity in y axis with step duration of 1.024 s



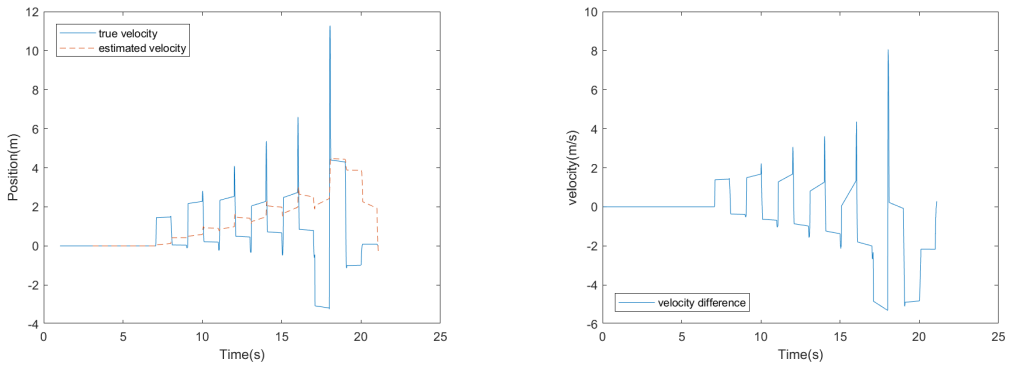

(a) Velocity comparison (b) Velocity difference

Figure A.31: Comparison between true velocity and the estimated velocity in y axis with step duration of 1.536 s



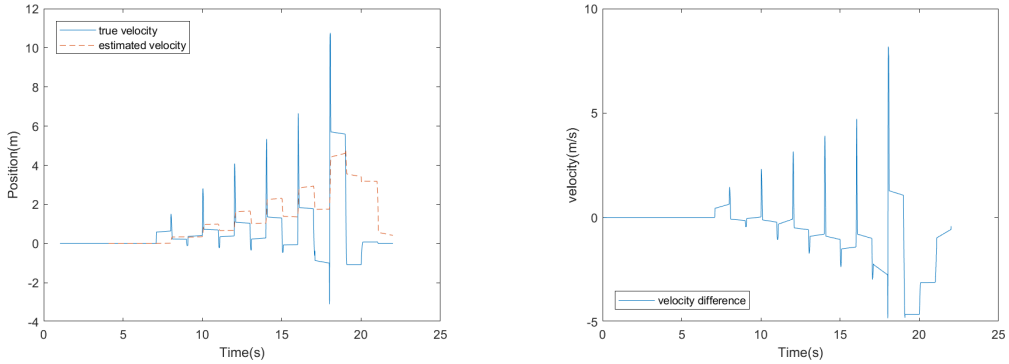

(a) Velocity comparison (b) Velocity difference

Figure A.32: Comparison between true velocity and the estimated velocity in y axis with step duration of 2.048 s




(a) Velocity comparison (b) Velocity difference

Figure A.33: Comparison between true velocity and the estimated velocity in y axis with step duration of 2.560 s




(a) Velocity comparison (b) Velocity difference

Figure A.34: Comparison between true velocity and the estimated velocity in y axis with step duration of 0. s