August 2015

3D Obstacle Avoidance for Unmanned Autonomous System (UAS)

Lin Zhao
University of Nevada, Las Vegas, crazymumu0804@gmail.com

3D OBSTACLE AVOIDANCE FOR UNMANNED AUTONOMOUS SYSTEM (UAS)

By

Lin Zhao

Bachelor of Science in Mechanical & Electronic Engineering

Zhejiang University City College, China

2013

A thesis submitted in partial fulfillment

of the requirements for the

Master of Science in Engineering – Mechanical Engineering

Department of Mechanical Engineering

Howard R. Hughes College of Engineering

The Graduate of College

University of Nevada, Las Vegas

August 2015

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

July 24, 2015

This thesis prepared by

Lin Zhao

entitled

3D Obstacle Avoidance for Unmanned Autonomous System (UAS)

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Engineering – Mechanical Engineering
Department of Mechanical Engineering

Woosoon Yim, Ph.D.
Examination Committee Chair

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Interim Dean

Mohamed Trabia, Ph.D.
Examination Committee Member

Kwang J. Kim, Ph.D.
Examination Committee Member

Sahjendra Singh, Ph.D.
Graduate College Faculty Representative

ABSTRACT

The goal of this thesis is to design a real-time, three-dimensional algorithm, named as the vector mesh (VM) algorithm, for unmanned aerial vehicles (UAV) to generate collision-free motion in indoor or outdoor environments with unknown obstacles. This promising technology can be utilized in both military and commercial applications. The VM approach employs three data reduction phases to compute optimal navigation directions while on-board scanning range sensor continuously updates depth data. In order to develop the VM, vector filed histogram (VFH) which applied in 2D space was first simulated in Matlab. Then a 2D autonomous navigation was implemented on a developed Vision-based Ground Vehicle (VGV) and the entire system was controlled by a modified VFH method which was computing in the Robot Operating System (ROS). Also, the VM algorithm was simulated in ROS and integrated into Gazebo simulator which is an effective graphic based robot simulator in complex indoor and outdoor environment. In this study, it has been shown that the proposed VM can be an effective 3D obstacle avoidance algorithm for typical small-UAVs if 3D information is continuously provided.

ACKNOWLEDGMENTS

It's been two years since the first day I came to University of Nevada, Las Vegas, the place I started to research and chase personal objective. Firstly, I would like to thank my advisor, Dr. Woosoon Yim for the continuous support and patience to my master study. His guidance helped me in all the time of research and writing of this thesis. He is a nice advisor, gracious professor and dedicated researcher. I could not have imagined having a better advisor and mentor for study.

I would like to thank the NSF for financially supporting my research.

I would like to thank my thesis committee: Dr. Mohamed Trabia, Dr. Kwang J. Kim and Dr. Singh, Sahjendra, for their insightful comments and encouragement, but also support me to solve problems in their teaching. I would like to thanks all the professors and staffs in UNLV helped me to research and classes.

I would like to my friends and workfellows that are Jameson Lee and Zachary Cook to work with me and give me suggestions to finish the projects. Also, it's very kind of Zack to review my thesis.

I would like to thank my friends Qi Shen, Chao Chen and Wenlan Wu to give me supports in both study and life; also I am so grateful with your advice in writing this thesis.

Lastly, I would like to thank my parents to raise me up and support me to study. And thanks to my girlfriend's concern and encourage.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Unmanned Autonomous System

In the past decades, main reasons behind the exponential increase rate in the operation and development of unmanned autonomous systems (UAS) is the growing necessity of replacing humans work in dangerous missions or unreachable locations. More than twenty countries have invested substantial resources towards the development and the manufacturing UAS for a wide range of applications, both in the military and civilian domains [1]. Additionally, future UAS design, such as Unmanned Ground Vehicle (UGV) and Unmanned Aerial Vehicle UAV without human controller, becomes more and more intelligent and robust according to requirement for upper level autonomy and increased difficulty tasks.

A UAV is often classified as one of UAS and also known as drone which is a vehicle or an aircraft without a human pilot aboard. Two types of system control this aircraft, whether autonomous system which means small size onboard computer or remote controller such as ground control station and personal RC controller. Different propulsion methods of UAV lead to discriminative configurations, such like fixed-wing aircraft and rotary-wing aircraft.

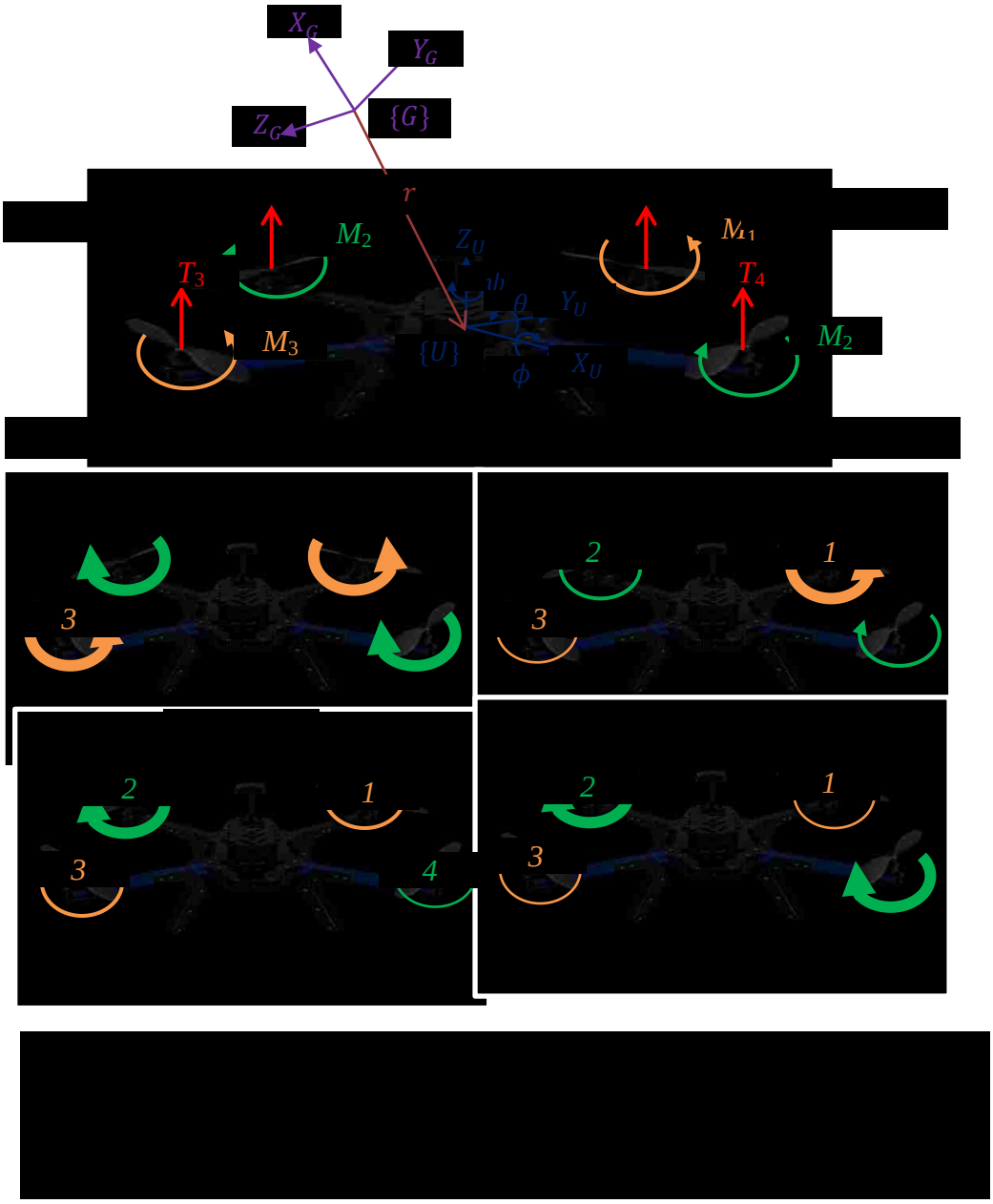
MQ-9 Reaper Hunter UAV, shown in Fig. 1.1(a), is the typical military example which first flight was in 2007 and also executed a combat mission in Afghanistan [2]. Beyond the military applications of UAVs, which have been used in plenty of civilian area containing aerial surveillance, commercial Photography, urgency rescue and high-class toy. Titan Aerospace bought by Google designed a solar-powered UAV, shown in Fig. 1.1(b), travels up to 20 kilometers high with capacity of having satellite typical functions, one

such example is weather monitoring. The future work of this fixed-wing UAV is bringing Internet connectivity to distant region [3]. The A160 Hummingbird helicopter UAV, shown in Fig. 1.1(c), built by Boeing has ability to take multiple missions just as target acquisition and surveillance with files at 260 km/h at a maximum altitude of 9150 m [4]. Phantom3, shown in Fig. 1.1(d), is a commercial entertainment drone developed by Chinese company DJI with the controller a maximum range of 2000 meters and visual position system [5]. The superiority of UAV such as reducing the exposure risk of pilot and enjoying magnificent landscape is becoming more and more apparent.

1.1.1 Overview of Multicopter

A multicopter is classified as a rotary-wing aircraft, also is most popular one of vertical take-off and landing (VTOL) aircraft that can hover, take off, and land vertically [6]. Less kinetic energy cost in a flight condition because of individual rotor diameter smaller than equivalent rotary-wing aircraft. Small-scale multicopter haves frame that enclose the rotors, permitting flights through more challenging environments, with lower risk of damaging the vehicle or its surroundings [7]. The most typical multicopter is the quadrotor which has 4 propellers and the hexarotor with 6 propellers is also a popular variety of multicopter because it has more thrust than equivalent quadrotor.

Although quadrotor is an appealing VTOL aircraft with relative simple structure and light weight, it is a typical under-actuated, non-linear coupled system. This is due to quadrotor system have four inputs to control six degrees of freedom (DOF), which including translations and rotation along three principal axes. Four basic movements chose to be controllable variables. Quadrotor lift (or land) with increasing (or decreasing) throttle



during all the motors rotation shown in Fig. 1.2 (b). By increasing angular velocity of motor 1 and decreasing angular velocity of motor 3, shown in Fig. 1.2 (c), roll action is accomplished which will lead quadrotor move to rear direction. By increasing angular velocity of motor 2 and decreasing angular velocity of motor 4, shown in Fig. 1.2 (d), pitch action is accomplished which will lead quadrotor move to right direction. By

increasing angular velocity of motor 1 and motor 3 while decreasing angular velocity of motor 3 and motor 4, shown in Fig. 1.2 (e), yaw action is accomplished which will arm quadrotor rotate with z-axis along clockwise.

1.2 Methods of Vision-based Obstacle Avoidance

Unlike path planning in known environment, strong assumptions on knowledge of obstacles situation need to be completed in order to generate collision-free path. Vision sensing technique as the most effective and powerful method is now largely applied in the autonomous navigation task or obstacle avoidance mission [8].

EPIX stereo camera [9] has two 2048 ×1088 pixel lines global shutter and captures 10 bits images at 340 fps shown in Fig. 1.3 (a). Hokuyo UTM-30LX scanning laser rangefinder, shown in Fig. 1.3 (b), as the widely used laser sensor has 30 m and 270 degree scanning

range with high frequency 40 Hz which can implement in normal robotics research [10].

A time-of –flight (TOF) camera is a high frame-rate and accuracy 3D image vision sensor which works by illuminates the scene with a modulated light source and observes the reflected light [11], such as SR4000 TOF Camera [12] shown in Fig. 1.3(c) and Microsoft Kinect sneosr. 2D sensor or 3D camera suit for relevant dimensional algorithm for UGV, UAV or humanoid robot, nevertheless some cases employ several cameras for exhaustive surrounding information.

A Markov Random Field approach to distinguish obstacle region and free-space area based on single monocular camera producing obstacle classifications [13], especial useful to avoid assorted varieties obstacles including tree and fences for low-power outdoor robotics applications. Similarly, monocular camera used for escaping from obstacles but utilizing different strategy that is Fast Terrain Mapping, which extracted features to update a sequential extended Kalman filter [14]. Acknowledge of omnidirectional obstacle perception is the primary problem for a clutter and restricted environment. To solve that, autonomous system set up with multiple sensors containing a 3D laser scanner, two stereo camera and ultrasonic sensors proceeds global mission planner and local trajectory planning in multi-resolution local grid maps [15]. A state-of art registration algorithm to detect obstacle location by reconstructing 3D point clouds, which achieved with a rotating 2D laser scanner [16]. Simultaneous localization and mapping (SLAM) [17] is the approach of constructing a map of unknown environment from available sensor data and continuously estimating robot position and orientation.

1.3 Contribution

The contribution of this study is the development of 3D obstacle avoidance, named Vector Mesh (VM), an algorithm for real-time navigation of quadrotor in unknown indoor environment. In the simulation of presented algorithm, 3D Kinect sensor [18] is used for collecting range data stored momentarily for generation of obstacle perceptions which are larger than the single detection. Additionally, obstacles estimation and path planning are based on Vector Field Histogram (VFH) [19] algorithm extended from 2D to 3D.

1.4 Thesis Organization

This thesis is organized into five chapters as follows. Fundamental knowledge of Unmanned Aerial Vehicle (UAV) is introduced in Chapter 1 including current state-of-art applications of UAV. Relevant background theory of in sensors and algorithms of obstacle avoidance is described. In Chapter 2, a review of VFH algorithm is presented and drawbacks and comprehend advantages are illustrated using an example simulation. Chapter 3 presents a developed modified VFH algorithm for 2D environment. Besides, a vision-based ground vehicle system is demonstrated in order to perform autonomous navigation with developed 2D algorithm. In Chapter 4, the theory of proposed VM algorithm is presented and discussed. Also a computer simulation is presented to show the result. Moreover, a modified method is introduced to the algorithm to improve simulation results. Finally, conclusions and future work are presented in Chapter 5.

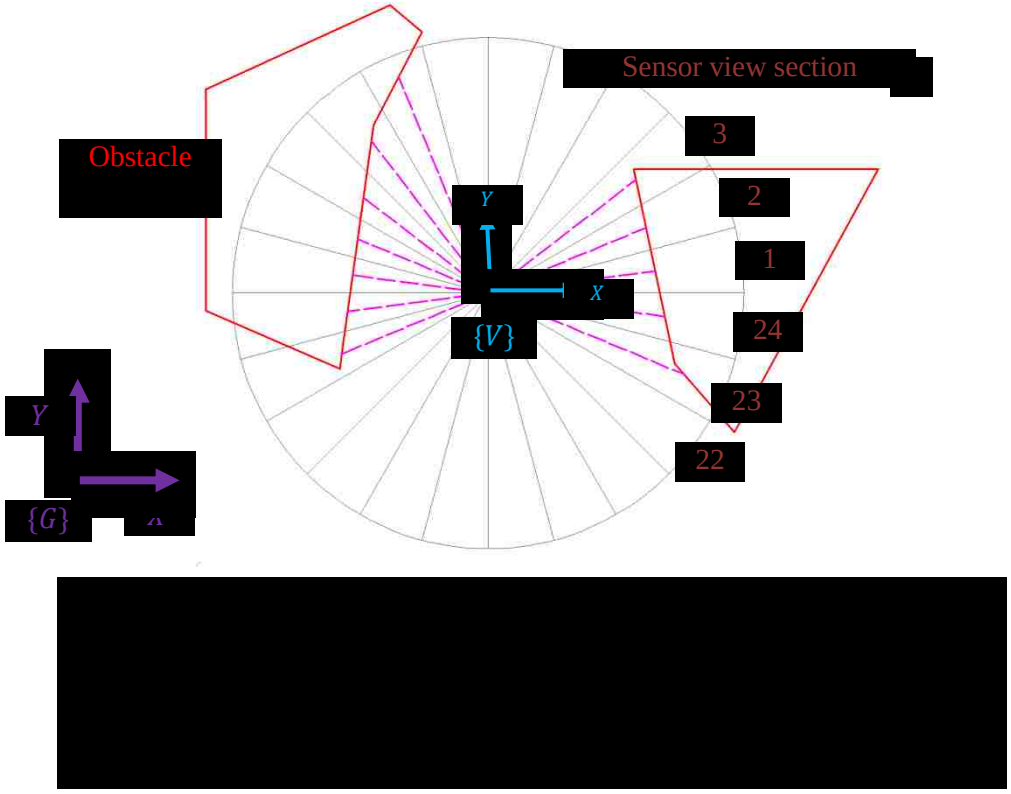
CHAPTER 2

2D OBSTACLE AVOIDANCE APPROACH

2.1 The Vector Field Histogram (VFH) Obstacle Avoidance Algorithm

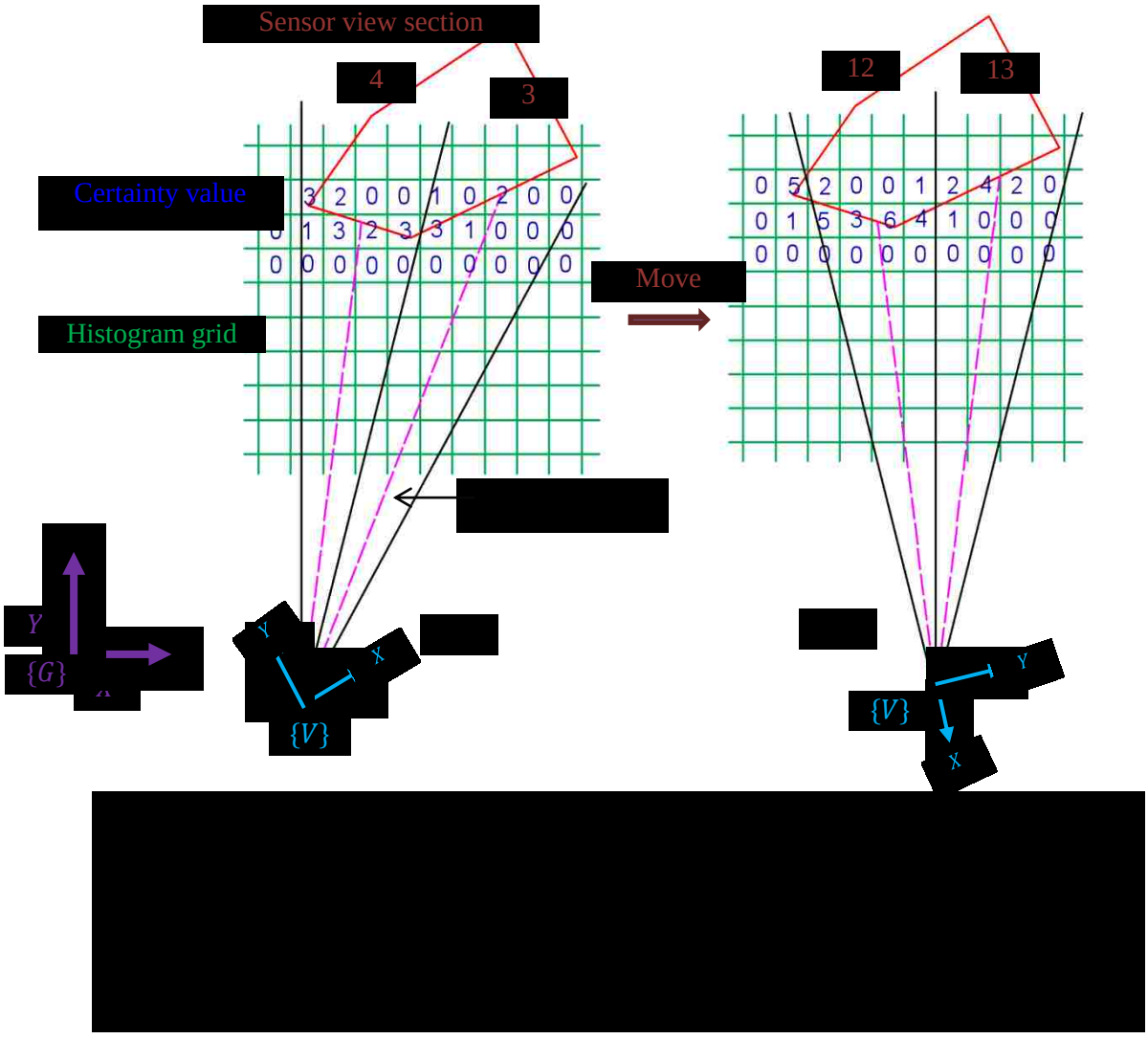
In 1991, Jojann Borenstein and Yoram Koren proposed a new real-time obstacle avoidance algorithm named as Vector Field Histogram, which detects unknown environment with range sensor and simultaneously generates collision-free path for ground mobile robots. This method covers three main components that are a two-dimensional Cartesian histogram grid, polar histogram sector and candidate valley. To begin with, on-board sensors such as ultrasonic sensor or laser rangerfinder are used for mapping obstacles into histogram grid. Moreover, one-dimensional polar histogram whose sector density denotes a probability of obstacle in that direction is made after reducing first step data. Moreover, an optimal solution is selected in each candidate valley such that every sector density less than an experimental threshold value. The VFH algorithm is suitable for both ground and aerial vehicles and has been developed and implemented in many different types of robotics platforms. Also, Enhanced VFH algorithm (VFH+) [20] and VFH* algorithm [24] had been proposed for improved tracking and automated avoidance performance. In this Section, a 2D VFH algorithm is presented and will be extended to the 3D case which can be useful for complex maneuvers of robotic aerial vehicles or UAV in unstructured environments.

2.1.1 VFH Algorithm



In this algorithm, the global world is described as a two-dimensional Cartesian histogram grid; a mesh size of cell in the grid is determined by a vehicle size as well as a computational power of the on-board computer. The VFH method shown in Fig. 2.1 uses 24 on-board range sensors, each with 15 degree view, to measure distance between vehicle and obstacles. As shown in Fig. 2.1, 24 sectors represent range sensors, red polygons are obstacles and pink dash lines express a mean distance between obstacles and the vehicle. For every range reading, the Certainty Value (CV), the larger CV represents higher possibility of obstacle in that location, is increased by 1 at that detected cell which is located along the acoustic axis as shown in Fig. 2.2 (a). This process is repeated while a vehicle moves so that a probabilistic distribution of obstacles is

calculated as shown in Fig. 2.2. The high value represents higher probability of collision to those directions as shown Fig. 3.2(b).



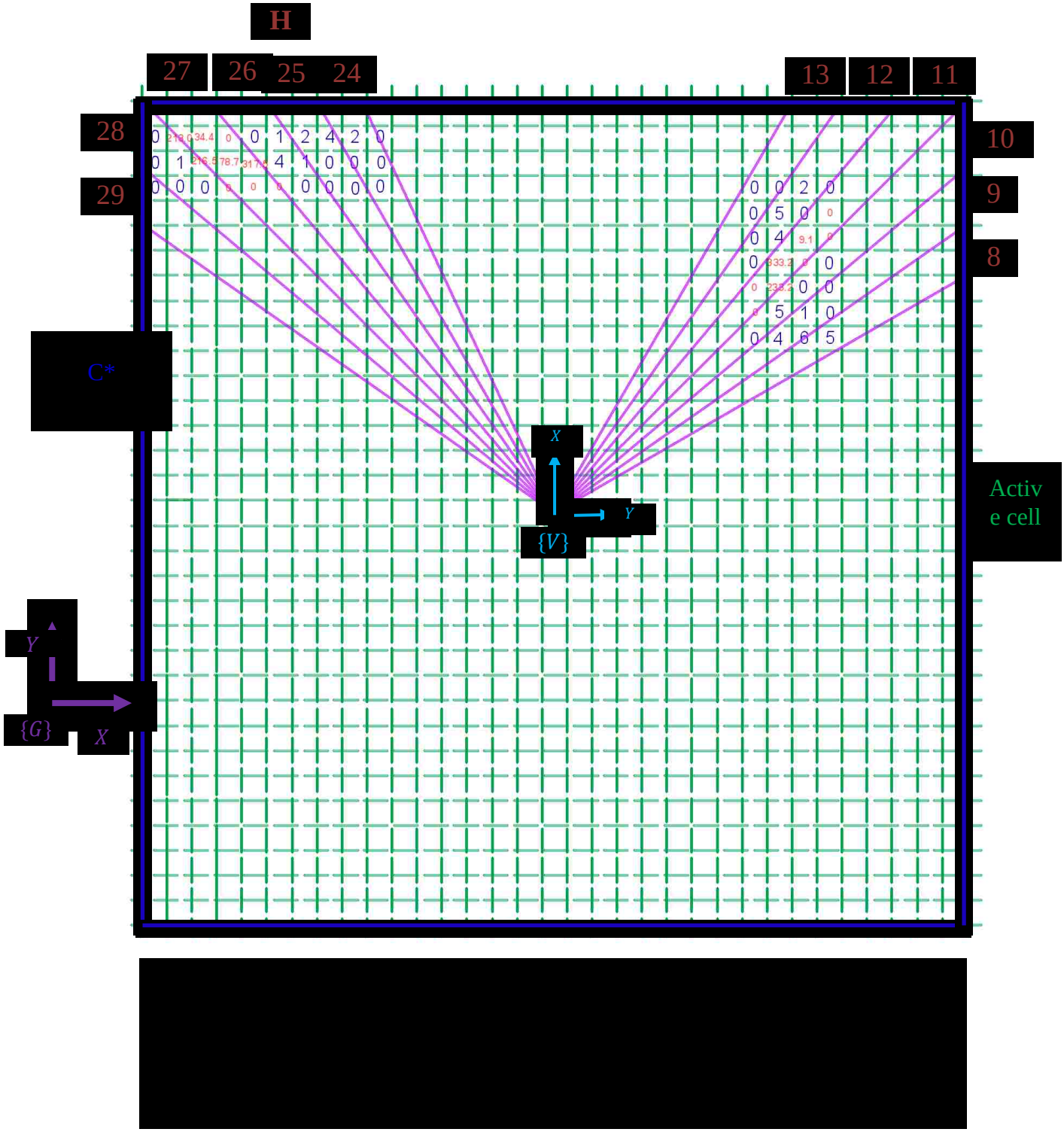


Fig. 2.3 shows an obstacle map transformed to polar coordinates. This transformation is needed for more efficient computation and it is conceptually easier to define the direction

of vehicle motion using one-dimensional polar histogram (denoted as H) as shown in Fig. 2.3. Also, the active window (C*) is defined with a vehicle at its center as shown in Fig. 2.3. The size of the active window can be adjusted for different applications; the one used here is 33 × 33 cells which correspond to range of sensor. It should be noted that this active window moves together with the vehicle and it overlay a new area. We also call the cells inside of the active window are called active cells. Angular resolution $\alpha = 5°$ used in Fig. 2.3 so that H contains 72 sectors. The active cells shown in Fig. 2.3 can be mapped to the polar map using:

$$\beta_{i,j} = \tan^{-1}\left(\frac{y_j - y_0}{x_i - x_0}\right) \tag{2.1}$$

$$k = INT\left(\frac{\beta_{i,j}}{\alpha}\right), k = 0,1,2,\ldots,71 \tag{2.2}$$

where (x_0, y_0) is an active cell location and (x_i, y_j) is a vehicle location in the active window. $\beta_{i,j}$ is an angle from x-axis of vehicle frame $\{V\}$, k is the sector which corresponds to the active cell of interest.

After converting active cells into a one-dimensional polar histogram, the magnitude of active cell $m_{i,j}$ as shown in Fig. 2.3 becomes:

$$m_{i,j} = (c^*_{i,j})^2\left(a - bd_{i,j}\right) \tag{2.3}$$

$$a - bd_{max} = 0 \tag{2.4}$$

where $c^*_{i,j}$ is the certainty value of active cell (i, j), the square here expresses to reduce noise which caused by single occurrence of sensor detection. $d_{i,j}$ is the distance between active cell and vehicle. a, b are positive constants which are obtained from Eq. 2.4, d_{max} is a half of diagonal of active window, if a is selected as an arbitrary integer then b is determined.

The polar obstacle density h_k which represents probability of meeting the obstacle in the direction of sector k becomes:

$$h_k = \sum m_{i,j}, if\ \beta_{i,j} = k \qquad (2.5)$$

The distribution of polar obstacle density is discrete may cause to ragged obstacle estimation so that a function is applied to obtain smooth polar obstacle density h'_k:

$$h'_k = \frac{h_{k-l} + 2h_{k-l+1} \dots + lh_k + \cdots + 2h_{k+l+1} + h_{k+l}}{2l+1}; l = 1,2, \dots, n; \qquad (2.6)$$

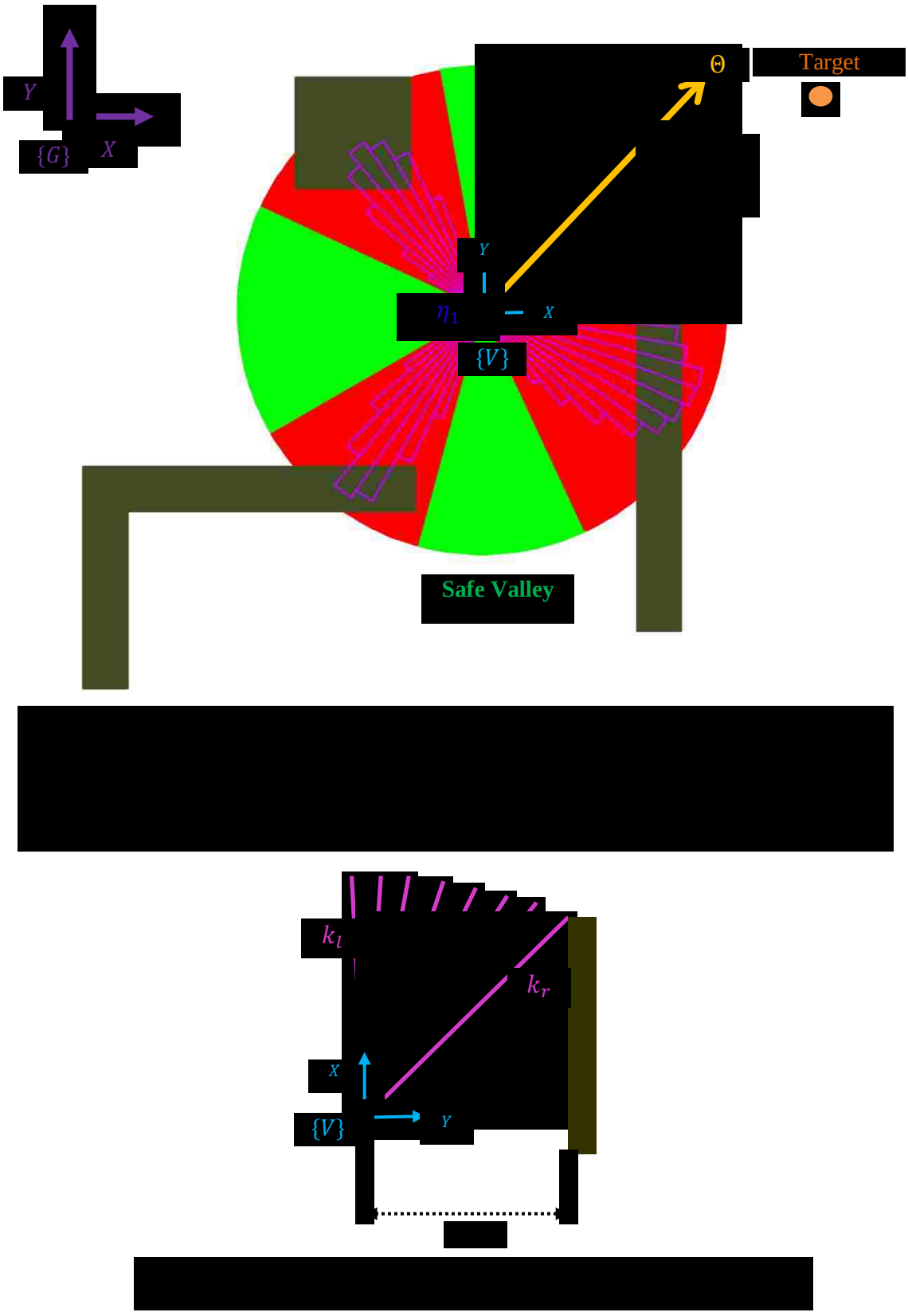
where n is the amount of sectors. l is a constant integer which is chosen depending on experiments or simulations.

As shown in Fig. 2.4, entire sectors defined around the vehicle or robot by the polar histogram are separated into insecure and safe valleys using a threshold η_1.

For determining an optimal direction of the vehicle, safe valleys are divided into narrow and wide valleys with the threshold η_2 (easily chosen from simulation) which is determined in the situation as shown in Fig. 2.5. The vehicle was set close to obstacle in a safe distance D that the one was chosen in wide environment larger than the one in narrow environment. Two specific sectors in the safe valley (not represent all the sectors) indicated as k_l the sector closet to X axis of vehicle and k_r the sector nearest to obstacle were selected. The threshold η_2 was expressed as:

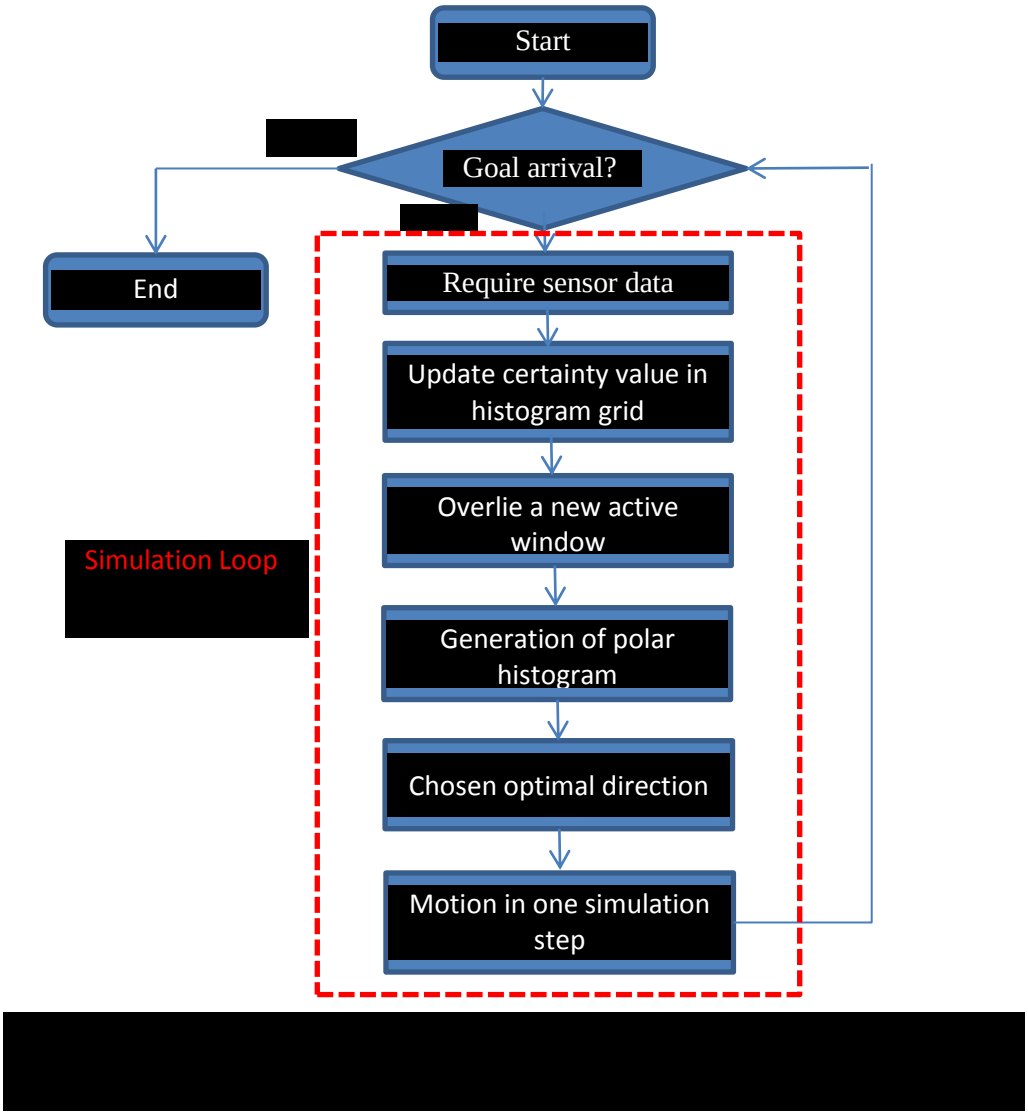
$$\eta_2 = 2(k_l - k_r) * \alpha \qquad (2.7)$$

where α is the angular resolution.

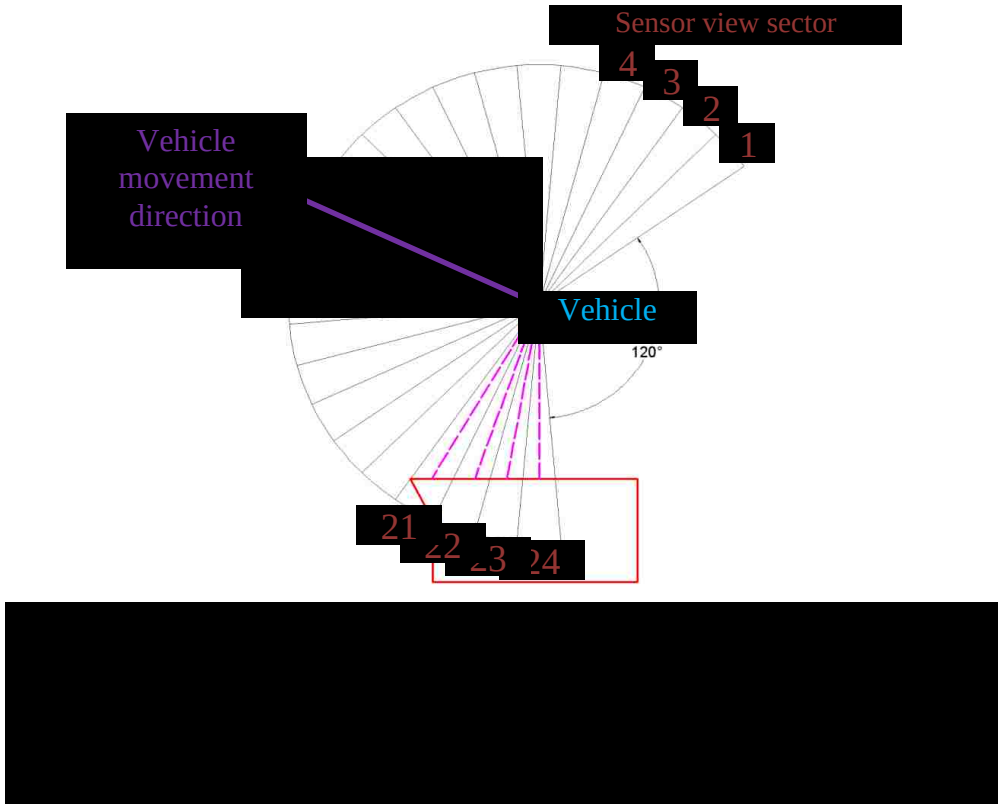

Y
{G}
X
Θ
Target
Y
η1
X
{V}
Safe Valley
kl
kr
X
{V}
Y

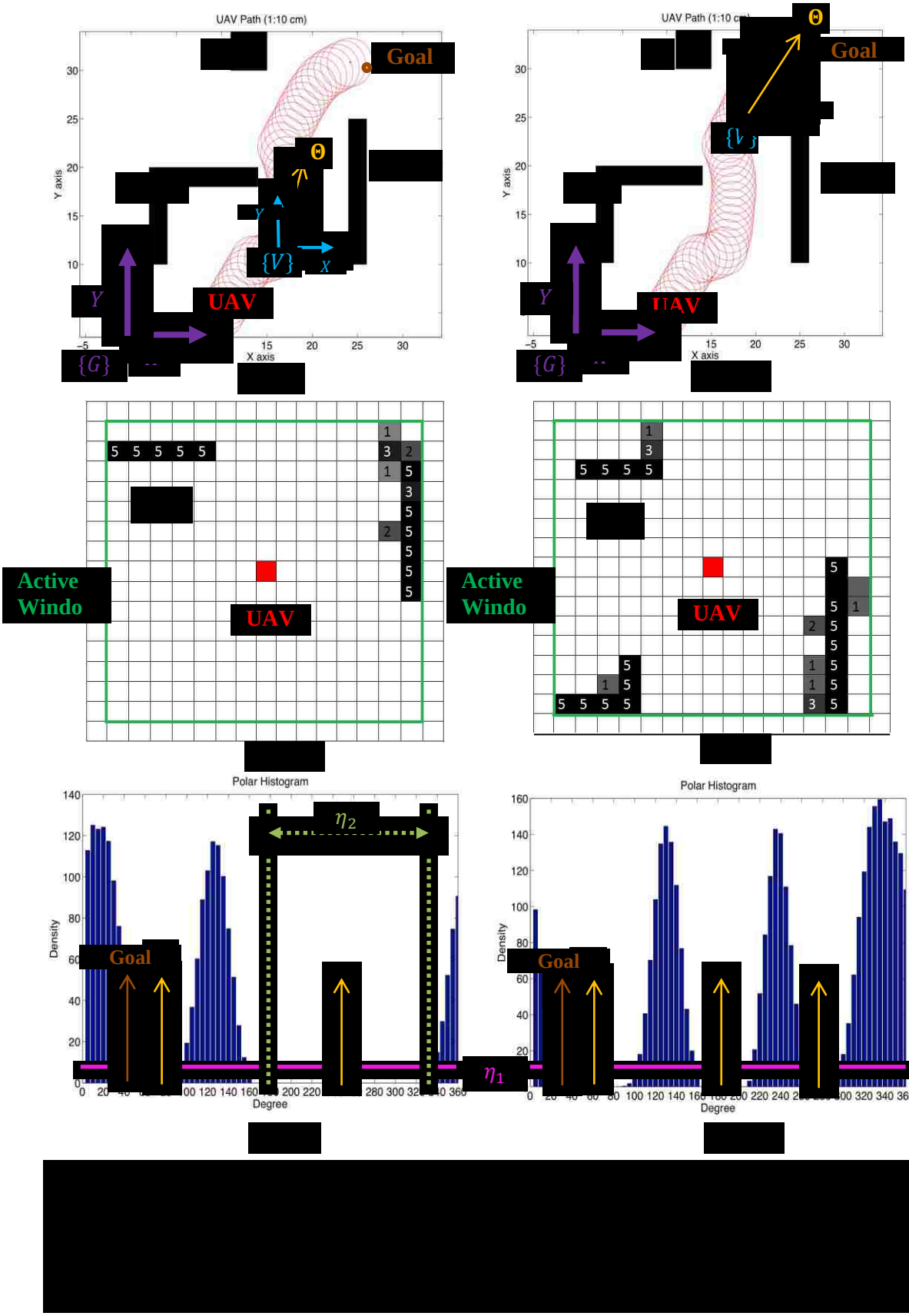
A narrow valley is the one less than η_2 and the sub-optimal direction is considered as the sector in the middle of narrow valley. For a wide valley greater than the η_2, a sub-optimal direction can be selected as a particular sector which has an interval of half threshold η_2 on the boundary closest to the target as shown in Fig. 2.4. In the end, the optimal direction Θ prefers to the sub-optimal direction which is closest to the target and is transformed into vehicle frame to control locomotion.

2.1.2 Simulation Results and Discussion



Computer simulation is carried out for the VFH algorithm using in Matlab [24] and its flow chart shown in Fig. 2.6. In this 2D simulation, it is assumed that a vehicle is equipped with a scanning laser range sensor with a field of view of 240°. Also, vehicle location is known and only kinematic motions of the vehicle are considered. The laser range sensor is simulated in Matlab such that it returns 24 range data in each sampling period and as shown in Fig. 2.7.

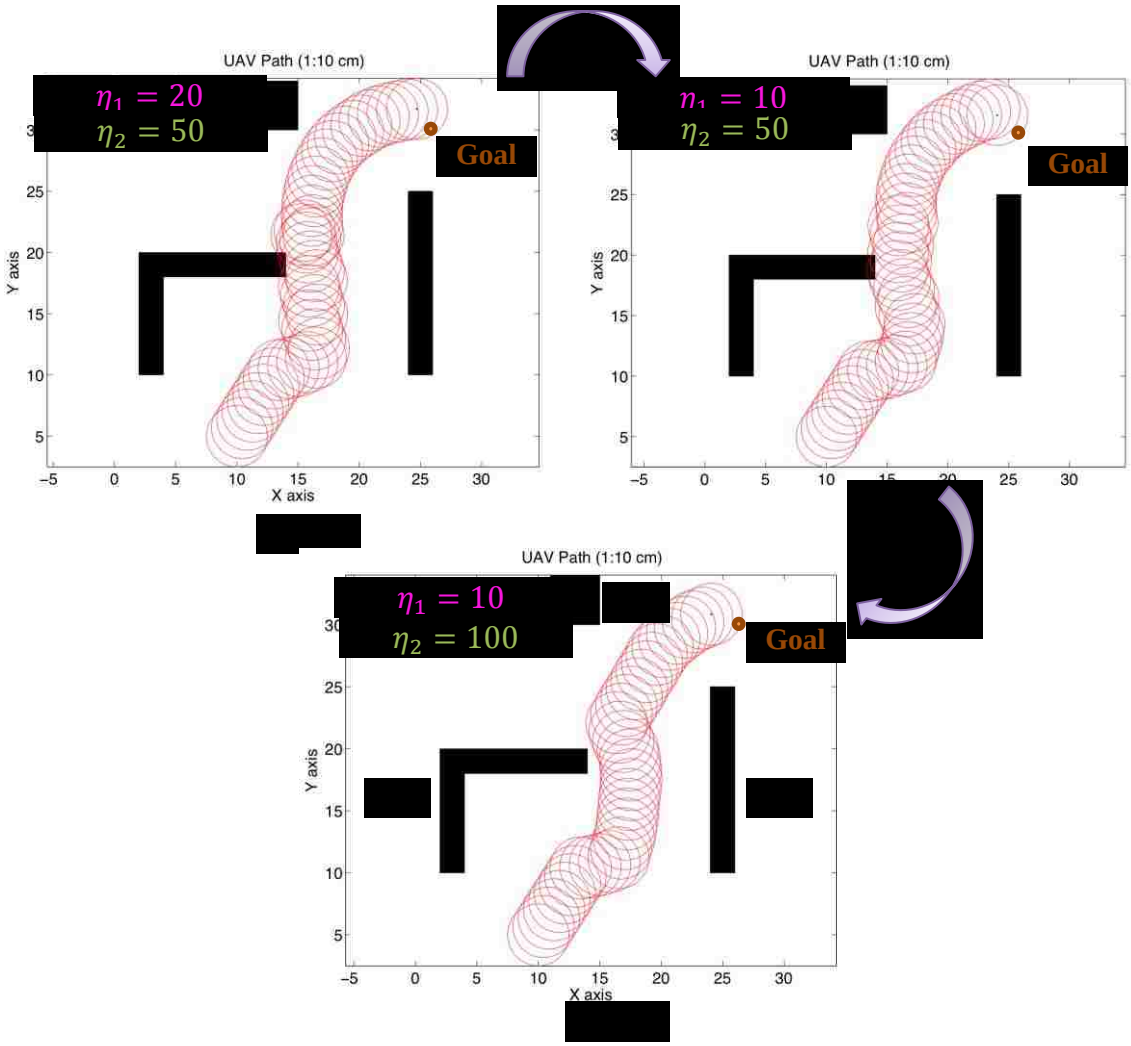



UAV Path (1:10 cm)
Goal
Θ
UAV
{G}
{V}
X axis
Y axis
Active Windo
Polar Histogram
Density
Degree
η_1
η_2

In this simulation, a size of UAV is assumed with a radius of 25 cm circular shape and it moves with a speed of 0.8 m/s. The UAV successfully navigates through environment with unknown obstacle location with relatively smooth path as shown in Fig. 2.8. It should be noted that two UAV positions in Fig. 2.8(a) and Fig. 2.8(d) are particular importance since it is too close to obstacles and can face to more than one option of its next move. As shown in Fig. 2.8(b) and Fig. 2.8(e), corresponding histogram grids represent obstacle distribution with CV values. The larger CV value means higher probability of collision with obstacles. Firstly, η_1 was chosen as 15% of the maximum polar histogram density from a simulation and selection of η_2 was described as Section 2.1.1. Then, both thresholds were tuned in the simulation to obtain best performance so that η_1 equaled to 10 and value of η_2 was 150 at last. In Fig. 2.8(c) and Fig. 2.8(f), safe valleys were determined by η_1 and separated into wide valley and narrow valleys with η_2. The sub-optimal direction which was the nearest one to goal would be selected to optimal direction .

As shown in Fig. 2.9, a smooth path can be obtained by tuning values of two thresholds. For higher value of the threshold η_1 will result in crash with obstacle because some sectors represented area close to obstacles were considered into safe valleys. For the high value of the threshold η_2 it can result in less chance of collision but navigation trajectory became less smooth and stiffer. Because wide valleys existed were regarded as narrow valleys based on high value of η_2 so that sub-optimal directions of those valleys directly were chosen in the middle. Also, one of the drawbacks of this algorithm was shown in Fig. 2.9(c) that the path was close to obstacle 1. The long obstacles such as obstacle 2 would lead more impact on vehicle than short obstacles (obstacle 1) did because of the

method of obstacle estimation which based on the weighting of obstacles in detection area. Another shortcoming of this approach is luxurious computational cost due to the mapping strategy that is storing data all the time.



[redacted]

2.2 Summary

In this Chapter, a detailed introduce of Vector Filed Histogram (VFH) for 2D obstacle avoidance is presented. Additionally, a computer simulation based on Matlab is accomplished assuming laser rangefinder is equipped on the vehicle. Also, simulation results are expressed and demonstrated to discuss the performance of this algorithm. Moreover, drawbacks are presented and analyzed in order to improve in the extended 3D obstacle avoidance.

CHAPTER 3

MODIFIED VFH ALGORITHM AND EXPERIMENT

The 2D navigation mission with utilization of the VFH method has satisfied a reasonable result although the only sensor used is the range sensor. The usage of this sensor is the main reason leading to high computational complexity, because a single reading of ultrasonic sensors results in incomprehensive obstacle information. Currently, high-resolution range sensors such as laser scanner are available for mapping accurate environments for indoor applications. Hence, a modified VFH method proposed uses a laser rangefinder. Following sections explain this algorithm in detail and describe experiment implemented for further verification.

3.1 Review of modified VFH

The modified VFH is a local obstacle avoidance algorithm that generates navigation vectors based on a robot or a vehicle surrounding environment perception. This method can be implemented in autonomous navigation of a ground vehicle or 2D mission of a UAV assuming a laser scanner is onboard. This algorithm is computed on onboard computer and controls are sent to vehicle.

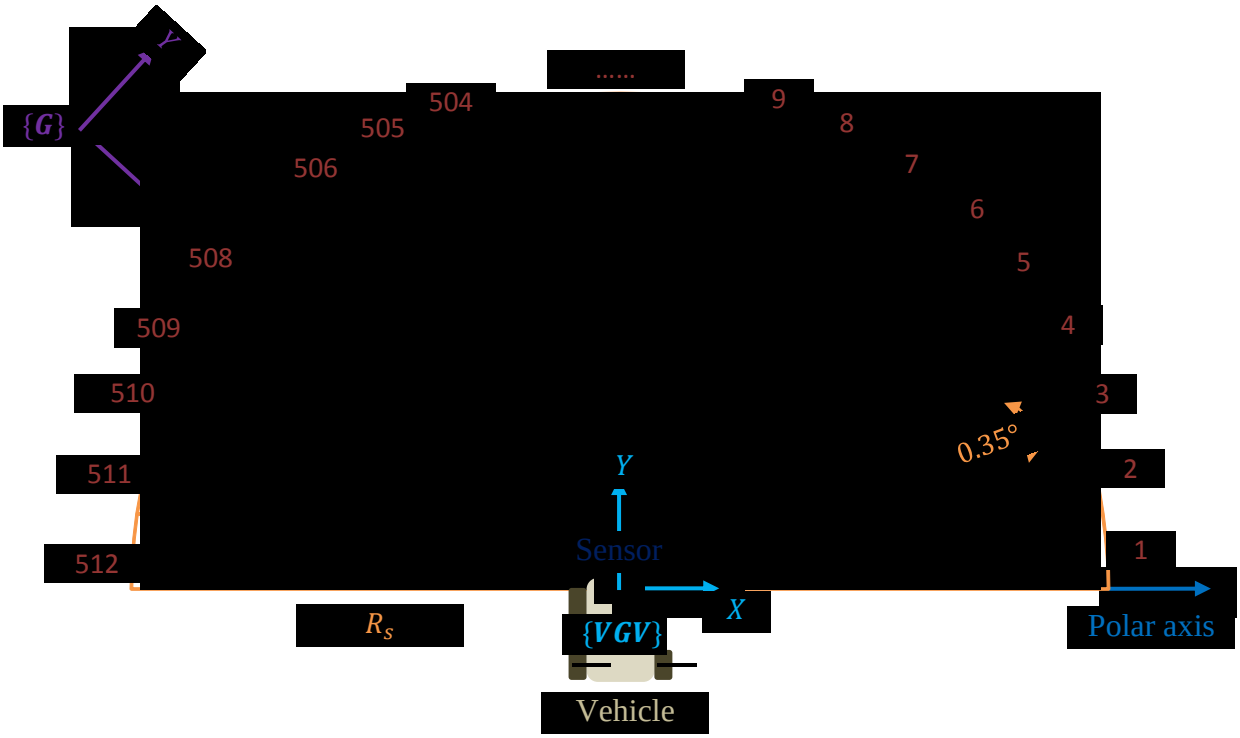
A Vision-based Ground Vehicle (VGV) consists of a laser scanner, onboard computer and a ground vehicle and will both be used in the implementation of the modified VFH algorithm. The laser scanner equipped on the vehicle is assumed to have a FOV of 180° and a radius, denoted as R_s, equal to 4 meters. The detection area of the sensor is divided

into 512 portions every one contains a range value, denoted as RV, and has an angular resolution $\alpha = 0.35°$ as shown in Fig. 3.1.

Each portion is expressed as a vector:

$$Vector_i = (RV),\ i = \{1,2,3,\dots,512\} \qquad (3.1)$$

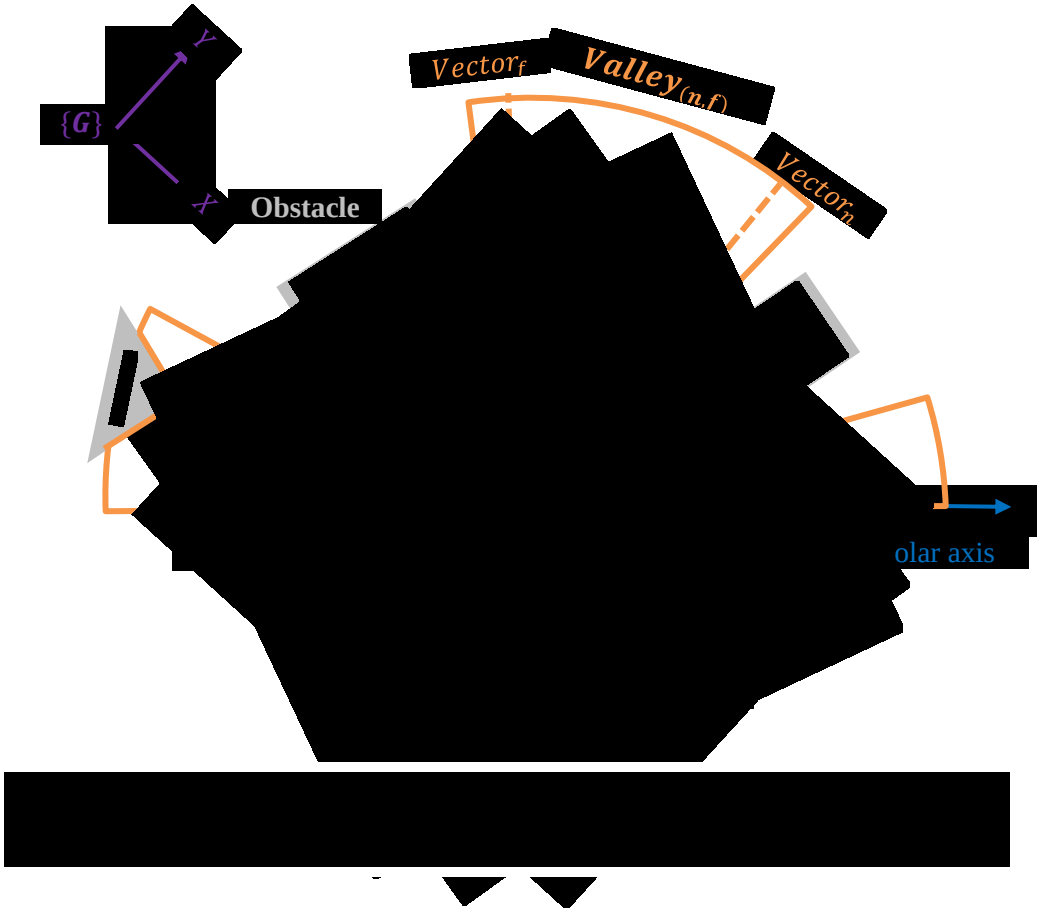
Where i is index of vector in Polar coordinate system that the polar axis as same as the X axis of the frame of the VGV which located in connection point of vehicle and sensor as shown in Fig. 3.1 and RV is the magnitude and equal to R_s.



In order to analyze vector representation of obstacle information, a threshold is applied to determine dangerous valleys and safe valleys to travel through which are composed of merged vectors. The threshold T is expressed as:

$$T = a \tag{3.2}$$

Where a is a constant can be adjust in the experiment.

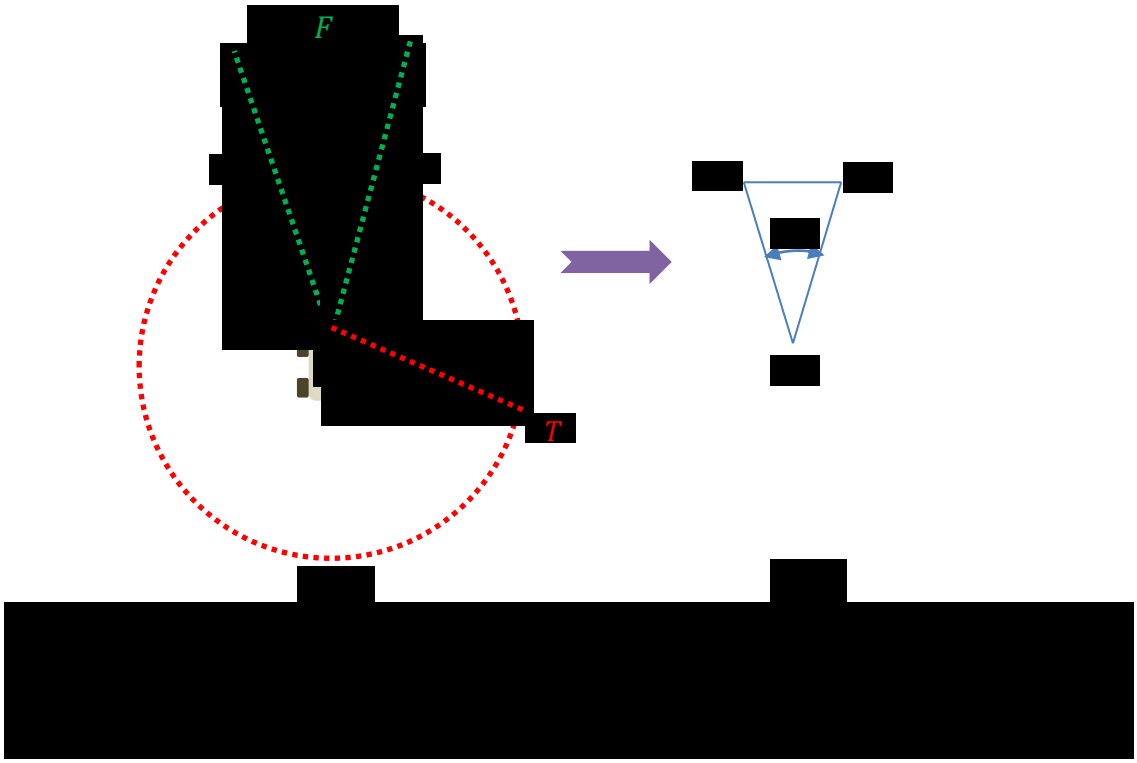


The dangerous valley, aqua zone as shown in Fig. 3.2, consists of continuous vectors which have a RV value less than the threshold. The safe valley, signified by orange outlines as shown in Fig. 3.2, is comprised of sequential vectors whose RV greater than threshold.

In the case that no safe valley exists means that no safe direction is available; in other cases that a safe valley exists, the value of safe valley represents range is recorded as:

$$Valley_{(n,f)} = (n - f + 1) * \alpha;\ (n \in i, f \in i) \quad (3.2)$$

Where n and f are the index of inside vectors which are separately nearest and farthest to the polar axis as shown in Fig. 3.2.The angular resolution of each vector is represented by α.



A high-pass threshold, denoted as F, calculates the minimum valley that can be passed by VGV and removes impassable previously safe valleys. As shown in Fig. 3.3(*a*), the threshold intersects with threshold with three points A, B and C. This situation is modified into a geometry model as shown in Fig. 3.3(b), side $\overline{AC}$ and side $\overline{BC}$ are equal to the threshold T, side $\overline{AB}$ is equivalent to the width of VGV denoted as S. The included angle θ is equal to the value of high-pass threshold and solved by the law of cosines:

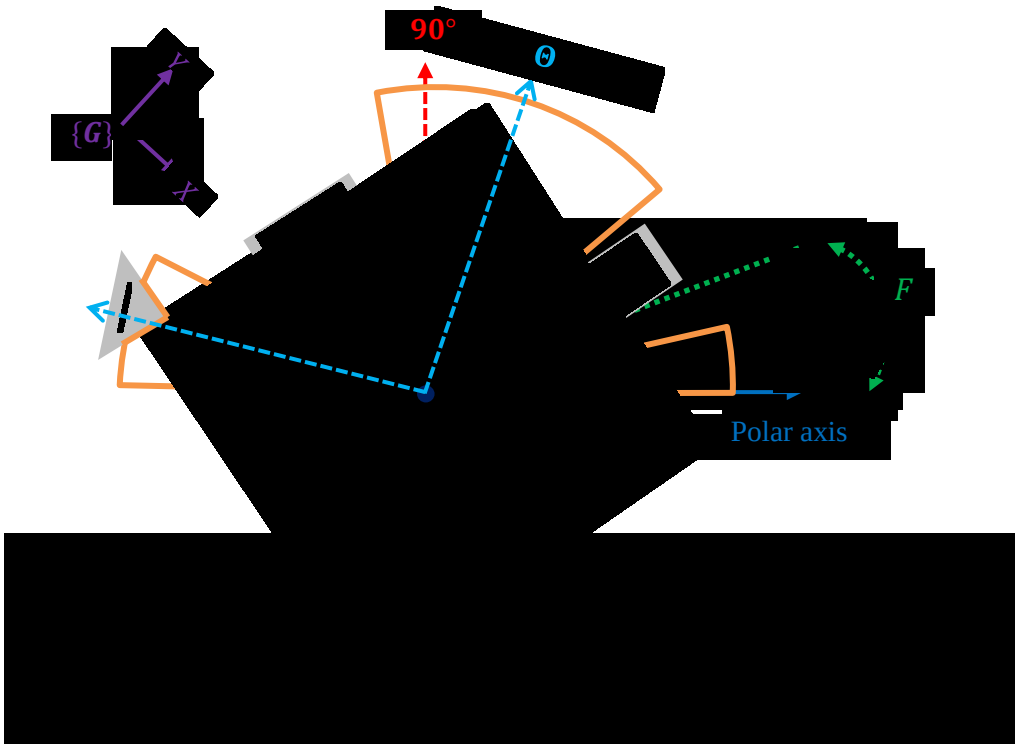
$$\theta = \cos^{-1}(\frac{1 - S^2}{2T^2}) \quad (3.3)$$

As shown in Fig. 3.4, the safe valley marked with a black X is removed by the high-pass threshold. After that every remaining safe valley will have a generated a sub-direction denoted as Θ signifying the middle of the valley using the angular resolution:

$$\Theta = \frac{(f + n - 1)}{2} * \alpha \qquad (3.4)$$

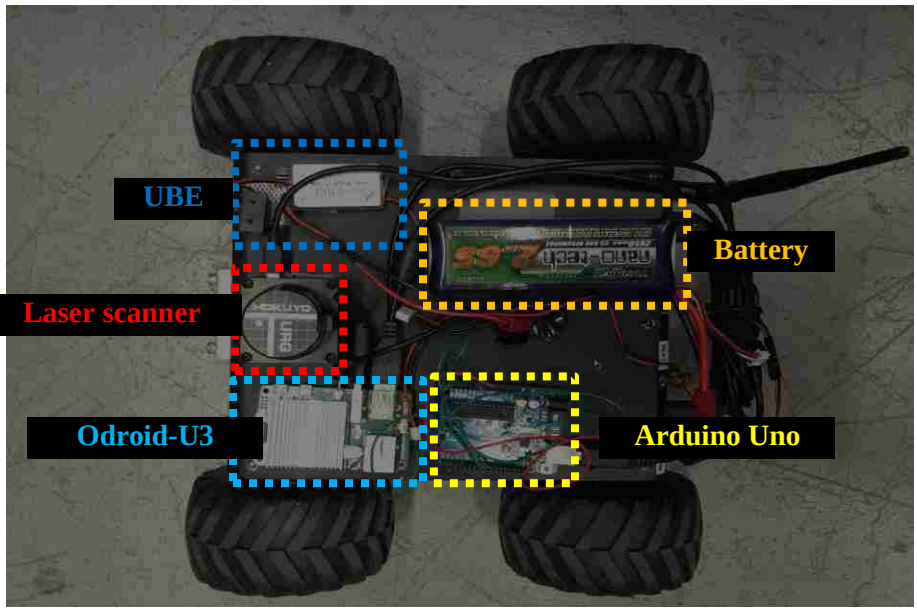
Where n and f are the indexes of boundary vectors of corresponding safe valley.

Finial direction for the vehicle to travel is chosen from the sub-direction that is nearest to 90 degrees from the polar axis as shown in Fig. 3.4. The 90 degree is chosen in order to force the vehicle to travel forward if it can.



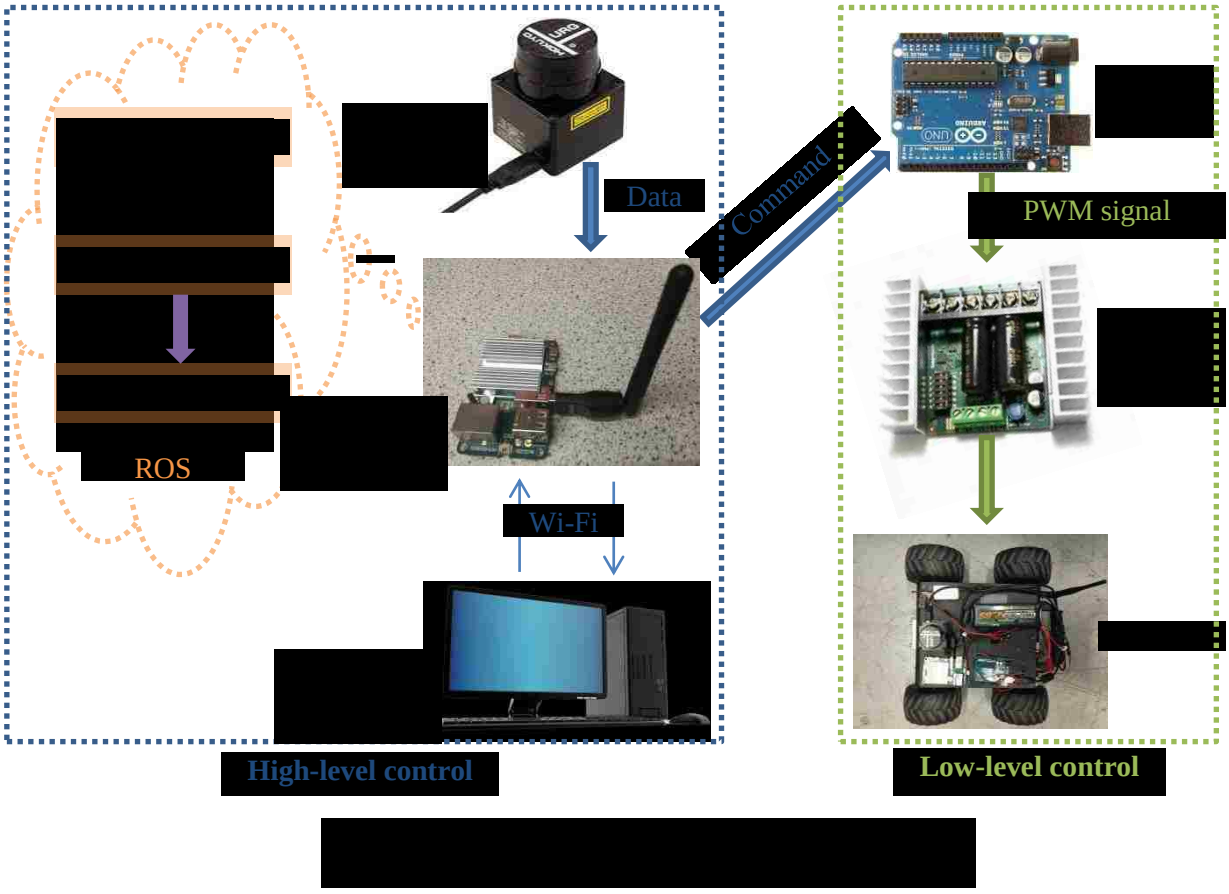
3.2 Development of Ground Vehicle Robot System

3.2.1 System Setup



The laser scanner sensor and on-board computer could be easily broken if a UAV crashed so a ground vehicle is used first to the test effectiveness of the 2D algorithm. This platform used the MiniBot, a commercially available 4-wheel-drive vehicle, from Inspector Bots [22]. To develop the VGV system with autonomous obstacle avoidance, the onboard computer Odroid-U3 [23] with strong processing capacity, a Hokuyo URG-04LX-UG01 laser scanner [24] and an Arduino Uno microcontroller board [25] which outputs PWM signal, the motor driver of the vehicle were all added to the platform as shown in Fig.3.5. The Odroid-U3 is a powerful Linux computer with 1.7GHz Quad-Core processor

and 2GByte RAM. The laser scanner employs a 180-degree FOV (other mode of FOV which is 240-degree was not used in this system) and a maximum range of 4 meters. Fig. 3.6 illustrates the system setup. The laser scanner sends distance data of obstacles to



the onboard computer with Robot Operating System (ROS) [26] framework installed in order to implement high-level control constituting of the computation of the modified VFH algorithm. ROS nodes that execute calculation and communication were used. Hokuyo node collected the range data, algorithm node computed the final direction and serial node sent the commands to the Arduino from the onboard computer. A client computer remotely controlled desktop of onboard computer to launch ROS nodes with Wi-Fi. In the low-level control, the Uno board received a command denoting which was

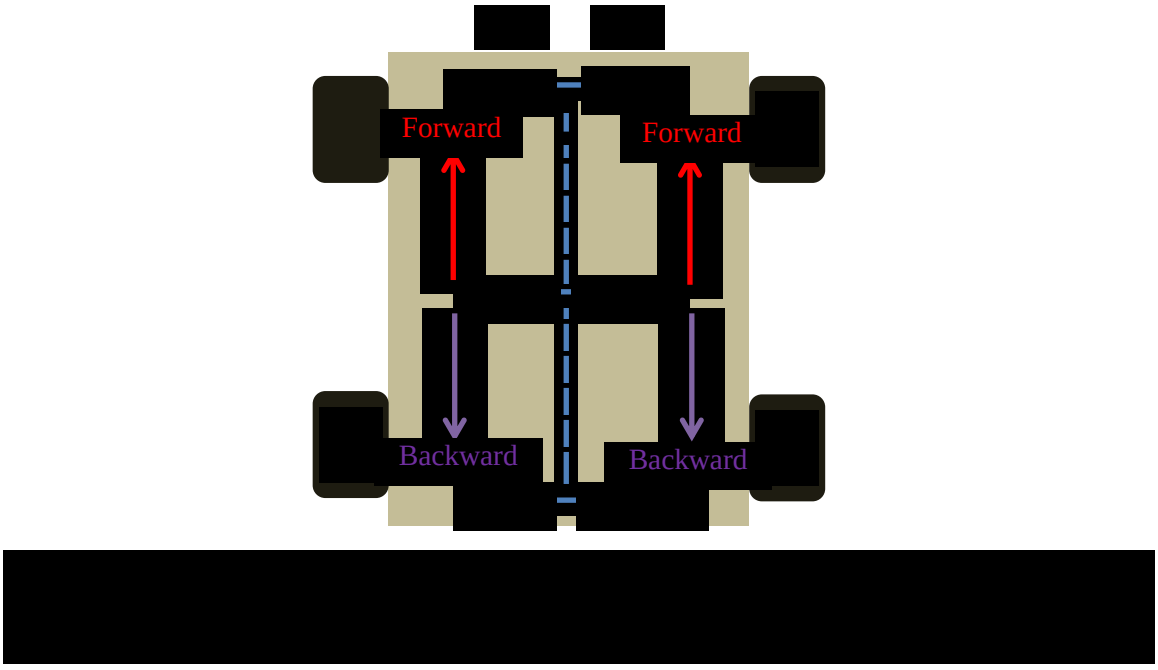
the safe direction and converted that directional command into PWM signals that were sent to motor driver to manipulate the VGV.

3.2.2 Low level control of VGV

The Uno board calculated the control values denoted as ζ_l and ζ_r after it has received its command from the high-level control. ζ_l and ζ_r were defined from 0 to 180.0 and scaled to corresponding PWM signals to drive left two motors and right two motors respectively based on an Arduino Servo library [34]. However, only the range between 25.0 and 155.0 can generate appropriate PWM signals to actuate motors in the test. The relation between control values and motor directions is shown in Fig. 3.7 and are expressed as:

$$\begin{cases} 90.0 < \zeta_l \leq 155.0, left\ motors\ go\ forward \\ \zeta_l = 90.0, left\ motors\ stop \\ 25.0 \leq \zeta_l < 90.0, left\ motors\ go\ backward \end{cases} \quad (3.4)$$

$$\begin{cases} 25.0 \leq \zeta_r < 90.0, right\ motors\ go\ forward \\ \zeta_r = 90.0, right\ motors\ stop \\ 90.0 < \zeta_r \leq 155.0, right\ motors\ go\ backward \end{cases} \quad (3.5)$$



For the vehicle platform the 4 kinds of locomotion modes used were rotation, advance, left-turn and right-turn which were determined by control values. The rotation mode was defined as rotating clockwise while the command from high-level control indicated there was no safe direction available and expressed as:

$$\begin{cases}\zeta_l = 125.0 \\ \zeta_r = 125.0\end{cases} \tag{3.6}$$

where angular velocity of rotation became faster if the value of ζ_l and ζ_r were increased. If the command showed that final direction Θ is existed, an error denoted as ε was defined to determine other locomotion of VGV and expressed as:

$$\varepsilon = |90° - \Theta| \tag{3.7}$$

The advance mode that forced VGV to go straightly satisfied the following condition:

$$\begin{cases}\zeta_l = 115.0 \\ \zeta_r = 65.0\end{cases}, if\ \varepsilon \leq \gamma \tag{3.8}$$

where γ was an adjustable error angle, and was selected as 8°.Increase of ζ_l and decrease of ζ_r would raise the forward velocity but if the condition that $\zeta_l + \zeta_r = 180.0$ was guaranteed.

In the right-turn mode which satisfied the conditions $\Theta < 90°$ and $\varepsilon > \gamma$, a turning ration λ was defined as:

$$\lambda = \frac{\zeta_l - 90}{90 - \zeta_r} \tag{3.9}$$

where larger value of λ indicated drastic momentary turning.

First of all, two set of relations between Θ and λ were expressed as:

$$\begin{cases}\Theta_1 = 30° \\ \lambda_1 = 10\end{cases} and \begin{cases}\Theta_2 = 10° \\ \lambda_2 = 20\end{cases} \tag{3.10}$$

where λ_1 and λ_2 were adjustable depend on performance of experiments.

Furthermore, λ_1 and λ_2 were substituted into Eq. 3.11 to obtain ζ_{r_1} and ζ_{r_1} respectively.

$$\zeta_r = 90 - \frac{\zeta_l - 90}{\lambda} \tag{3.11}$$

where ζ_l was selected as 135 in right-turn mode.

Moreover, it's assumed that Θ and ζ_r were satisfied the first order equation just as:

$$\zeta_r = k\Theta + \mathrm{b} \tag{3.12}$$

where k and b were solved by substituting two points (Θ_1, ζ_{r_1}) and (Θ_2, ζ_{r_2}).

Lastly, the right-turn mode was expressed as:

$$\begin{cases} \zeta_l = 135.0 \\ \zeta_r = -0.13125\Theta + 88.3125 \end{cases} \tag{3.13}$$

In the left-turn mode which satisfied the conditions $\Theta > 90°$ and $\varepsilon > \gamma$, a turning ration λ was defined as:

$$\lambda = \frac{90 - \zeta_r}{\zeta_l - 90} \tag{3.14}$$

where ζ_r was selected to 45 in left-turn mode.

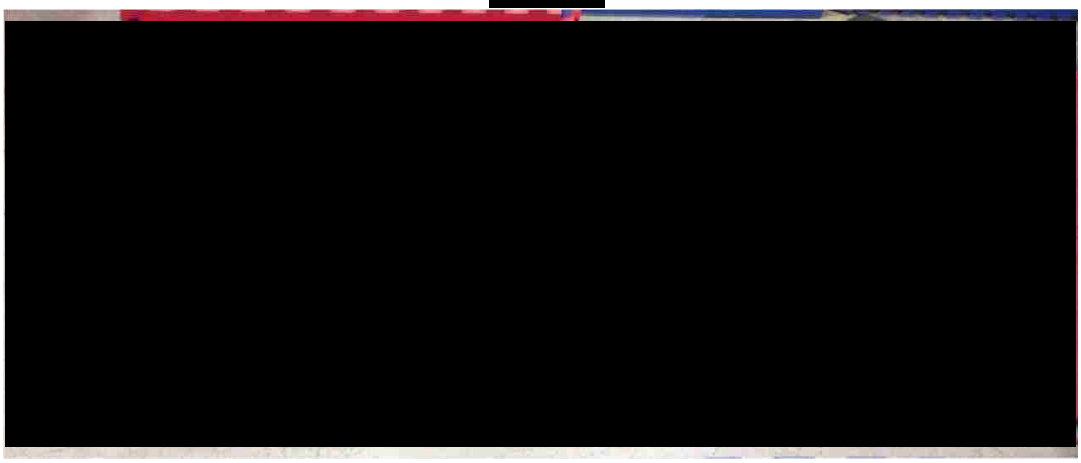
Substituting the supplementary angle of final direction which is $(180 - \Theta)$ into Eq. 3.13 and substituting the received solution into Eq. 3.9 to obtain the turning ratio which should equal to Eq. 3.14 because of symmetry of the turning ratio. And the result was:

$$\begin{cases} \zeta_l = -0.13125\Theta + 115.3125 \\ \zeta_r = 45 \end{cases} \tag{3.15}$$

3.3 Result and Discussion

[redacted]

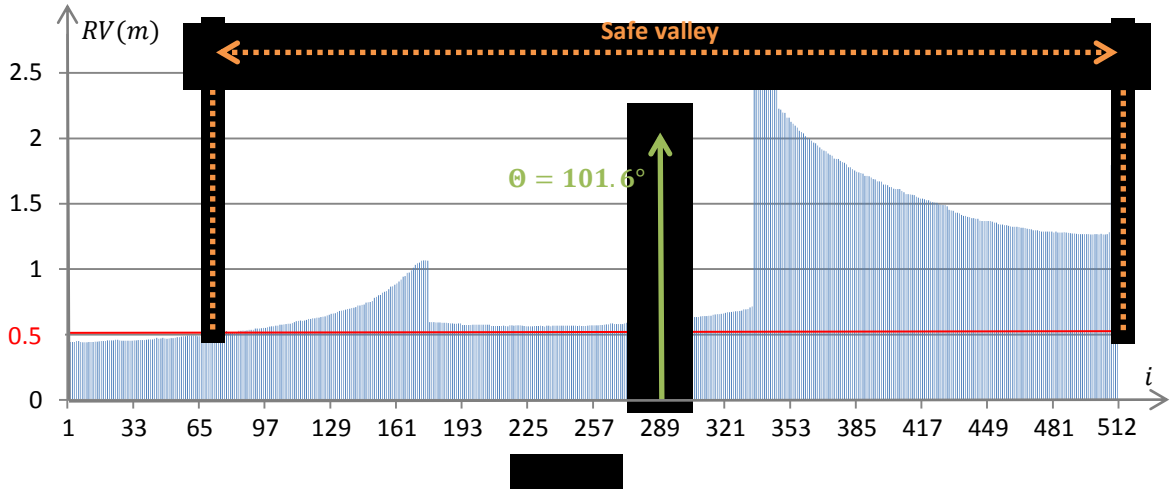
[redacted]

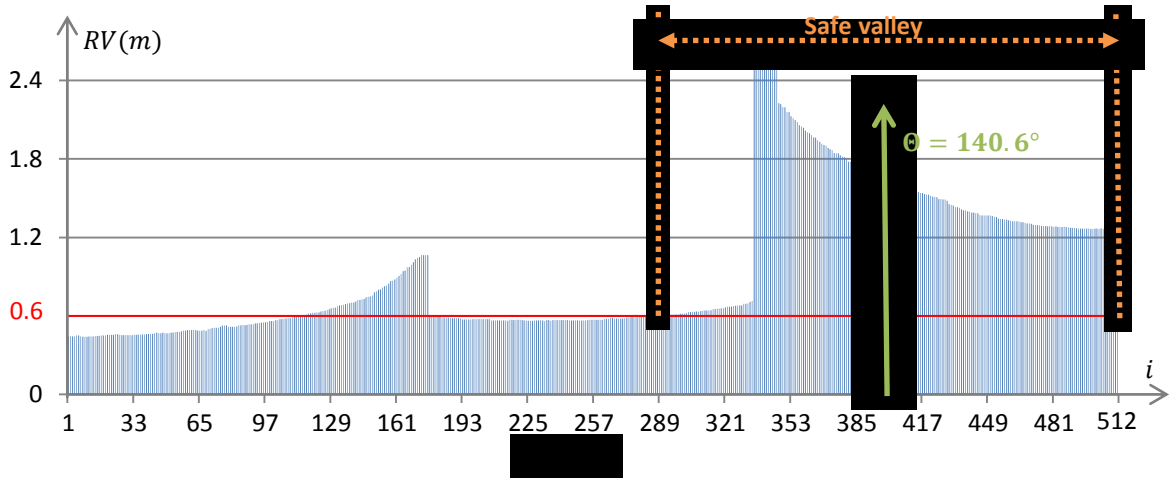


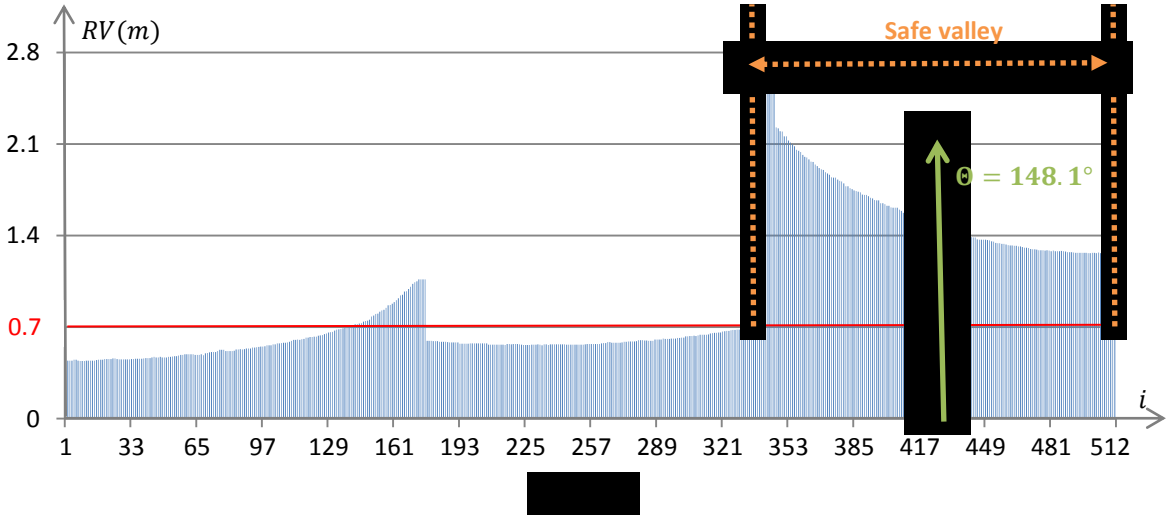
[redacted]

[redacted]

Three experiments were implemented in order to find the best appropriate threshold of 2D modified VFH algorithm and obtain optimal performance. The result of first experiment that was applied a threshold of 0.5m was fail because the VGV didn't avoid the obstacle front of it as shown in Fig. 3.8(a). Obviously, the ugly left-wheel leaded this collision so two possible reasons that one was dynamic problem of VGV and the other one was incorrect threshold occurred. In the Fig. 3.8(b), the threshold was increased to 0.6 m and VGV avoid the front obstacle successfully but the performance was not perfect because it was close to obstacle and almost crashed the following obstacle. Therefore, the main reason was threshold so that it was raised to 0.7 m in the next experiment and the result was shown in Fig. 3.8(c). The VGV not only navigated between unknown obstacles without causing any collision but also kept safe range with obstacles and the path was perfectly smooth.


RV(m)
Safe valley
Θ = 101.6°
2.5
2
1.5
1
0.5
0
i
1
33
65
97
129
161
193
225
257
289
321
353
385
417
449
481
512


RV(m)
Safe valley
Θ = 140.6°
2.4
1.8
1.2
0.6
0
i
1
33
65
97
129
161
193
225
257
289
321
353
385
417
449
481
512


RV(m)
Safe valley
Θ = 148.1°
2.8
2.1
1.4
0.7
0
i
1
33
65
97
129
161
193
225
257
289
321
353
385
417
449
481
512

In the Fig. 3.9, a special position which at the corner was surrounded by obstacles selected to demonstrate the reason of success of obstacle avoidance applied with relative large threshold. As shown in Fig. 3.10(a), the safe valley, assumed the exceptional safe valley had been removed by high-pass threshold so that didn't display, with threshold of 0.5m was very broad and even the range representing front obstacle approximately from $i = 173$ to $i = 300$ located in safe valley. Then the inexact direction equaled to 101.6° leaded to collision at last. In Fig. 3.10(b), the edge area of obstacle still enclosed in the safe valley so that the VGV was close to obstacle but might hit in other situations. With increasing the value of threshold, the safe valley diminished but still was not risk for going through.

3.4 Summary

This Chapter presents a modified VHF obstacle avoidance algorithm for 2D navigation of vehicle based on laser scanner. Also a VGV system was developed and detail discussion is presented including hardware structure and software integration. ROS environment will be used for high-level control which is the same framework implemented in the following chapter for simulation. This autonomous system will be extended for the aerial vehicle platform. Moreover, experiment result is demonstrated and analysis.

CHAPTER 4

3D OBSTACLE AVOIDANCE ALGORITHM

The 3D navigation missions or exploration tasks are studied for multicopter in this chapter. In the 3D obstacle avoidance task, necessary sensor based detection of environments always extremely challenging. Currently, a simple 3D camera such as Kinect sensor [30] can achieve reasonable accuracy in three-dimensional range data but has relatively small field of view (FOV) for effective navigation in unstructured environments. The proposed 3D vector mesh (VM) algorithm can make up this weakness of sensor. The 3D VM algorithm is an extension of the 2D algorithm discussed in Chapter 3 for ground vehicle navigation, and it consists of three core stages: (1) voxel obstacle computation, (2) vector obstacle estimation, and (3) binary mesh representation. The advantage of 3D VM approach is that range data can be provisionally stored for adequate obstacle perception enough for free-collision path generation. The descriptions of these three stages and computer simulation results have been discussed in detail in the following chapter.

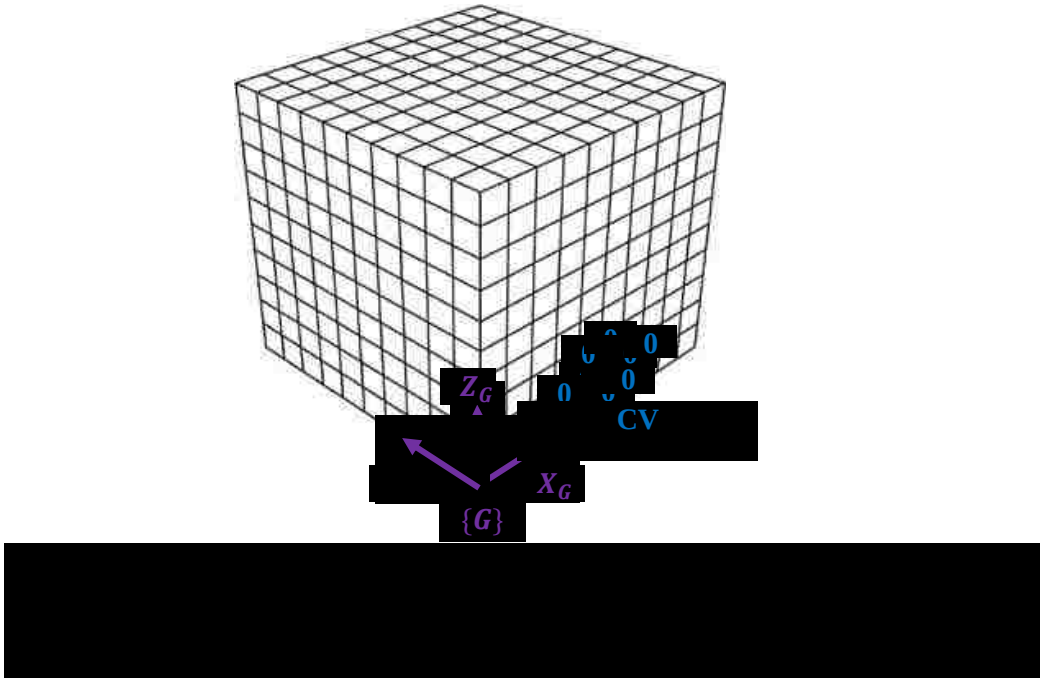
4.1 Voxel Obstacle Estimation

The global world space is described by three-dimensional Cartesian voxels and its size, l, can be determined by two factors. One is a computational cost and the other is detection accuracy of environment. Too small size can be computationally expensive and too large size results in potentially incorrect obstacle estimation. The voxel size can be adjusted

depending on different applications and easily tuned using computer simulation. The voxel is assumed to have a dimension of 10cm × 10cm × 10 cm as shown in Fig. 4.1. The voxel is expressed as:

$$Voxel_{(X_G,\ Y_G,\ Z_G)} = (CV) \tag{4.1}$$

Where X_G, Y_G, Z_G mean coordinate of voxel in global frame which is denoted as $\{G\}$; CV is the obstacle probability for this voxel.



The entire process is that sensors explore local environments and provide estimation of obstacle presence in global space expressed by voxels. This procedure of voxel representation of obstacles comprises of two simple phases: (1) transformation of obstacle position and (2) mapping of obstacles to voxels.

4.1.1 Transformation of Obstacle Position

It is assumed that, a Kinect sensor from Microsoft® is used for detecting obstacles in real-time. Each pixel of a depth image obtained from the sensor is stored as range data. ${}^{K}\boldsymbol{P} = [{}^{K}x \ {}^{K}y \ {}^{K}z]^{T}(m)$ that obstacle position measured with respect to a Kinect frame $\{K\}$. The range points ${}^{K}\boldsymbol{P}$ needs to be transformed to the global frame, as shown in Fig. 4.2 using the rotation matrix ${}^{G}_{U}\boldsymbol{R}$ defined for the UAV frame using X-Y-Z Euler angles and is denoted as $\{U\}$,

$$ {}^{G}_{U}\boldsymbol{R} = \underbrace{\begin{bmatrix} C(\alpha_1)C(\beta_1) & C(\alpha_1)S(\beta_1)S(\gamma_1) - S(\alpha_1)C(\gamma_1) & C(\alpha_1)S(\beta_1)C(\gamma_1) + S(\alpha_1)S(\gamma_1) \\ S(\alpha_1)C(\beta_1) & S(\alpha_1)S(\beta_1)S(\gamma_1) + C(\alpha_1)C(\gamma_1) & S(\alpha_1)S(\beta_1)C(\gamma_1) - C(\alpha_1)S(\gamma_1) \\ -S(\beta_1) & C(\beta_1)S(\gamma_1) & C(\beta_1)C(\gamma_1) \end{bmatrix}}_{3\times3} \quad (4.2) $$

where $\gamma_1, \beta_1, \alpha_1$ indicate rotation angles in $\hat{X}_G, \hat{Y}_G, \hat{Z}_G$ axes respectively, and each rotation angle is assumed is be obtained from an inertial measurement unit (IMU) mounted on UAV. It should be noted that $C(\alpha_1) = \cos(\alpha_1)$ and $S(\alpha_1) = \sin(\alpha_1)$.

The rotation matrix, ${}^{G}_{U}\boldsymbol{R}$, between UAV and Kinect also expressed in the similar way as:

$$ {}^{U}_{K}\boldsymbol{R} = \underbrace{\begin{bmatrix} C(\alpha_2)C(\beta_2) & C(\alpha_2)S(\beta_2)S(\gamma_2) - S(\alpha_2)C(\gamma_2) & C(\alpha_2)S(\beta_2)C(\gamma_2) + S(\alpha_2)S(\gamma_2) \\ S(\alpha_2)C(\beta_2) & S(\alpha_2)S(\beta_2)S(\gamma_2) + C(\alpha_2)C(\gamma_2) & S(\alpha_2)S(\beta_2)C(\gamma_2) - C(\alpha_2)S(\gamma_2) \\ -S(\beta_2) & C(\beta_2)S(\gamma_2) & C(\beta_2)C(\gamma_2) \end{bmatrix}}_{3\times3} \quad (4.3) $$

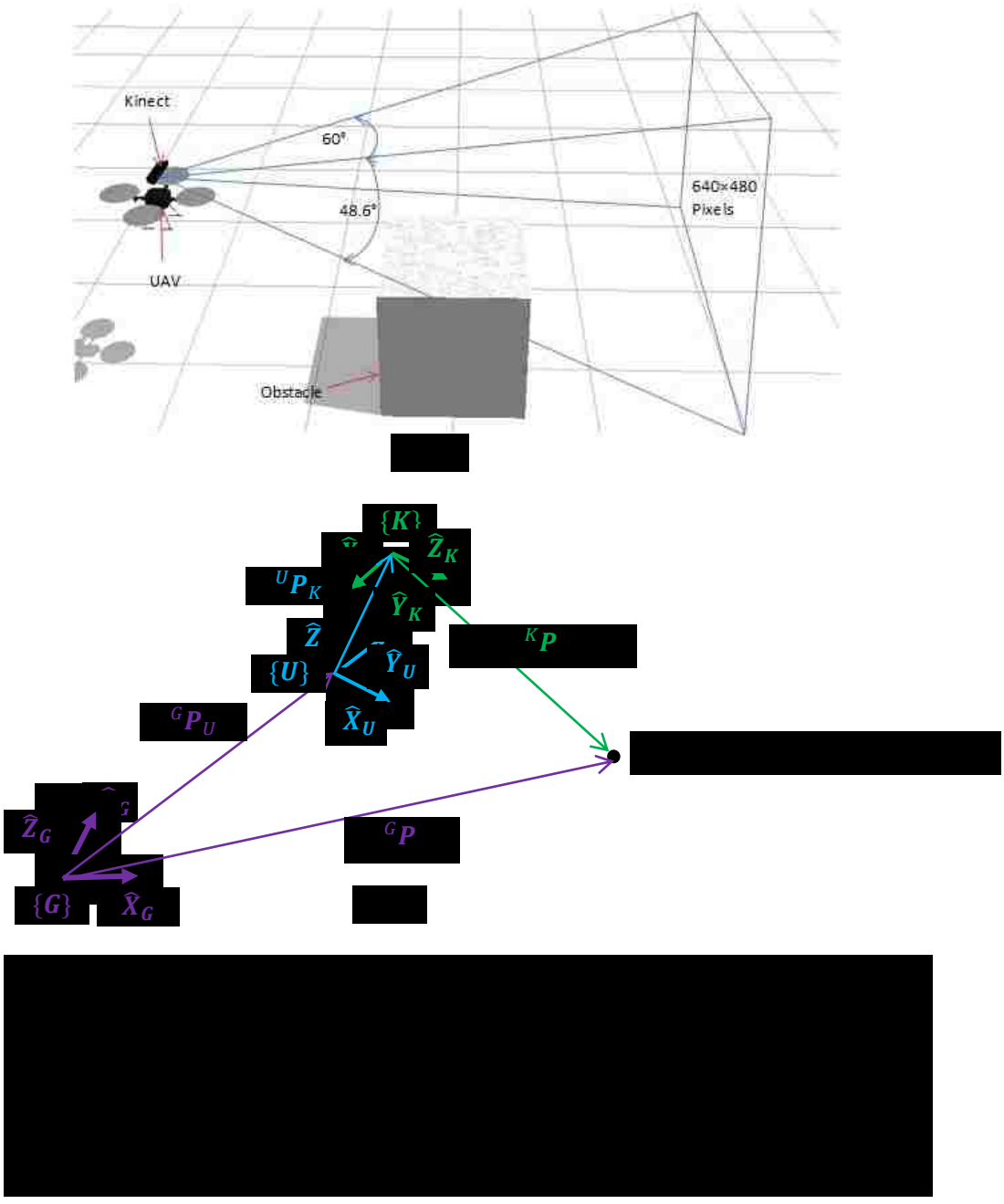
where $\gamma_2, \beta_2, \alpha_2$ respectively represent rotation angles in $\hat{X}_U, \hat{Y}_U, \hat{Z}_U$ axes defined on UAV frame, where rotation angles are known because the Kinect sensor is fixed on UAV frame.

$$ \underbrace{\begin{bmatrix} {}^{G}\boldsymbol{P} \\ 1 \end{bmatrix}}_{4\times1} = \underbrace{\begin{bmatrix} {}^{G}_{U}\boldsymbol{R}\,{}^{U}_{K}\boldsymbol{R} & {}^{G}_{U}\boldsymbol{R}\,{}^{U}\boldsymbol{P}_{K} + {}^{G}\boldsymbol{P}_{U} \\ 0_{1\times3} & 1 \end{bmatrix}}_{4\times4} \underbrace{\begin{bmatrix} {}^{K}\boldsymbol{P} \\ 1 \end{bmatrix}}_{4\times1} \quad (4.4) $$

where ${}^{G}\boldsymbol{P} = [{}^{G}x \ {}^{G}y \ {}^{G}z]^{T}(m)$ is the position of a target point in global frame; ${}^{G}\boldsymbol{P}_{K}$ is the position of the Kinect frame origin in UAV frame and it is known.

${}^{G}\boldsymbol{P}_{U} = [{}^{G}x_U \ {}^{G}y_U \ {}^{G}z_U]^{T}(m)$ is the position of the UAV frame origin in a global frame

and assumed known from onboard IMU sensor. Substituting Eq. 4.2 and Eq. 4.3 into Eq. 4.4, the transformed position ${}^{G}\boldsymbol{P}$ can be calculated.



4.1.2 Mapping

The next step is to transform range data obtained in the previous step to be mapped into corresponding voxels in the global frame. The certain value (CV) of voxel will increase by 1at each data sampling up to 20 whose value can be determine by a number of factors including a vehicle speed and complexity of environments. Fig. 4.3 shows this mapping procedure expressed as:

$$\begin{bmatrix} X_G \\ Y_G \\ Z_G \end{bmatrix} = \begin{bmatrix} ceil(\frac{{}^G x}{l}) \\ ceil(\frac{{}^G y}{l}) \\ ceil(\frac{{}^G z}{l}) \end{bmatrix} \quad (4.1)$$

where $ceil$ is a function returning smallest integer greater than or equal to a given number. X_G, Y_G, Z_G are coordinates of voxels and ${}^G x, {}^G y, {}^G z$ are the coordinates of any target-points measured in global frame. l is the size length of voxel.

For computational efficiency of the proposed algorithm, a local space is defined around the UAV which is fixed to the UAV. This local space is a cube and its size, L, can be determined by the maximum detectable range of on-board sensor using the following relationship:

$$\begin{cases} L = n * l \\ n = 2 * ceil\left(\frac{R_s}{l}\right) - 1 \end{cases} \quad (4.6)$$

where n is the number of voxels in each row or column in the local space. R_s is the sensor range.

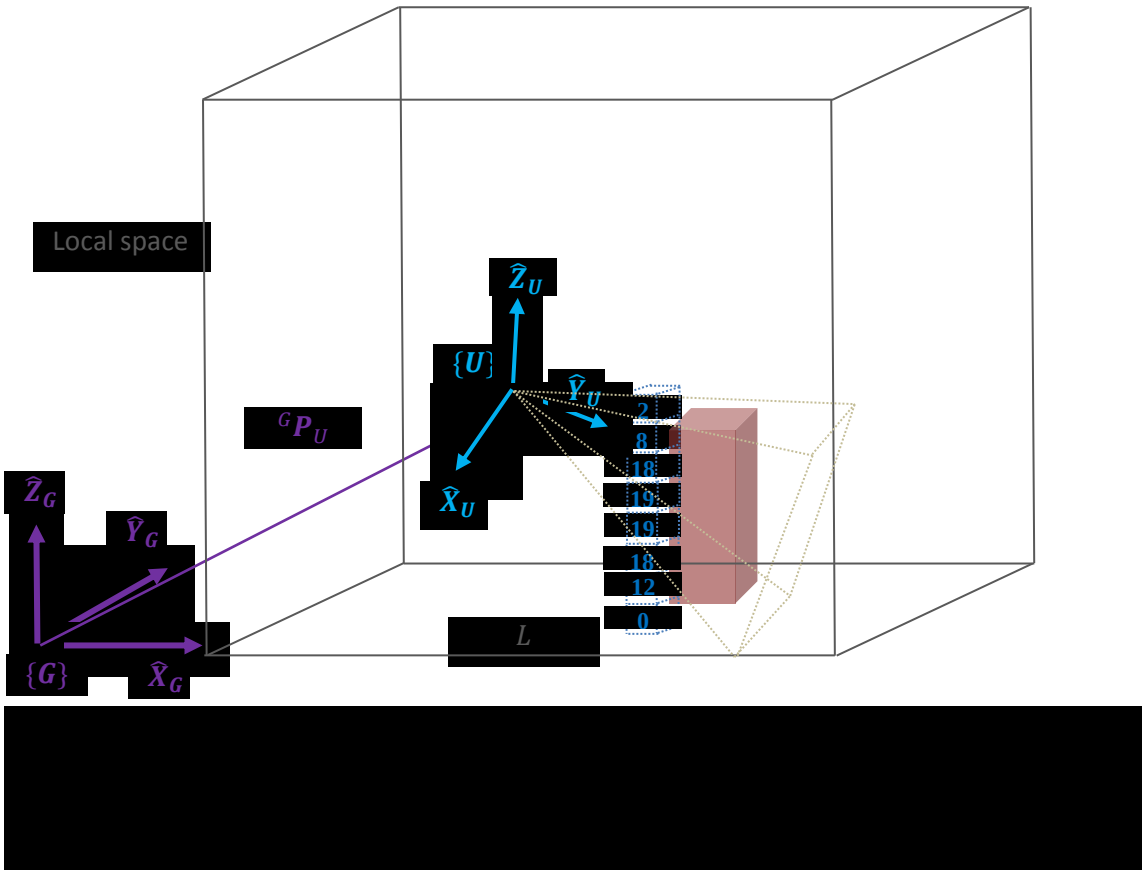
If voxel coordinates X_G, Y_G, Z_G are satisfied with the following conditions then voxels will be considered inside of the local space:

$$\begin{cases} {}^{G}X_U - \dfrac{n+1}{2} \ll X_G \ll {}^{G}X_U + \dfrac{n+1}{2} \\ {}^{G}Y_U - \dfrac{n+1}{2} \ll Y_G \ll {}^{G}Y_U + \dfrac{n+1}{2} \\ {}^{G}Z_U - \dfrac{n+1}{2} \ll Z_G \ll {}^{G}Z_U + \dfrac{n+1}{2} \end{cases} \tag{4.2}$$

where ${}^{G}X_U, {}^{G}Y_U, {}^{G}Z_U$ are the voxel coordinates corresponding to UAV location in global frame and can be obtained using Eq. 4.8 which is similar to Eq. 4.5.

$$\begin{bmatrix} {}^{G}X_U \\ {}^{G}Y_U \\ {}^{G}Z_U \end{bmatrix} = \begin{bmatrix} ceil(\dfrac{{}^{G}x_U}{l}) \\ ceil(\dfrac{{}^{G}y_U}{l}) \\ ceil(\dfrac{{}^{G}z_U}{l}) \end{bmatrix} \tag{4.8}$$

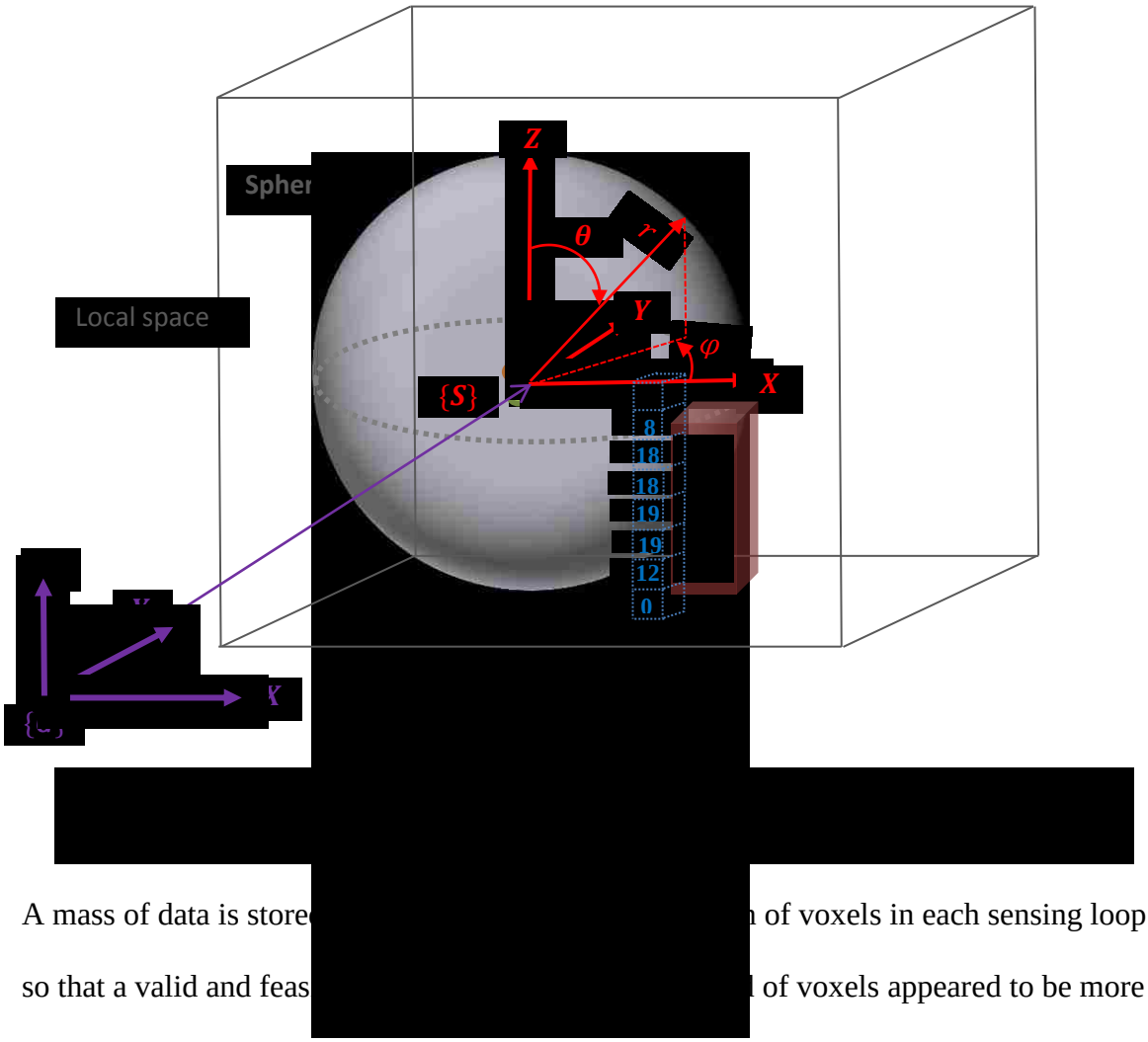
where ${}^{G}x_U, {}^{G}y_U, {}^{G}z_U$ are the coordinates of UAV in the global frame.



While UAV is navigating in unknown environment, voxels inside of local space will be stored so that relatively detailed perception of environments is achieved.

4.2 Vector Obstacle Computation

This section introduces how to convert voxel obstacle representation into vector based obstacle presentation. Firstly, a sphere space is defined in the local space; additionally, meshing of the sphere surface is to obtain vectors which divide an entire space; at the end, computation of vector magnitude which represents obstacle proportion in this direction.



A mass of data is store[illegible] of voxels in each sensing loop so that a valid and feas[illegible] of voxels appeared to be more

attractive in 3D obstacle avoidance. The following example shows validity of the proposed approach:

A spherical space is defined as shown in Fig. 4.4 with radius R equal to 60% of sensor range R_s. The UAV is not only located at the center of sphere but also moves synchronously with it. The sphere frame, denotes as {S}, is parallel to the global frame as in Fig. 4.4. The reason that the sphere space is smaller than the local map is that the voxels inside of sphere can be good enough for estimating obstacles. Moreover, the voxels located between the local sphere and sphere space have been mapped in advance and can be used for next step of algorithm without wasting mapping time as long as those voxels contained by sphere space.

The voxels need to be transformed from global frame to sphere frame and voxel expressed as:

$$Voxel_{(X_S,\ Y_S,\ Z_S)} = (CV) \tag{4.9}$$

where X_S, Y_S, Z_S are the Cartesian coordinate of voxel in sphere frame and can be obtained from Eq. 4.10.

$$\begin{bmatrix} X_S \\ Y_S \\ Z_S \end{bmatrix} = \begin{bmatrix} X_G - {}^{G}X_U \\ Y_G - {}^{G}Y_U \\ Z_G - {}^{G}Z_U \end{bmatrix} \tag{4.10}$$

Now, coordinates of the voxels are converted spherical coordinate system as shown in Fig. 4.4 and can be expressed as:

$$Voxel_{(r,\ \theta,\ \varphi)} = (CV) \tag{4.11}$$

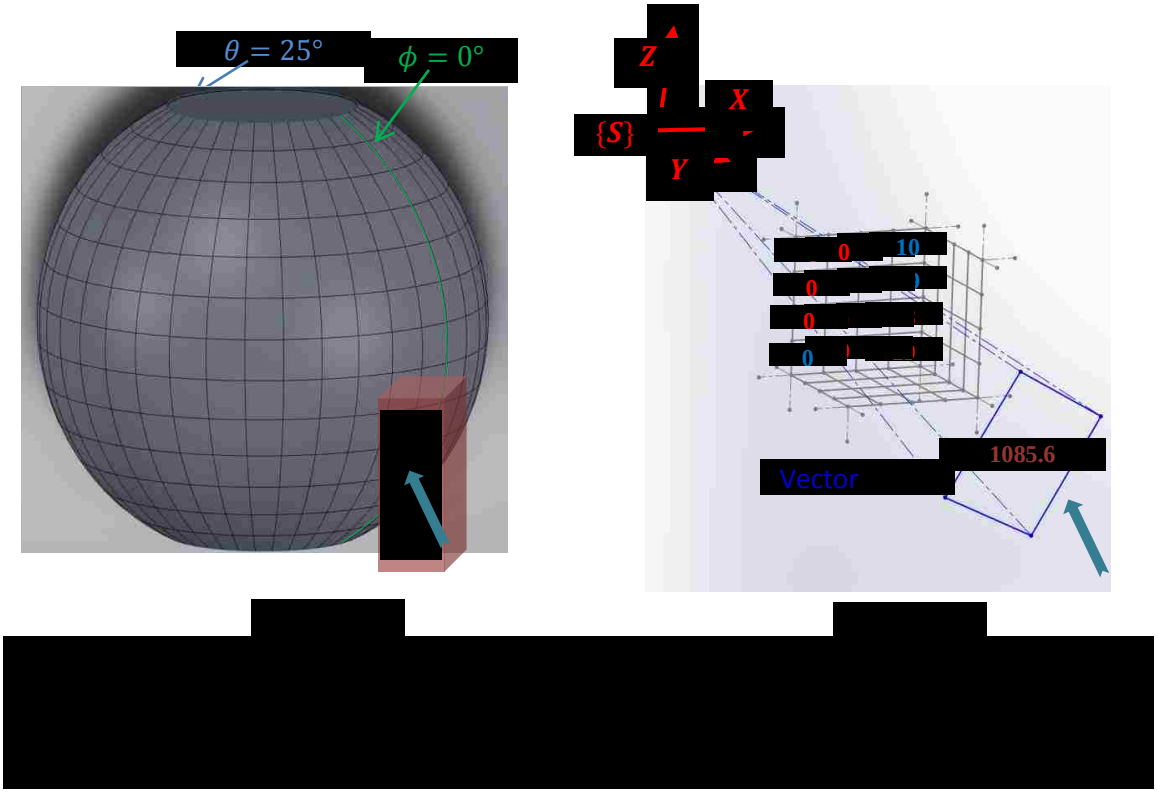
where $r,\ \theta,\ \varphi$ are the spherical coordinates of voxel and can be obtained from Eq. 4.12.

$$\begin{cases} r = \sqrt{(X_S)^2 + (Y_S)^2 + (Z_S)^2} \\ \theta = \cos^{-1}(\frac{Z_S}{r}) \\ \varphi = \tan^{-1}(\frac{Y_S}{X_S}) \end{cases} \tag{4.12}$$

Meshing the sphere space to generate vector, $Vector_{(i,\ j)}$, from $\theta = 25°$ to $\theta = 155°$ and from $\varphi = 0°$ with an interval of 10° as shown in Fig. 4.5. This interval, α, can be adjusted by either computer simulation as well as experiment. Mapping voxels into a vector space can be done by Eq. 4.13:

$$\begin{cases} i = ceil\left(\frac{\theta - 15}{\alpha}\right), & i = 1,2 \dots \dots 13 \\ j = ceil\left(\frac{\varphi}{\alpha}\right), & j = 1,2 \dots \dots 36 \\ Voxel_{(r,\ i,\ j)} = (CV) \end{cases} \tag{4.13}$$

where $i,\ j$ are the coordinates of vector in the sphere frame and $Voxel_{(r,\ i,\ j)}$ belongs to $Vector_{(i,\ j)}$ as shown in Fig. 4.5(b).



Vector magnitude, denoted as $MV_{(i,\ j)}$, need to be calculated based on all the voxels belong to vector. The $MV_{(i,\ j)}$ represents obstacle probability in the direction of vector and can be expressed as:

$$\begin{cases} MV_{(i,\ j)} = \sum {CV_{Voxel_{(r,\ i,\ j)}}}^2 * \mathrm{W} & (4.14a) \\ W = (a - b * r_{Voxel_{(r,\ i,\ j)}}) & (4.14b) \\ a - b * R = 0 & (4.14c) \end{cases}$$

where $CV_{Voxel_{(r,\ i,\ j)}}$ is the CV of $Voxel_{(r,\ i,\ j)}$ and W is the weighting function obtained from Eq. 4.14b. Square of the CV and weighting function affect sensitivity of detected obstacle so that weight of distant obstacle will be reduce and a reliable obstacle distribution can be received. $r_{Voxel_{(r,\ i,\ j)}}$ is the distance from $Voxel_{(r,\ i,\ j)}$ to sphere frame; a and b are constants satisfying Eq.4.14c; R is a sphere space radius.

4.3 Binary Mesh Representation

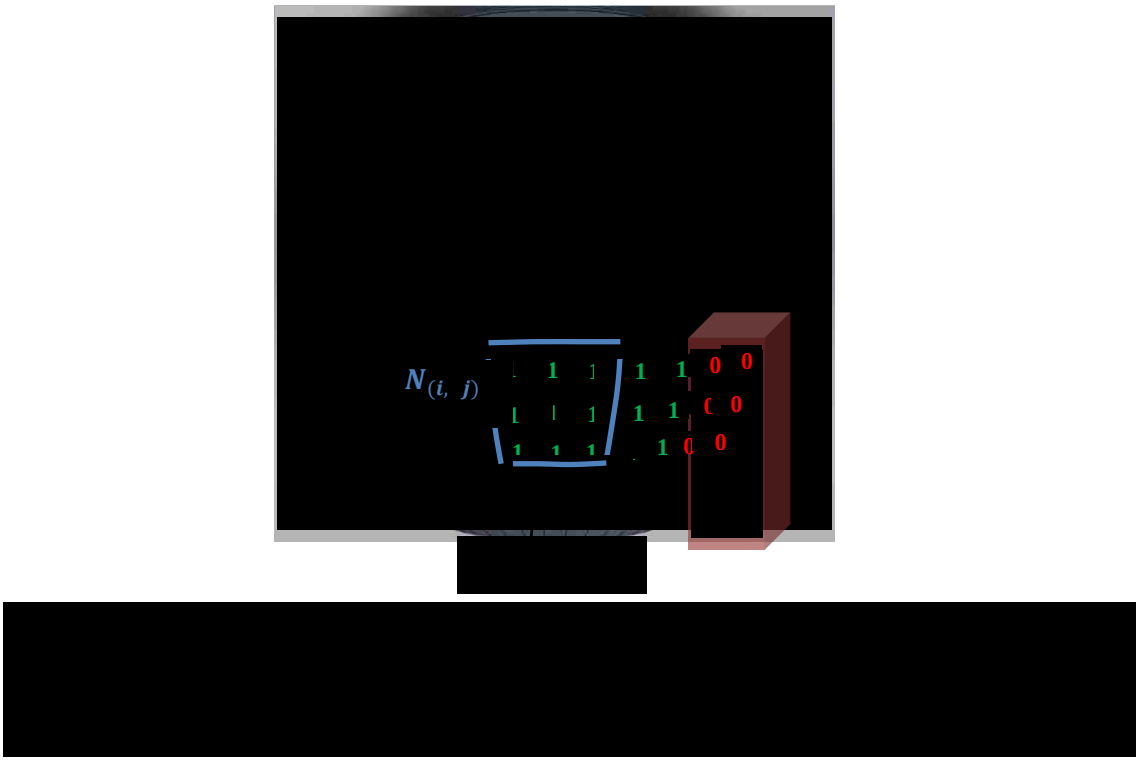
To improve efficiency of algorithm, a mesh obstacle representation, $Mesh_{(i,\ j)}$, is created to replace vector obstacle representation without changing coordinate as shown in Fig. 4.6. The binary value, denoted as $BV_{(i,\ j)}$, means the obstacle possibility in the direction of corresponding mesh. $BV_{(i,\ j)} = 1$ indicates safe, $BV_{(i,\ j)} = 0$ expresses dangerous. A threshold modified by computer simulation or experiment can be applied to determine $BV_{(i,\ j)}$ based on $MV_{(i,\ j)}$ of each vector as shown in Eq. 4.15.

$$BV_{(i,\ j)} = \begin{cases} 1, if\ MV_{(i,\ j)} < threshold \\ 0\ if\ MV_{(i,\ j)} > threshold \end{cases} \quad (4.15)$$

A global mesh matrix, *M*, and local mesh matrix, $N_{(i,\ j)}$, are defined as following:

$$M = \underbrace{\begin{bmatrix} Mesh_{(1,1)} & \cdots & Mesh_{(1,36)} \\ \vdots & \ddots & \vdots \\ Mesh_{(13,1)} & \cdots & Mesh_{(13,36)} \end{bmatrix}}_{13\times36} \tag{4.16}$$

$$N_{(i,\ j)} = \underbrace{\begin{bmatrix} Mesh_{(i-1,\ j-1)} & Mesh_{(i,\ j-1)} & Mesh_{(i+1,\ j-1)} \\ Mesh_{(i-1,\ j)} & Mesh_{(i,\ j)} & Mesh_{(i+1,\ j)} \\ Mesh_{(i-1,\ j+1)} & Mesh_{(i,\ j+1)} & Mesh_{(i+1,\ j+1)} \end{bmatrix}}_{3\times3} \tag{4.17}$$



For an arbitrary $Mesh_{(i,\ j)}$ chosen from *M*, the corresponding sub-direction in the sphere frame, denoted as ${}^{S}dir_{(i,\ j)}$, is considered desirable if following two requirements are satisfied. Firstly, $BV_{(i,\ j)}$ equals to *1*. Secondly, binary values of all the meshes from $N_{(i,\ j)}$ equal to *1*. The $dir_{(i,\ j)}$ can be expressed as:

$$\begin{cases} {}^{S}dir_{(i,\ j)} = (\theta, \varphi) \\ \theta = 25 + \alpha * (i-1) \\ \varphi = 5 + \alpha * (i-1) \end{cases} \quad (4.18)$$

where θ, φ are spherical coordinate with unit degree in the sphere frame.

Traversing the global mesh matrix to acquire entire feasible sub-directions and the one is closest to target selected to be optimum that has minimum included angle with target.

To determine which sub-direction is nearest to the target, unit vector of sub-direction ${}^{S}\boldsymbol{A}$ and target vector ${}^{S}\boldsymbol{B}$ representing in the sphere frame are created to solve included angle β_k as shown in *Fig. 4.7(a)*.

$$\begin{cases} {}^{S}\boldsymbol{A} = ({}^{S}x, {}^{S}y, {}^{S}z) \\ {}^{S}x = r\sin\theta\cos\varphi \\ {}^{S}y = r\sin\theta\sin\varphi \\ {}^{S}z = r\cos\theta \end{cases} \quad (4.19)$$

where ${}^{S}x, {}^{S}y, {}^{S}z$ are unit vector coordinates of sub-direction. θ, φ are spherical coordinate of sub-direction and *r* equal to *1* for unit vector.

$$\begin{cases} {}^{S}\boldsymbol{B} = [{}^{S}x_T, {}^{S}y_T, {}^{S}z_T] \\ {}^{S}x_T = {}^{G}x_T - {}^{G}x_U \\ {}^{S}y_T = {}^{G}y_T - {}^{G}y_U \\ {}^{S}z_T = {}^{G}z_T - {}^{G}z_U \end{cases} \quad (4.20)$$

where ${}^{S}x_T, {}^{S}y_T, {}^{S}z_T$ are coordinates of target in the sphere frame. ${}^{G}x_T, {}^{G}y_T, {}^{G}z_T$ are coordinates of target in global frame and ${}^{G}x_U, {}^{G}y_U, {}^{G}z_U$ are coordinate of UAV in global frame. The space frame located in the same position as the UAV in global.

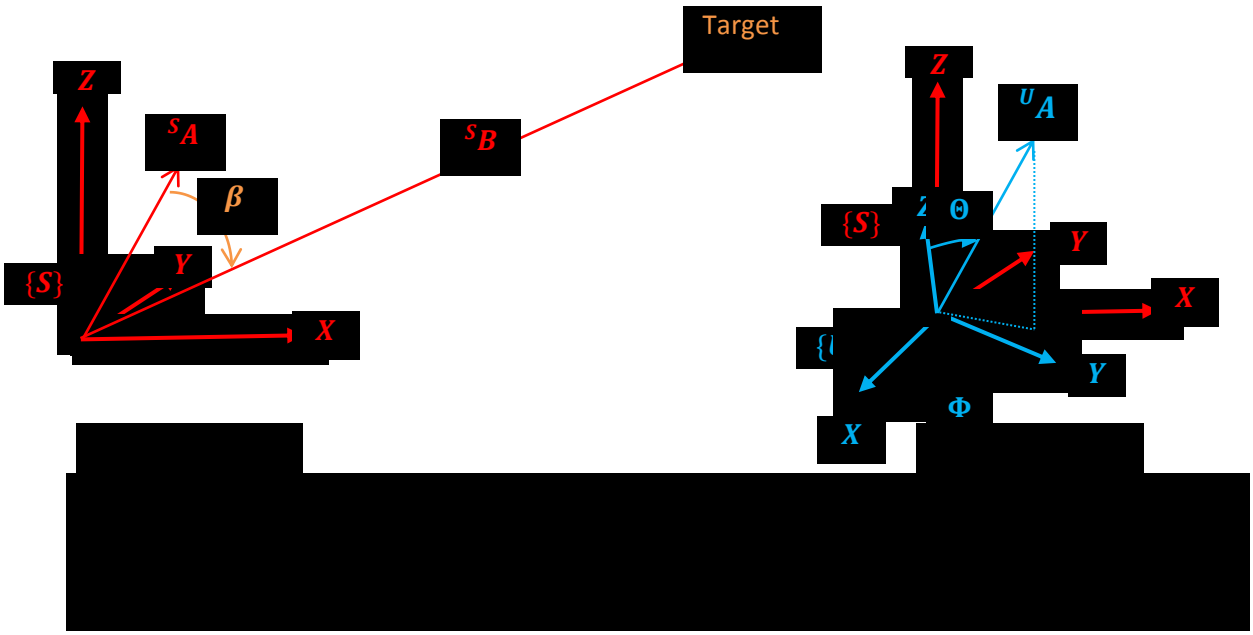
$$\beta = \cos^{-1}\left(\frac{{}^{S}\boldsymbol{A} \cdot {}^{S}\boldsymbol{B}}{|{}^{S}\boldsymbol{A}||{}^{S}\boldsymbol{B}|}\right) \quad (4.3)$$

where Eq.4.21 is solved with cosine formula between two Euclidean vectors to obtain β. Substitute unit vector of sub-directions respectively to obtain all the available β.

The sub-direction corresponding with mimimum included angle is the optimum and is represented in the UAV frame, denoted as ${}^{U}\boldsymbol{A}$ which also is an unit vector as shown in Fig. 4.7(b).

$$ {}^{U}\boldsymbol{A} = {}^{S}\boldsymbol{A} * {}^{S}_{U}\boldsymbol{R}^{-1} = [{}^{U}x, {}^{U}y, {}^{U}z] \tag{4.22} $$

where ${}^{S}\boldsymbol{A}$ is the optimal sub-direction in the sphere frame. ${}^{S}_{U}\boldsymbol{R}$ is the rotation matrix as same as ${}^{G}_{U}\boldsymbol{R}$ mentioned in Section 4.1; ${}^{U}x, {}^{U}y, {}^{U}z$ are coordinates of optimal sub-direction in the UAV frame.



The final direction, ${}^{U}dir$, in UAV frame is expressed as:

$$ \begin{cases} {}^{U}dir = (\Theta, \Phi) \\ \Theta = \cos^{-1}\left(\dfrac{{}^{U}z}{\sqrt{{}^{U}x^{2} + {}^{U}y^{2} + {}^{U}z^{2}}}\right) \\ \Phi = \tan^{-1}\left(\dfrac{{}^{U}y}{{}^{U}x}\right) \end{cases} \tag{4.23} $$

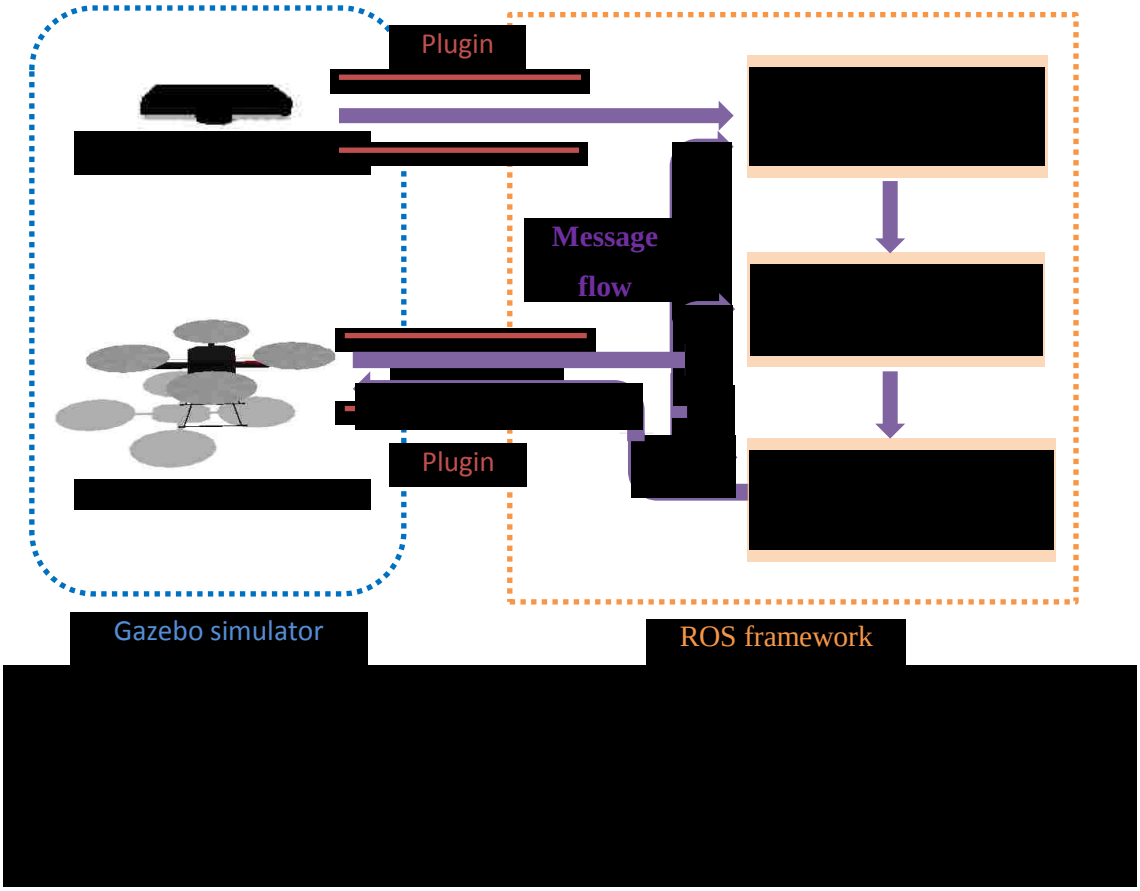
Where Θ, Φ are the spherical coordinates with unit degree in the UAV frame as shown in Fig. 4.7(b).

4.4 Simulation Environment

Simulation is the general way to test validity of algorithm before algorithm is implemented into the real robotic platform. Not only promotion of effectiveness can be done by simulation but also risk of mission taking in an unknown environment will be reduced. Gazebo open source simulator [27] integrated with the Robot Operating System (ROS) framework will be installed for simulation of 3D VM algorithm. The Gazebo accesses to high performance physics engines such as ODE [28] providing capacity of dynamics simulation. Additionally, a lot of environment models, robotics platforms as multicopter and common sensors models like Kinect sensor are available in Gazebo by contribution of worldwide researchers and scholars. In the simulation, the 3D VM algorithm will be test on a Vision-based Aerial Vehicle (VAV) system which consists with a quadrotor and a Kinect sensor. Fig. 4.8 shows an example screenshot of the

simulated kitchen environment with the VAV system. The quadrotor model was developed by Technical University of Darmstadt with a radius of 0.8 m and the Kinect sensor is assumed with a FOV (horizontal 60° and vertical 48.6°) and a sensor range of 3m. The kitchen size is 15m × 7.5m (length × width).

Fig. 4.9 illustrates system work diagram in simulation environment. The simulation ran on a Dell desktop computer running Linux operating system, Ubuntu [29]. ROS node is a process that executes computation and communicates with each other by the publisher/subscriber message mechanism. The Gazebo models can publish data to and subscribe control command from ROS through plugins which are compatible with ROS message interface.



4.5 Result and Discussion

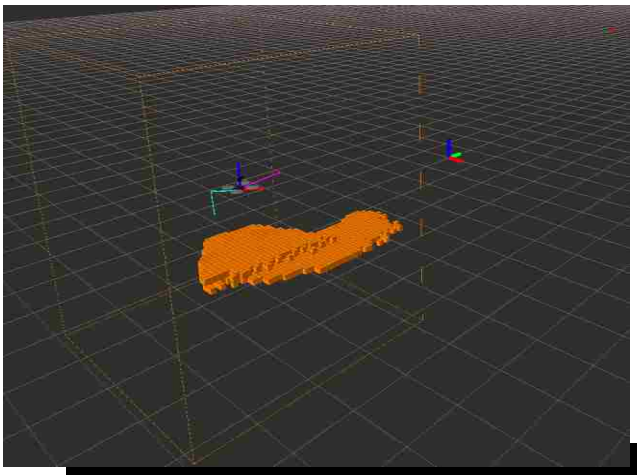
In this simulation, the static target located in the position $(2.5, 5.0, 3.0)(m)$ and the VAV located in the position $(-1.0, -7.0, 0.3)(m)$ in the global frame at the beginning. The global frame located in the middle of kitchen. As 3D VM algorithm does not deal with a low level control which is the quadrotor dynamics system, the quadrotor kinematics is generated automatically using the simulated model. The quadrotor model is controlled by the velocity command $\boldsymbol{V} = [v_x\ v_y\ v_z\ v_{roll}\ v_{pitch}\ v_{yaw}]^{-1}, \boldsymbol{V} \in \mathbb{R}^6$. However, only three of them will be used in the simulation for two situations as expressed:

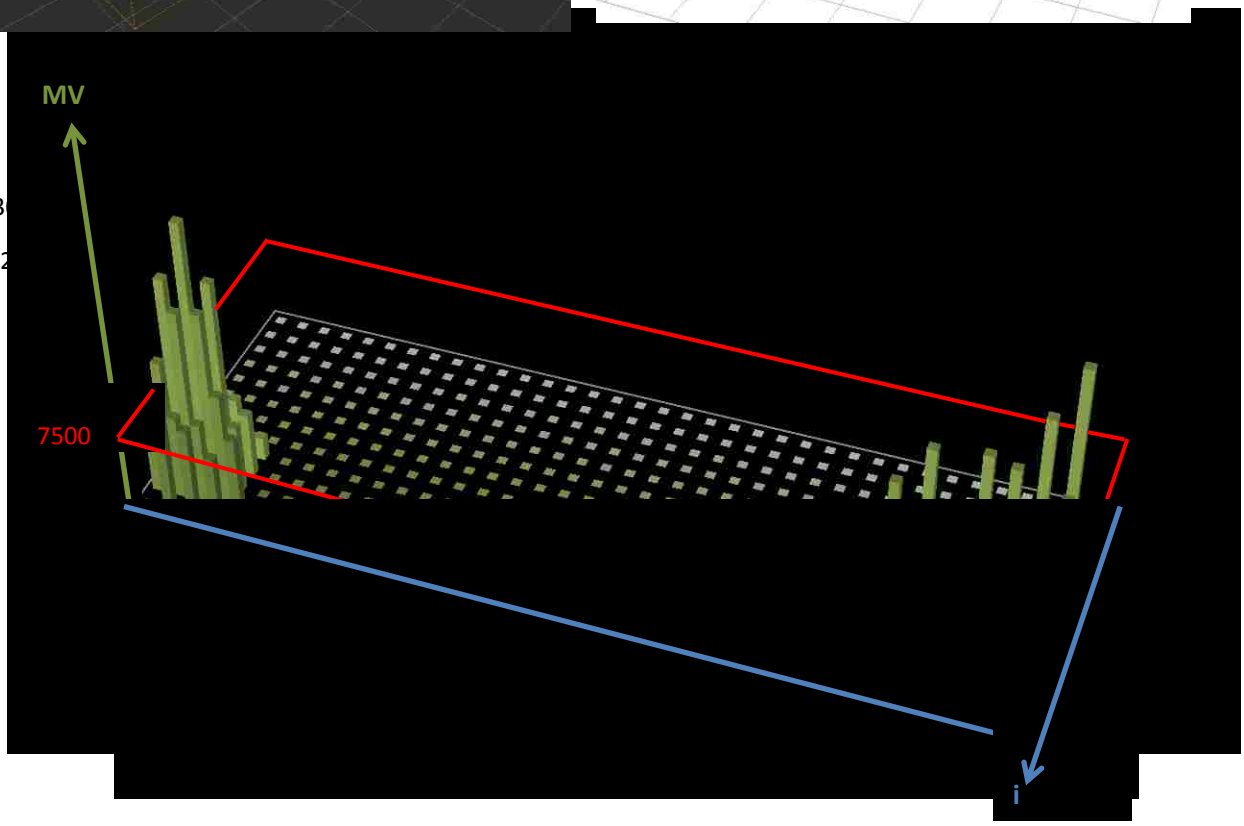
$$v_z = 0.2\left(\frac{m}{s}\right), if\ the\ VAV\ is\ under\ 0.5m \tag{4.24}$$

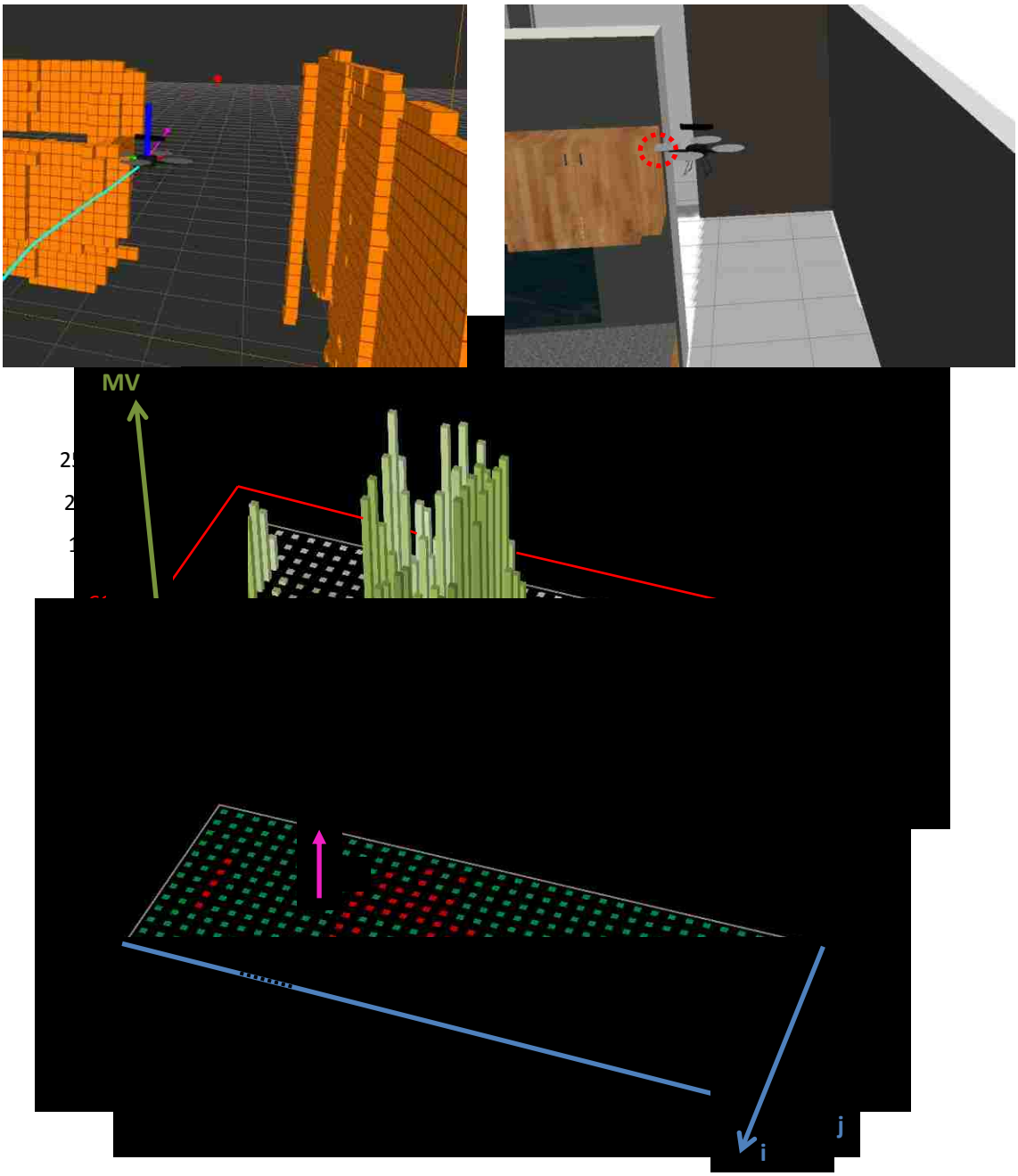
where v_z is the linear velocity along Z axis, Eq. 4.24 is the first situation that VAV rises up.

$$\begin{cases} v_x = 0.2(m/s) \\ v_z = 0.2\tan(90° - \Theta)(m/s) \\ v_{yaw} = \begin{cases} 0.2(m/s), if\ \Phi \ll 90° \\ -0.2(m/s), if\ \Phi > 90° \end{cases} \end{cases} \tag{4.25}$$

where v_x, v_z are linear velocity along X axis and Z axis respectively; v_{yaw} is angular velocity yaw movement. Θ, Φ are control commands which are the final direction in quadrotor frame. Eq. 4.25 is the second situation that is the navigation of VAV system. To determine the threshold, the simulation is started with a threshold of zero firstly as shown in Fig. 4.10. After the VAV rises up, the $MV_{(i,j)}$ of each vectors in the sphere space are displayed in Fig. 4.10(c). Two axes represent indexes of vector that are i and j in the sphere frame as described in section 4.2. After observation of $MV_{(i,j)}$ distributions, an adjustable percentage 25% is applied to multiply by the maximum $MV_{(i,j)}$in this position of the VAV to obtain approximate threshold equal to 7500.




MV
7500
i


MV
j
i

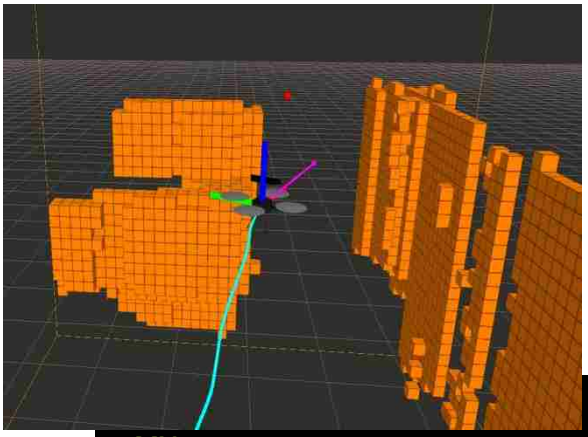
However, there is a tiny collision as shown in Fig. 4.11(b) while the VAV appearing in the corner in the simulation. Although the final direction chosen, the pink arrow as shown in Fig. 4.11(a), theoretically points to safe area, but it's not the optimal one which can avoid the obstacles successfully. Because of the discrete nature of $MV_{(i,j)}$ distributions displayed in Fig. 4.11(c), the result of direction selection is rough and did not consider the space close to obstacles.

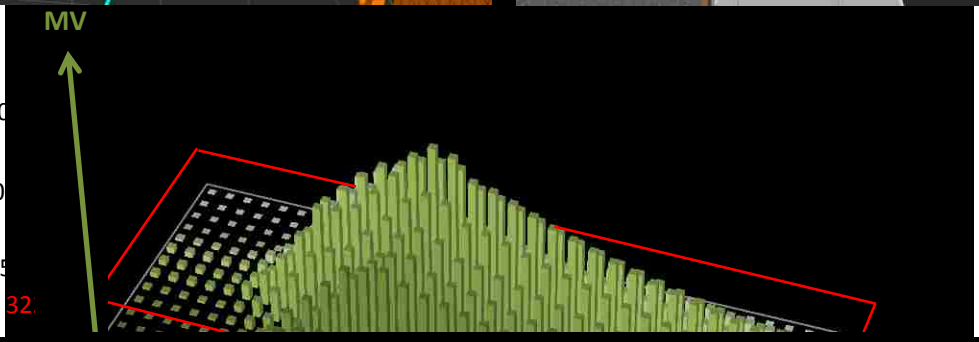
Therefore, a smoothing function is applied to obtain modified magnitude value of vector, $MV'_{(i,j)}$, as expressed as:

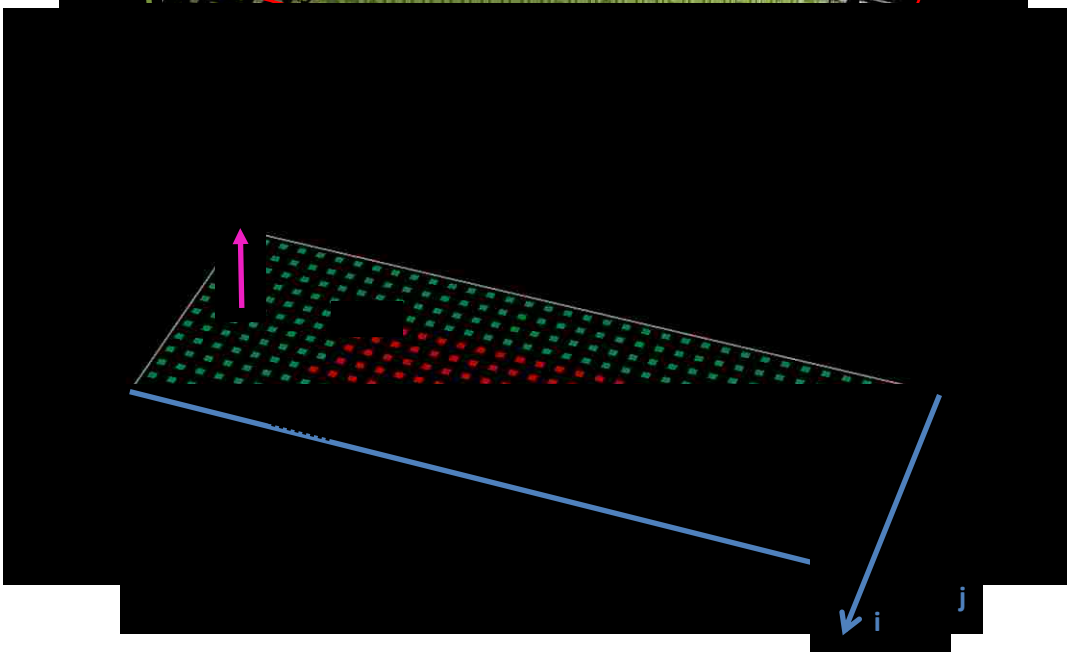
$$MV'_{(i,j)} = \frac{\mathrm{MV}_{(\mathrm{i,j-k+1})}+2\mathrm{MV}_{(\mathrm{i,j-k+2})}+\cdots+\mathrm{kMV}_{(\mathrm{i,j})}+\cdots+2\mathrm{MV}_{(\mathrm{i,j+k-2})}+\mathrm{MV}_{(\mathrm{i,j+k-1})}}{2\mathrm{k}+1} \qquad (4.26)$$

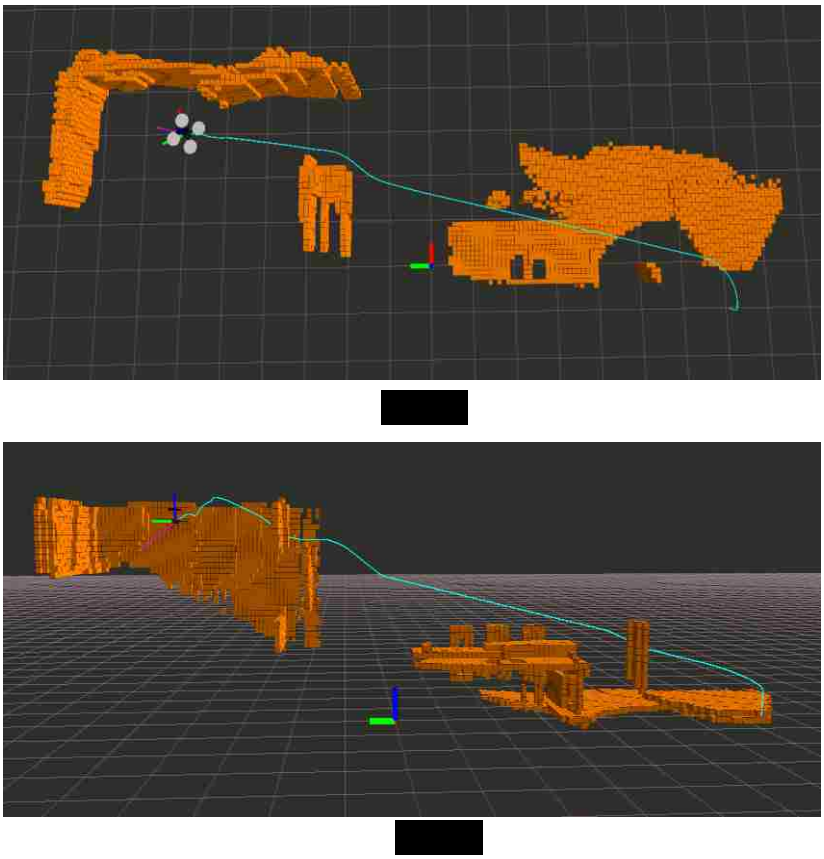
where k is adjustable constant, in this simulation it selected as 5.

In the new simulation with implementing Eq. 4.26, the VAV can avoid the obstacles in the corner as shown in Fig. 4.12. The $MV_{(i,j)}$ distributions now become much smoother than previous one as shown in Fig. 4.12(c). Then $Mesh_{(i,j)}$ representations are converted from $MV_{(i,j)}$ distributions with green and red patterns which mean safe direction and risk direction respectively as displayed in Fig. 4.12(d). Compare of this $Mesh_{(i,j)}$ representations and the previous one as shown in Fig. 4.11(d), the area enclosed in black dash line is the critical space close to obstacles and in front of VAV. The preceding simulation chooses final direction expressed with pink arrow inside of critical space resulting in a collision and the last simulation selects final direction outside of critical space leading to safe navigation.

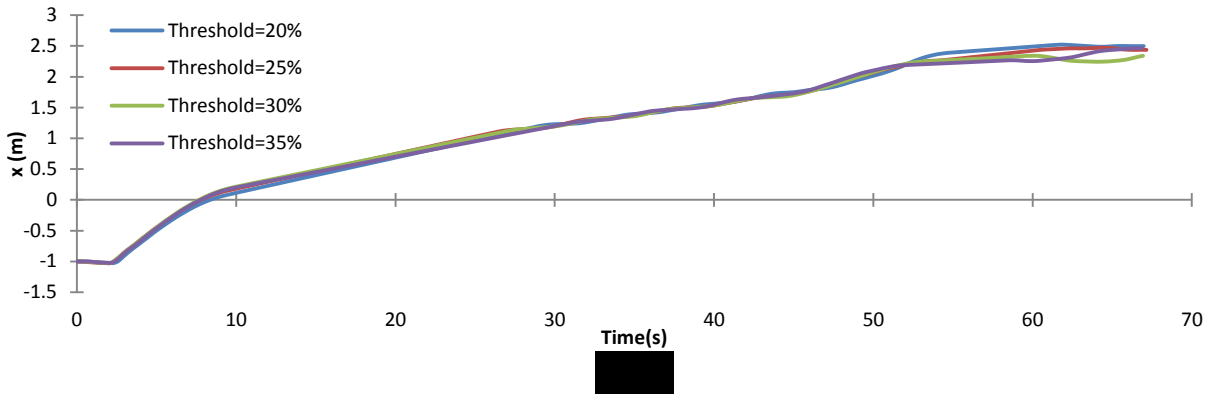



MV
150
10
32

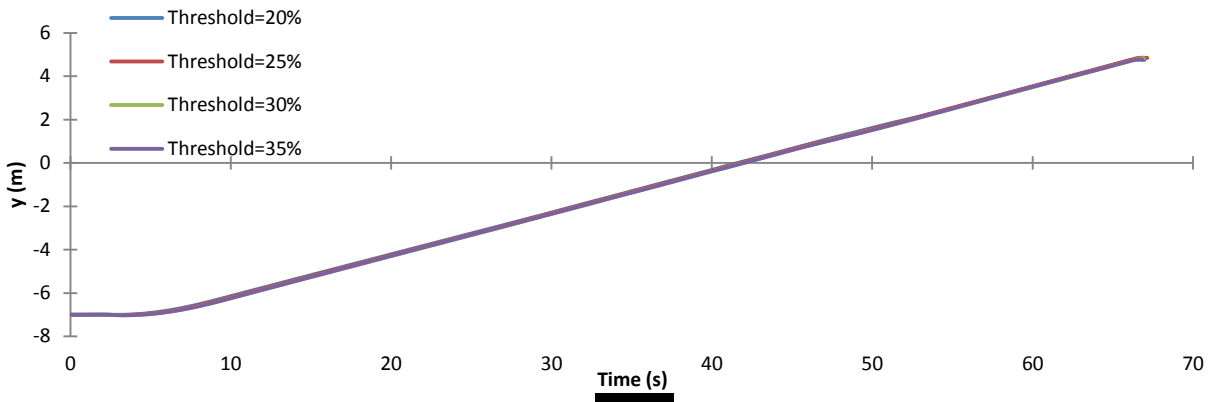

j
i



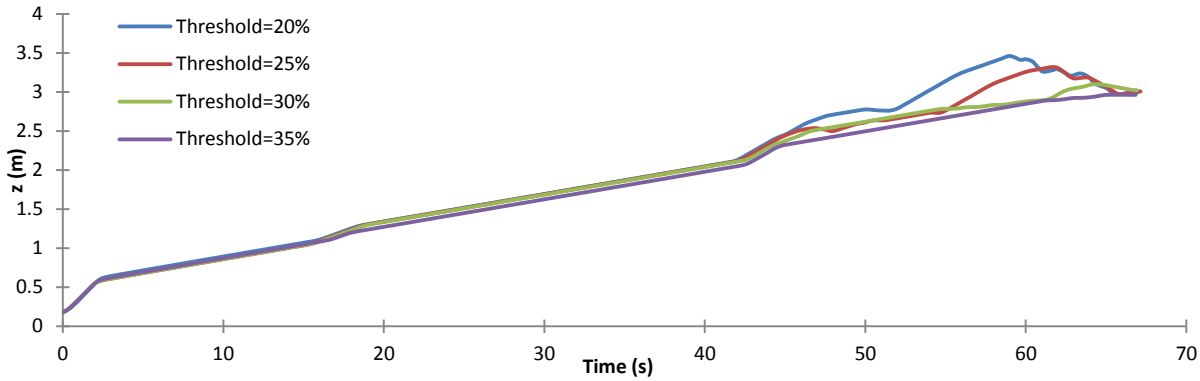
The entire path is shown in Fig. 4.13 applied a threshold equals to 20%. The VAV system can navigate in the unknown obstacles successfully controlled by this 3D VM algorithm and the flight trajectory is relatively smooth. The implementation in the real platform will be test in the future.



[redacted]

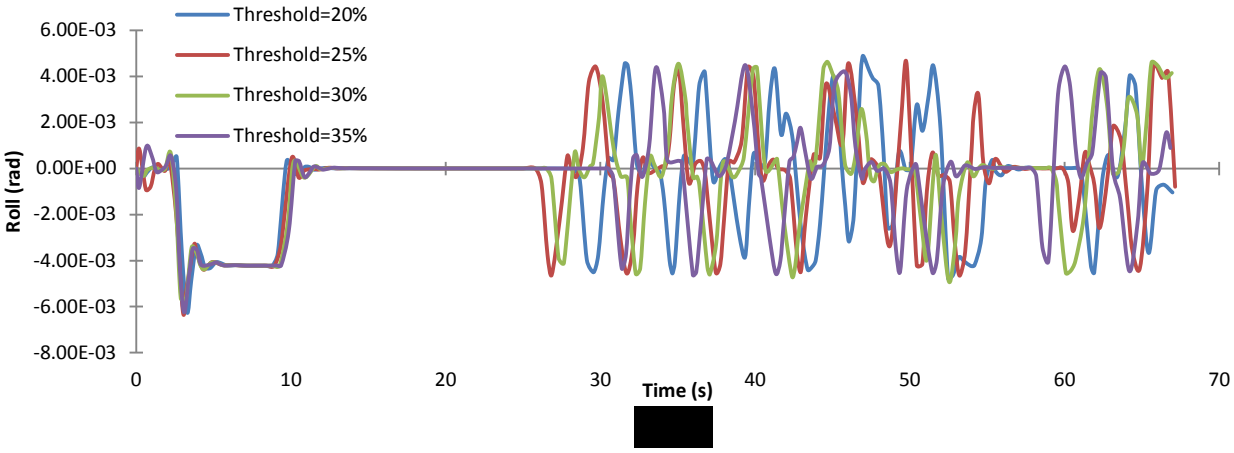


[redacted]

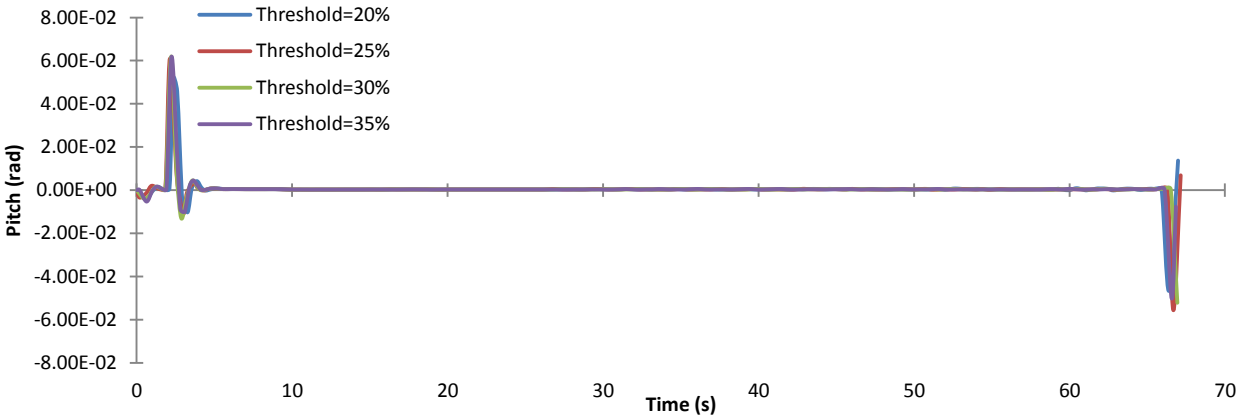


[redacted]

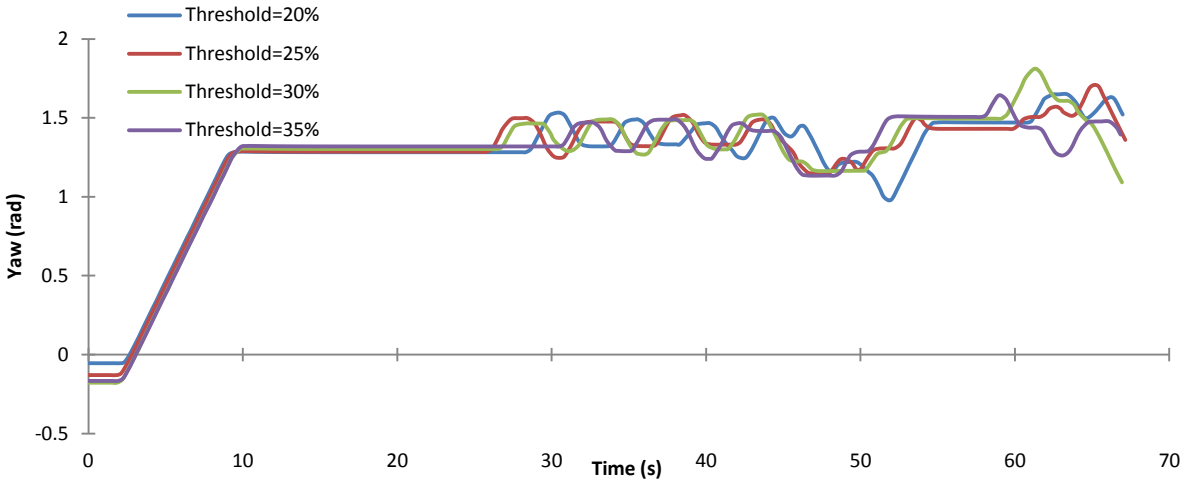
[redacted]



[redacted]



[redacted]

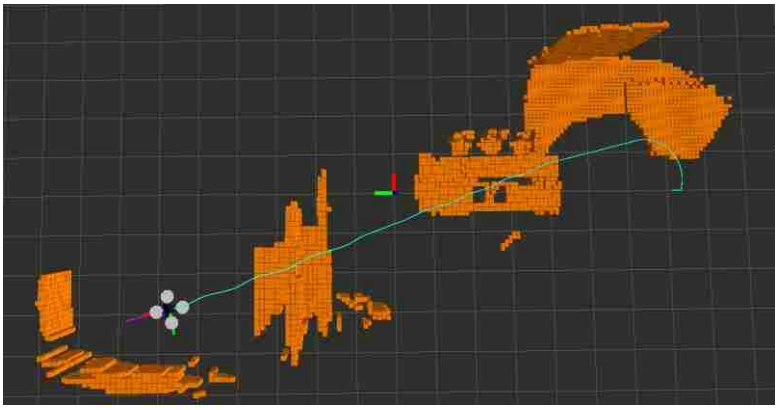


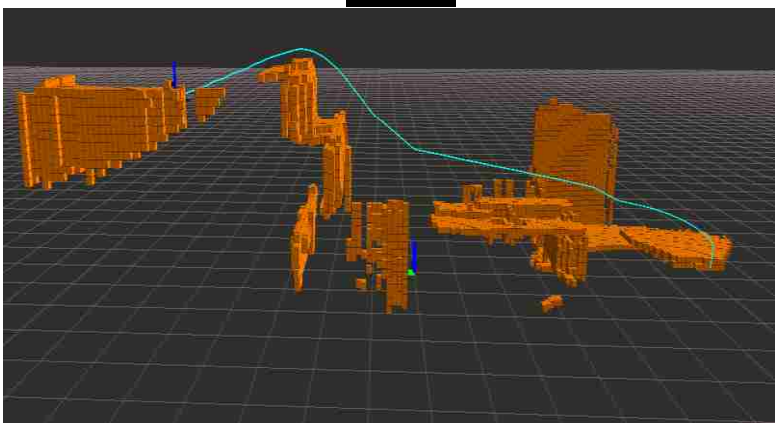
[redacted]

[redacted]

The entire positions and attitudes of VAV based on different thresholds as shown in Fig. 4.14 and Fig. 4.15 respectively. There were 4 simulations implemented according to the thresholds equal to 20%, 25%, 30% and 35% separately and the VAV successfully avoided all the obstacles in the simulations. The x position of VAV in global frame was effected by the thresholds from simulation time 50s to 70s as shown in Fig. 4.14(a). The smaller threshold value leaded to larger x position to avoid obstacles. Due to velocity along Y axis didn't consider in quadrotor model so that the y position shown in Fig. 4.14(b) didn't change with varied thresholds. In Fig. 4.14(c), the smaller threshold value made more oscillations in z position. The attitudes of VAV have similar results based on different thresholds as shown in Fig. 4.15. From simulation time 10s to 25s, there are relative smooth performances because of no huge obstacle in front of VAV. Then the VAV occurred in the corner surrounded by obstacles, yaw angle and roll angle were changing a lot. For the pitch angle as shown in Fig. 4.15(b), it changed while the VAV raised up in the initial and dropped down in the end as shown in Fig. 4.13. In Fig. 4.15(c), the yaw angle performance increased a lot at beginning because the VAV was pointed to different direction compared to target.

Also second simulation was implemented at a start global position $(0.0, -7.0, 0.3)$ and a target position $(-2.5, 5.0, 3.0)$ as shown in Fig. 4.16. The VAV still can avoid obstacles and navigate smoothly in unknown indoor environment.





4.6 Summary

This chapter presented the specific demonstration of *3D* VM algorithm with *3* sections for data reduction. In the first section discusses sensor data is discussed to convert into voxels which are stored in the global. Additionally, vector representation of obstacle is introduced in order to decrease data in created sphere space frame. Then, binary value of mesh instead of vector is manipulated to determine final direction which in UAV frame. Also, the setup of simulation environment is presented in detail. Moreover, an improved method is demonstrated to avoid collision in corner area and desirable result is obtained.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

This thesis proposes a novel method for obstacle avoidance for UAS to generate collision-free path in unknown environment assuming the attitude and location of UAS are obtained. The presented Vision-based Aerial Vehicle (VAV) system employs a Kinect senor with low cost and light weight features to detect environment and computes safe direction to navigate.

To achieve this, the 2D Vector Filed Histogram (VFH) algorithm is studied as described in Chapter 2. The VFH uses two-stage data reductions that are mapping obstacles from sensor data into two-dimensional grids and conversion of obstacles between grids to histograms. Based on the VFH algorithm, the presented Vector Mesh (VM) approach extends detection space from 2D to 3D with three data reductions including obstacle reconstructions from range data into voxels, obstacle estimations between voxels and vectors and conversion of obstacle representations from vectors into meshes as presented in Chapter 5. The 2D modified VFH algorithm was implemented in the developed Vision-based Ground Vehicle (VGV) system based on the Robot Operating System (ROS) environment described in chapter 3 so that computer simulation was also performance in the same environment with integrated Gazebo simulator presented in Chapter 5.

In summary, the VAV system applied the VM algorithm firstly had a crash in the corner because of the discrete distribution of vector obstacle estimations. Then a smooth

function was implemented to weight the distributions so that the final result that the VAV could avoid the obstacles successfully was accomplished.

5.2 Future Work

There are numerous enhancement can be built upon the presented thesis. The most important developments will be focused on three main areas: localization of VAV system and quadrotor dynamics controller.

One of the major limitations of this VM algorithm is that the localization information of quadrotor is obtained from simulator model. This part work needs to be done by using intelligent localization algorithm integrated with Inertial Measurement Unit (IMU) sensor if future experiment will be implemented in real platforms. What's more, the obstacle estimation of VM algorithm is based on attitude and global location of quadrotor. The more accurate location data is achieved, the more correct mapping of obstacles will be presented.

Another defect of this thesis is not including the quadrotor dynamics controller. A real platform such as the VGV system, described in chapter 3, with high-level control leading by ROS and low-level control managing by robotics dynamics controller can be developed for the VAV system. Due to the compatible of ROS environment, the VM algorithm code can be directly moved into onboard computer. Therefore, a fight controller module for low-level control such as Pixhaw [30] is the optional to determine locomotion of VAV.

APPENDIX A

C++ FILE FOR 3D VM ALGORITHM

A.1 Transformation of Data

A ROS node is used for transforming sensor data from Kinect frame to global frame.

```
#include "pcl_ros/point_cloud.h"
#include <pcl/point_types.h>
#include <pcl_ros/filters/filter.h>
#include <ros/ros.h>
#include <iostream>
#include <fstream>
#include <limits>
#include <tf/transform_datatypes.h>
#include <tf/LinearMath/Transform.h>
#include "pcl_ros/transforms.h"
#include <pcl_ros/impl/transforms.hpp>
#include <tf/transform_listener.h>
#include <math.h>

using namespace std;

class FilterPointcloud
{

public:
    FilterPointcloud()
    {

     sub_kinect = nh.subscribe<pcl::PointCloud<pcl::PointXYZ> > ("camera/depth/points", 1, &FilterPointcloud::callback,this);
     point_pub= nh.advertise<pcl::PointCloud<pcl::PointXYZ> >("FilteredPoints",1);
    }

    void callback(const pcl::PointCloud<pcl::PointXYZ>::ConstPtr& inputCloud );


private:
    ros::NodeHandle nh;
    ros::Subscriber sub_kinect;
    ros::Publisher point_pub;
};

void FilterPointcloud::callback(const pcl::PointCloud<pcl::PointXYZ>::ConstPtr& inputCloud )
{

   //convert PointCLoud2 to PoiintXYZ and remove NAN data
```

```
    pcl::PointCloud<pcl::PointXYZ>::Ptr tempCloud1(new pcl::PointCloud<pcl::PointXYZ>);
    std::vector<int> indices;
    pcl::removeNaNFromPointCloud(*inputCloud,*tempCloud1, indices);

   // reduce data amout
    pcl::PointCloud<pcl::PointXYZ>::Ptr simpleCloud(new pcl::PointCloud<pcl::PointXYZ>);
    pcl::PointCloud<pcl::PointXYZ>::iterator it;
    int counter = 0;
    for( it= tempCloud1->begin(); it!= tempCloud1->end(); it++)
   {
      if(counter%25==0)
      simpleCloud->push_back (pcl::PointXYZ (it->x, it->y, it->z));
      counter ++;
   }

    // transform obstacle XYZ from Kinect frame into quadrotor frame
    tf::Transform transform1;
    transform1.setOrigin( tf::Vector3(0.05,-0.02,0.2) );
    tf::Quaternion quat;
    quat.setRPY(-1.5707963, 0.0, -1.5707963);//roll,pitch,yaw angle
    transform1.setRotation(quat);
    pcl::PointCloud<pcl::PointXYZ>::Ptr tempCloud2(new pcl::PointCloud<pcl::PointXYZ>);
    pcl_ros::transformPointCloud(*simpleCloud,*tempCloud2, transform1);

   //transform obstacle XYZ from quadrotor into global frame
    tf::TransformListener listener;
    tf::StampedTransform transform2;
    tf::Transform transform3;
    tf::Quaternion Q;
    tf::Vector3 V;
    listener.waitForTransform("world", "base_link", ros::Time(0),ros::Duration(3.0));
    listener.lookupTransform("world", "base_link", ros::Time(0), transform2);
    Q=transform2.getRotation();
    V.setX(transform2.getOrigin().x());
    V.setY(transform2.getOrigin().y());
    V.setZ(transform2.getOrigin().z());
    transform3.setOrigin(V);
    transform3.setRotation(Q);

    pcl::PointCloud<pcl::PointXYZ>::Ptr tempCloud3(new pcl::PointCloud<pcl::PointXYZ>);
    pcl_ros::transformPointCloud(*tempCloud2,*tempCloud3, transform3);

    //publish filtered pointcloud
    pcl::PointCloud<pcl::PointXYZ> outputCloud;
    for( it= tempCloud3->begin(); it!= tempCloud3->end(); it++)
   {
      outputCloud.points.push_back (pcl::PointXYZ (it->x, it->y, it->z));
   }
    point_pub.publish(outputCloud);
}

int main(int argc, char **argv)
{
     ros::init(argc, argv, "filter_pointcloud_node");
     FilterPointcloud  Filterproject;
     ros::spin();
```

```
96.      return 0;
97. }
```

A.2 Mapping Voxels

A ROS is node used for mapping the data into obstacle representations of voxels

```
1.  #include "pcl_ros/point_cloud.h"
2.  #include <pcl/point_types.h>
3.  #include <pcl_ros/filters/filter.h>
4.  #include <ros/ros.h>
5.  #include <iostream>
6.  #include <fstream>
7.  #include <limits>
8.  #include <tf/transform_datatypes.h>
9.  #include <tf/LinearMath/Transform.h>
10. #include "pcl_ros/transforms.h"
11. #include <pcl_ros/impl/transforms.hpp>
12. #include <tf/transform_listener.h>
13. #include <math.h>
14. #include <map>
15. #include <string>
16. #include "hector_navigation/Map.h"
17. #include "hector_navigation/Coordinate.h"
18. #include "hector_navigation/StructKeyMap.h"
19. #include "visualization_msgs/Marker.h"
20. #include "visualization_msgs/MarkerArray.h"
21.
22. class LocalMap
23. {
24.
25. public:
26.     LocalMap()
27.     {
28.
29.      sub_filter = nh.subscribe<pcl::PointCloud<pcl::PointXYZ> > ("FilteredPoints
    ", 1, &LocalMap::callback,this);
30.      sub_map = nh.subscribe<hector_navigation::Map>("Local_Map", 1, &LocalMap::c
    allback_map,this);
31.      map_pub = nh.advertise<hector_navigation::Map>("Local_Map", 1);
32.      pub_CubeList = nh.advertise<visualization_msgs::Marker>("CubeList", 1);
33.      pub_MapOutline = nh.advertise<visualization_msgs::Marker>("MapOutline", 1);

34.    }
35.
36.     void callback_map(const hector_navigation::Map::ConstPtr& inputMap);
37.     void callback(const pcl::PointCloud<pcl::PointXYZ>::ConstPtr& inputCloud );

38.
39.     hector_navigation::Map local_map;
40.     typedef std::map<position,int> map_;
41.
42. private:
43.     ros::NodeHandle nh;
```

```
    ros::Subscriber sub_filter;
    ros::Subscriber sub_map;
    ros::Publisher map_pub;
    ros::Publisher pub_CubeList;
    ros::Publisher pub_MapOutline;
};

void LocalMap::callback_map(const hector_navigation::Map::ConstPtr& inputMap)
{

   local_map.points=inputMap->points;
   local_map.cv=inputMap ->cv;

}

void LocalMap::callback(const pcl::PointCloud<pcl::PointXYZ>::ConstPtr& inputCloud )
{
    tf::TransformListener listener;
    tf::StampedTransform transform2;
    listener.waitForTransform("world", "base_link", ros::Time(0),ros::Duration(3.0));
    listener.lookupTransform("world", "base_link", ros::Time(0), transform2);
    double x= round(transform2.getOrigin().x()/0.1);//UAV location in gloabl 3D grid frame
    double y= round(transform2.getOrigin().y()/0.1);
    double z= round(transform2.getOrigin().z()/0.1);

//visualize loacal space and voxels
    visualization_msgs::Marker  line_list;
    line_list.header.frame_id = "world";
    line_list.header.stamp = ros::Time::now();
    line_list.ns = "Outline";
    line_list.id = 1;
    line_list.type = visualization_msgs::Marker::LINE_LIST;
    line_list.action = visualization_msgs::Marker::ADD;
    line_list.pose.orientation.w = 1.0;
    line_list.scale.x = 0.005;
    line_list.color.r = 1.0f;
    line_list.color.g = 0.5f;
    line_list.color.a = 1.0;

    geometry_msgs::Point vertex;
    vertex.x=x*0.1-0.05-2.9;
    vertex.y=y*0.1-0.05-2.9;
    vertex.z=z*0.1-0.05+2.9;
    line_list.points.push_back(vertex);
    vertex.x=x*0.1-0.05-2.9;
    vertex.y=y*0.1-0.05-2.9;
    vertex.z=z*0.1-0.05-2.9;
    line_list.points.push_back(vertex);//12

    vertex.x=x*0.1-0.05-2.9;
    vertex.y=y*0.1-0.05-2.9;
    vertex.z=z*0.1-0.05-2.9;
    line_list.points.push_back(vertex);
    vertex.x=x*0.1-0.05+2.9;
    vertex.y=y*0.1-0.05-2.9;
    vertex.z=z*0.1-0.05-2.9;
           line_list.points.push_back(vertex);//23

```

```
102.            vertex.x=x*0.1-0.05+2.9;
103.            vertex.y=y*0.1-0.05-2.9;
104.            vertex.z=z*0.1-0.05-2.9;
105.            line_list.points.push_back(vertex);
106.            vertex.x=x*0.1-0.05+2.9;
107.            vertex.y=y*0.1-0.05-2.9;
108.            vertex.z=z*0.1-0.05+2.9;
109.            line_list.points.push_back(vertex);//34
110.
111.            vertex.x=x*0.1-0.05+2.9;
112.            vertex.y=y*0.1-0.05-2.9;
113.            vertex.z=z*0.1-0.05+2.9;
114.            line_list.points.push_back(vertex);
115.            vertex.x=x*0.1-0.05-2.9;
116.            vertex.y=y*0.1-0.05-2.9;
117.            vertex.z=z*0.1-0.05+2.9;
118.            line_list.points.push_back(vertex);//41
119.
120.            vertex.x=x*0.1-0.05+2.9;
121.            vertex.y=y*0.1-0.05-2.9;
122.            vertex.z=z*0.1-0.05+2.9;
123.            line_list.points.push_back(vertex);
124.            vertex.x=x*0.1-0.05+2.9;
125.            vertex.y=y*0.1-0.05+2.9;
126.            vertex.z=z*0.1-0.05+2.9;
127.            line_list.points.push_back(vertex);//48
128.
129.            vertex.x=x*0.1-0.05+2.9;
130.            vertex.y=y*0.1-0.05+2.9;
131.            vertex.z=z*0.1-0.05+2.9;
132.            line_list.points.push_back(vertex);
133.            vertex.x=x*0.1-0.05+2.9;
134.            vertex.y=y*0.1-0.05+2.9;
135.            vertex.z=z*0.1-0.05-2.9;
136.            line_list.points.push_back(vertex);//87
137.
138.            vertex.x=x*0.1-0.05+2.9;
139.            vertex.y=y*0.1-0.05+2.9;
140.            vertex.z=z*0.1-0.05-2.9;
141.            line_list.points.push_back(vertex);
142.            vertex.x=x*0.1-0.05+2.9;
143.            vertex.y=y*0.1-0.05-2.9;
144.            vertex.z=z*0.1-0.05-2.9;
145.            line_list.points.push_back(vertex);//73
146.
147.            vertex.x=x*0.1-0.05-2.9;
148.            vertex.y=y*0.1-0.05+2.9;
149.            vertex.z=z*0.1-0.05+2.9;
150.            line_list.points.push_back(vertex);
151.            vertex.x=x*0.1-0.05+2.9;
152.            vertex.y=y*0.1-0.05+2.9;
153.            vertex.z=z*0.1-0.05+2.9;
154.            line_list.points.push_back(vertex);//58
155.
156.            vertex.x=x*0.1-0.05-2.9;
157.            vertex.y=y*0.1-0.05+2.9;
158.            vertex.z=z*0.1-0.05+2.9;
159.            line_list.points.push_back(vertex);
160.            vertex.x=x*0.1-0.05-2.9;
161.            vertex.y=y*0.1-0.05+2.9;
162.            vertex.z=z*0.1-0.05-2.9;
```

```
163.            line_list.points.push_back(vertex);//56
164.
165.            vertex.x=x*0.1-0.05-2.9;
166.            vertex.y=y*0.1-0.05+2.9;
167.            vertex.z=z*0.1-0.05-2.9;
168.            line_list.points.push_back(vertex);
169.            vertex.x=x*0.1-0.05+2.9;
170.            vertex.y=y*0.1-0.05+2.9;
171.            vertex.z=z*0.1-0.05-2.9;
172.            line_list.points.push_back(vertex);//67
173.
174.            vertex.x=x*0.1-0.05-2.9;
175.            vertex.y=y*0.1-0.05+2.9;
176.            vertex.z=z*0.1-0.05+2.9;
177.            line_list.points.push_back(vertex);
178.            vertex.x=x*0.1-0.05-2.9;
179.            vertex.y=y*0.1-0.05-2.9;
180.            vertex.z=z*0.1-0.05+2.9;
181.            line_list.points.push_back(vertex);//51
182.
183.            vertex.x=x*0.1-0.05-2.9;
184.            vertex.y=y*0.1-0.05+2.9;
185.            vertex.z=z*0.1-0.05-2.9;
186.            line_list.points.push_back(vertex);
187.            vertex.x=x*0.1-0.05-2.9;
188.            vertex.y=y*0.1-0.05-2.9;
189.            vertex.z=z*0.1-0.05-2.9;
190.            line_list.points.push_back(vertex);//62
191.
192.            pub_MapOutline.publish(line_list);
193.
194.        //Delete far away COR in local_map (std::map structure) project
195.            position P1;
196.            position P2;
197.            position P3;
198.            map_ tep_map;
199.            int map_size = local_map.cv.size();
200.            for(int i=0; i<map_size; i++)
201.           {
202.                 P1.x=local_map.points[i].x;
203.                 P1.y=local_map.points[i].y;
204.                 P1.z=local_map.points[i].z;
205.                 if ((P1.x >=(x-29) && P1.x<=(x+29)) && (P1.y >=(y-
    29) && P1.y<=(y+29)) && (P1.z >=(z-29) && P1.z<=(z+29)))
206.               {
207.                tep_map[P1]=local_map.cv[i];
208.
209.               }
210.           }
211.
212.        // add new COR into local_map (std::map structure) project
213.
214.             int cloudsize = (inputCloud -> width) * (inputCloud -> height);
215.             double x_;
216.             double y_;
217.             double z_;
218.             for( int j=0; j<cloudsize; j++)
219.             {
220.
221.                x_= round((inputCloud ->points[j].x)/0.1);
222.                y_= round((inputCloud ->points[j].y)/0.1);
```

```
223.                z_= round((inputCloud ->points[j].z)/0.1);
224.
225.                if ((x_ >=(x-29) && x_<=(x+29)) && (y_ >=(y-
    29) && y_<=(y+29)) && (z_ >=(z-29) && z_<=(z+29)))
226.               {
227.                P2.x=x_;
228.                P2.y=y_;
229.                P2.z=z_;
230.
231.                    int value =tep_map[P2];
232.                    if (value <20)
233.                {
234.                  value++;
235.                }
236.                    tep_map[P2]=value;
237.
238.              }
239.            }
240.
241.         // convert local_map project into (std::vector structure) to publish
242.            visualization_msgs::Marker cube_list;
243.            cube_list.header.frame_id = "world";
244.            cube_list.header.stamp = ros::Time::now();
245.            cube_list.ns = "Cubes";
246.            cube_list.id = 2;
247.            cube_list.type = visualization_msgs::Marker::CUBE_LIST;
248.            cube_list.action = visualization_msgs::Marker::ADD;
249.            cube_list.pose.orientation.w = 1.0;
250.            cube_list.scale.x = 0.1f;
251.            cube_list.scale.y = 0.1f;
252.            cube_list.scale.z = 0.1f;
253.            cube_list.color.r = 1.0f;
254.            cube_list.color.g = 0.5f;
255.            cube_list.color.a = 1.0;
256.
257.             hector_navigation::Coordinate coord;
258.             map_::iterator iter;
259.             for (iter = tep_map.begin(); iter != tep_map.end();++iter)
260.             {
261.               P3=  iter->first;
262.               coord.x=P3.x;
263.               coord.y=P3.y;
264.               coord.z=P3.z;
265.               if ((iter->second)==20)
266.               {
267.                   geometry_msgs::Point temp;
268.                   temp.x = coord.x*0.1-0.05;
269.                   temp.y = coord.y*0.1-0.05;
270.                   temp.z = coord.z*0.1-0.05;
271.                   cube_list.points.push_back(temp);
272.           }
273.
274.               local_map.points.push_back(coord);
275.               local_map.cv.push_back(iter->second);
276.             }
277.
278.             pub_CubeList.publish(cube_list);
279.             local_map.header.stamp=ros::Time::now();
280.             map_pub.publish(local_map);
281.       }
282.
```

```
283.       int main(int argc, char **argv)
284.       {
285.            ros::init(argc, argv, "local_map_node");
286.
287.            LocalMap  localproject;
288.
289.            ros::spin();
290.
291.            return 0;
292.       }
```

A.3 Mapping Vectors

A ROS is node used for converting voxels into vectors.

```
1.  #include <ros/ros.h>
2.  #include <iostream>
3.  #include <fstream>
4.  #include <limits>
5.  #include <tf/transform_datatypes.h>
6.  #include <tf/LinearMath/Transform.h>
7.  #include <tf/transform_listener.h>
8.  #include <math.h>
9.  #include <map>
10. #include <vector>
11. #include <string>
12. #include "hector_navigation/Map.h"
13. #include "hector_navigation/Coordinate.h"
14. #include "hector_navigation/StructKeyMap.h"
15. #include "hector_navigation/Mesh.h"
16. #include "hector_navigation/Row.h"
17. #include "hector_navigation/Size.h"
18.
19. #define PI 3.14159265
20.
21. class SphereMap
22. {
23.
24. public:
25.     SphereMap()
26.     {
27.
28.      sub_map = nh.subscribe<hector_navigation::Map>("Local_Map", 1, &SphereMap::
    callback_map,this);
29.      mesh_pub = nh.advertise<hector_navigation::Mesh>("Sphere_Mesh", 1);
30.     }
31.
32.     void callback_map(const hector_navigation::Map::ConstPtr& inputMap);
33.
34.     typedef std::map<sphere,int> sphere_;//[(theta,phi,r),(cv)]
35.     typedef std::map<Vector,double> vector_;//[(THETA,PHI),(mv)]
36.     typedef std::vector<Vector>   direction;
37.
38. private:
```

```
39.     ros::NodeHandle nh;
40.     ros::Subscriber sub_map;
41.     ros::Publisher mesh_pub;
42. };
43. 
44. void SphereMap::callback_map(const hector_navigation::Map::ConstPtr& inputMap)
45. {
46.     position goal;
47.     goal.x=3;
48.     goal.y=0;
49.     goal.z=1;
50. 
51.     tf::TransformListener listener;
52.     tf::StampedTransform transform2;
53.     listener.waitForTransform("world", "base_link", ros::Time(0),ros::Duration(3
    .0));
54.     listener.lookupTransform("world", "base_link", ros::Time(0), transform2);
55.     double x= round(transform2.getOrigin().x()/0.1);//UAV location in gloabl 3D
    grid frame
56.     double y= round(transform2.getOrigin().y()/0.1);
57.     double z= round(transform2.getOrigin().z()/0.1);
58. 
59. //convert local_map into sphere_map
60. 
61.     hector_navigation::Map local_map;
62.     sphere_ sphere_map;//[theta,phi,r] --[cv]
63.     sphere S_point;//[theta,phi,r]
64.     double angle1,angle2,phi_,dis, theta_,x_,y_,z_;
65.     int cv_;
66. 
67.     local_map.points=inputMap->points;
68.     local_map.cv=inputMap ->cv;
69. 
70.     for (int i = 0; i<local_map.cv.size(); i++)
71.    {
72.         x_ =local_map.points[i].x;
73.         y_ =local_map.points[i].y;
74.         z_ =local_map.points[i].z;
75.             cv_=local_map.cv[i];
76. 
77.             dis=sqrt(pow(x_-x,2.0)+pow(y_-y,2.0)+pow(z_-z,2.0));
78. 
79.         if (dis<=18.5)//local_map to sphere space, 70% of sensor range
80.            {
81.                     //3d grid to sphere --theta
82.         angle2 =acos((z_-z)/sqrt(pow(x_-x,2.0)+pow(y_-y,2.0)+pow(z_-z,2.0)));
83.             theta_=angle2*180/PI;
84. 
85.                  if (theta_ >25.0 && theta_ <=155.0)
86.           {
87.           S_point.Theta=ceil((theta_-25)/10);
88.           }
89. 
90.                     //3d grid to sphere --phi
91.                angle1 = atan2(y_-y,x_-x);
92.                if (angle1<=0  )
93.           {
94.           phi_=ceil((angle1*180/PI+360)/10);
95.           }
96.                else
97.           {
```

```
98.           phi_=ceil(angle1*180/PI/10);
99.           }
100.                        S_point.Phi=phi_;
101.
102.                             //3d grid to sphere --R
103.                        S_point.R=dis;
104.
105.                    sphere_map[S_point]=cv_;
106.
107.                    }
108.             }
109.
110.        //calcul Vector MV
111.
112.           vector_ vector_mv;//[THETA,PHI]--[MV]
113.           sphere_::iterator iter;
114.           Vector Index;//[THETA,PHI]
115.           sphere S_point1;//[theta,phi,r]
116.           for(iter=sphere_map.begin(); iter != sphere_map.end(); ++iter)
117.        {
118.             S_point1=iter -> first;
119.             Index.PHI=S_point1.Phi;
120.             Index.THETA=S_point1.Theta;
121.
122.             double value=vector_mv[Index];
123.             value=value+pow((iter -> second),2.0)*(10-0.5*S_point1.R);
124.             vector_mv[Index]=value;
125.        }
126.
127.        //smooth funcation appiled to  mv
128.        Vector index1, index2,index3,index4,index5,index6,index7,index8,index9;

129.
130.        for (int j=1; j<37;j++)
131.        {
132.          for(int k=1; k<14;k++)
133.           {
134.             index1.PHI=j-4;
135.             index1.THETA=k;
136.             index2.PHI=j-3;
137.             index2.THETA=k;
138.             index3.PHI=j-2;
139.             index3.THETA=k;
140.             index4.PHI=j-1;
141.             index4.THETA=k;
142.             index5.PHI=j;
143.             index5.THETA=k;
144.             index6.PHI=j+1;
145.             index6.THETA=k;
146.             index7.PHI=j+2;
147.             index7.THETA=k;
148.             index8.PHI=j+3;
149.             index8.THETA=k;
150.             index9.PHI=j+4;
151.             index9.THETA=k;
152.
153.             vector_mv[index5]=(vector_mv[index1]*1+vector_mv[index2]*2+vector_m
    v[index3]*3+vector_mv[index4]*4+vector_mv[index5]*5+vector_mv[index6]*4+vector_m
    v[index7]*3+vector_mv[index8]*2+vector_mv[index9]*1)/11;
154.           }
155.        }
```

```
156.
157.       //2D array structure store binary value
158.       hector_navigation::Row ROW;
159.       hector_navigation::Mesh MESH;
160.       Vector index_mesh;
161.       int mesh[13][36];
162.       for (int j=1; j<14;j++)
163.       {
164.         for(int k=1; k<37;k++)
165.          {
166.
167.           index_mesh.PHI=k;
168.           index_mesh.THETA=j;
169.           if (vector_mv[index_mesh]>7500.0)
170.           {
171.            mesh[j-1][k-1]=0;
172.           }
173.           else
174.           {
175.           mesh[j-1][k-1]=1;
176.           }
177.
178.           ROW.row.push_back(mesh[j-1][k-1]);
179.                file11<<vector_mv[index_mesh]<<" ";
180.           }
181.
182.        MESH.column.push_back(ROW);
183.        ROW.row.clear();
184.
185.       }
186.
187.       MESH.header.stamp=ros::Time::now();
188.       mesh_pub.publish(MESH);
189.
190.       }
191.
192.       int main(int argc, char **argv)
193.       {
194.            ros::init(argc, argv, "sphere_map_node");
195.            SphereMap  sphereproject;
196.            ros::spin();
197.            return 0;
198.       }
```

A.4 Converting Meshes and Direction Selection

A ROS is node used for converting vectors into mesh and determining optimal direction.

```
1.  #include <ros/ros.h>
2.  #include <iostream>
3.  #include <fstream>
4.  #include <limits>
5.  #include <tf/transform_datatypes.h>
6.  #include <tf/LinearMath/Transform.h>
```

```
7.  #include <tf/transform_listener.h>
8.  #include <math.h>
9.  #include <vector>
10. #include <string>
11. #include "hector_navigation/StructKeyMap.h"
12. #include <geometry_msgs/Twist.h>
13. #include <geometry_msgs/Pose.h>
14. #include <algorithm>
15. #include "hector_navigation/Mesh.h"
16. #include "hector_navigation/Row.h"
17. #include "hector_navigation/Size.h"
18. #include <visualization_msgs/Marker.h>
19. #include "visualization_msgs/MarkerArray.h"
20.
21. #define PI 3.14159265
22.
23. class ControlCommand
24. {
25.
26. public:
27.     ControlCommand()
28.     {
29.
30.      sub_mesh = nh.subscribe<hector_navigation::Mesh>("Sphere_Mesh", 1, &Control
    Command::callback_mesh,this);
31.      pub_vel = nh.advertise<geometry_msgs::Twist>("cmd_vel", 1);
32.      pub_LineStrip = nh.advertise<visualization_msgs::Marker>("LineStrip", 1);
33.      pub_Goalsphere = nh.advertise<visualization_msgs::Marker>("Goalsphere", 1);

34.      pub_Arrow= nh.advertise<visualization_msgs::Marker>("Arrow",1);
35.     }
36.
37.     void callback_mesh(const hector_navigation::Mesh::ConstPtr& inputMesh);
38.
39.     typedef std::vector<Vector>   direction;
40.     geometry_msgs::Twist moveCommand;
41.     visualization_msgs::Marker  line_strip;
42.
43. private:
44.     ros::NodeHandle nh;
45.     ros::Subscriber sub_mesh;
46.     ros::Publisher pub_vel;
47.     ros::Publisher pub_LineStrip;
48.     ros::Publisher pub_Goalsphere;
49.     ros::Publisher pub_Arrow;
50. };
51.
52. void ControlCommand::callback_mesh(const hector_navigation::Mesh::ConstPtr& inpu
    tMesh)
53. {
54.     position goal;
55.     goal.x=2.5;
56.     goal.y=5.0;
57.     goal.z=3.0;
58.
59. //PUBLISH Goal in RVIZ
60.     visualization_msgs::Marker  dummy_goal;
61.     dummy_goal.header.frame_id = "world";
62.     dummy_goal.header.stamp = ros::Time::now();
63.     dummy_goal.ns = "Dummy_Goal";
64.     dummy_goal.id = 4;
```

```
65.     dummy_goal.type = visualization_msgs::Marker::SPHERE;
66.     dummy_goal.action = visualization_msgs::Marker::ADD;
67.     dummy_goal.pose.orientation.w = 1.0;
68.     dummy_goal.scale.x = 0.1;
69.     dummy_goal.scale.y = 0.1;
70.     dummy_goal.scale.z = 0.1;
71.     dummy_goal.color.r = 1.0f;
72.     dummy_goal.color.a = 1.0;
73.     dummy_goal.pose.position.x = goal.x;
74.     dummy_goal.pose.position.y = goal.y;
75.     dummy_goal.pose.position.z = goal.z;
76.     dummy_goal.lifetime = ros::Duration();
77.     pub_Goalsphere.publish(dummy_goal);
78.
79.     tf::TransformListener listener;
80.     tf::StampedTransform transform2;
81.     tf::Transform transform3;
82.     tf::Quaternion Q;
83.     tf::Vector3 V;
84.     listener.waitForTransform("world", "base_link", ros::Time(0),ros::Duration(3
    .0));
85.     listener.lookupTransform("world", "base_link", ros::Time(0), transform2);
86.     double x= transform2.getOrigin().x();//UAV location in gloabl
87.     double y= transform2.getOrigin().y();
88.     double z= transform2.getOrigin().z();
89.
90.     Q=transform2.getRotation();
91.     V.setX(0);
92.     V.setY(0);
93.     V.setZ(0);
94.     transform3.setOrigin(V);
95.     transform3.setRotation(Q.inverse ());
96.
97. //PUBLISH path in RVIZ
98.
99.     line_strip.header.frame_id = "world";
100.            line_strip.header.stamp = ros::Time::now();
101.            line_strip.ns = "Lines";
102.            line_strip.id = 0;
103.            line_strip.type = visualization_msgs::Marker::LINE_STRIP;
104.            line_strip.action = visualization_msgs::Marker::ADD;
105.            line_strip.pose.orientation.w = 1.0;
106.            line_strip.scale.x = 0.02;
107.            line_strip.color.g = 1.0f;
108.            line_strip.color.b = 1.0f;
109.            line_strip.color.a = 1.0;
110.
111.                geometry_msgs::Point p;
112.            p.x=x;
113.            p.y=y;
114.            p.z=z;
115.            line_strip.points.push_back(p);
116.            pub_LineStrip.publish(line_strip);
117.
118.        //STEERING CONTROL
119.                hector_navigation::Mesh MESH;
120.                MESH.column= inputMesh ->column;
121.
122.
123.                int mesh[13][36];
124.                for (int j=0; j<13;j++)
```

```
125.               {
126.                 for(int k=0; k<36;k++)
127.                  {
128.                mesh[j][k]=MESH.column[j].row[k];
129.
130.                  }
131.               }
132.
133.           bool is_ok=0;//pick up safe sub_directions
134.           direction dire_vector;
135.           Vector dire_index;
136.           dire_vector.clear();
137.
138.           for (int j=1; j<12;j++)
139.           {
140.             for(int k=0; k<36;k++)
141.             {
142.
143.               if (mesh[j][k]== 1)
144.               {
145.                   if (k==0)
146.                   {
147.                      if (((mesh[j-1][35]==1 && mesh[j-1][k]==1)&& (mesh[j-
    1][k+1]==1&&mesh[j][35]==1))
148.             &&((mesh[j][k+1]==1 && mesh[j+1][35]==1)&& (mesh[j+1][k]==1&&mesh[j
    +1][k+1]==1))  )
149.                           {
150.                                              is_ok=1;
151.                             }
152.
153.                   }
154.                   else if (k==35)
155.                   {
156.                      if (((mesh[j-1][k-1]==1 && mesh[j-1][k]==1)&& (mesh[j-
    1][0]==1&&mesh[j][k-1]==1))
157.           &&((mesh[j][0]==1 && mesh[j+1][k-
    1]==1)&& (mesh[j+1][k]==1&&mesh[j+1][0]==1))  )
158.
159.                           {
160.                                              is_ok=1;
161.                           }
162.
163.                   }
164.                   else
165.                   {
166.                      if (((mesh[j-1][k-1]==1 && mesh[j-1][k]==1)&& (mesh[j-
    1][k+1]==1&&mesh[j][k-1]==1))
167.           &&((mesh[j][k+1]==1 && mesh[j+1][k-
    1]==1)&& (mesh[j+1][k]==1&&mesh[j+1][k+1]==1))  )
168.
169.                           {
170.                                              is_ok=1;
171.                           }
172.                    }
173.                    if (is_ok==1)
174.                   {
175.                     dire_index.THETA=(10*j+30)*PI/180;
176.                     dire_index.PHI=(10*k+5)*PI/180;
177.                     dire_vector.push_back(dire_index);
178.
179.                   }
```

```
180.
181.                     is_ok=0;
182.                  }
183.               }
184.
185.           }
186.       //choose the closest direction
187.           direction::iterator IT;
188.           vector<double> delta;
189.           double Dot_product;
190.           double magnitude;
191.           double Angle;
192.           for(IT=dire_vector.begin(); IT != dire_vector.end(); ++IT)
193.           {
194.                dire_index =*IT;
195.                Dot_product=(goal.x-
    x)*sin(dire_index.THETA)*cos(dire_index.PHI)+(goal.y-
    y)*sin(dire_index.THETA)*sin(dire_index.PHI)+(goal.z-z)*cos(dire_index.THETA);
196.                magnitude=sqrt(pow(goal.x-x,2.0)+pow(goal.y-y,2.0)+pow(goal.z-
    z,2.0));
197.                Angle=acos(Dot_product/magnitude);
198.                delta.push_back(Angle);
199.           }
200.       //transform direction to quadrotor frame and steer quadrotor which way t
    o rotate
201.           int min_index=min_element(delta.begin(),delta.end())-
     delta.begin();
202.
203.           double dir_theta=dire_vector[min_index].THETA;
204.           double dir_phi=dire_vector[min_index].PHI;
205.               tf::Vector3 S_dir;
206.           S_dir.setX(sin(dir_theta)*cos(dir_phi));
207.           S_dir.setY(sin(dir_theta)*sin(dir_phi));
208.           S_dir.setZ(cos(dir_theta));
209.               tf::Vector3 U_dir=transform3*S_dir;
210.
211.           double dir_THETA=acos(U_dir.getZ()/sqrt(pow(U_dir.getX(),2.0)+pow(U_
    dir.getY(),2.0)+pow(U_dir.getZ(),2.0)));
212.           double dir_PHI=atan2(U_dir.getY(),U_dir.getX());
213.
214.       //Pubslih direction in RVIZ
215.               visualization_msgs::Marker arrow;
216.           arrow.header.frame_id = "base_link";
217.           arrow.header.stamp = ros::Time::now();
218.           arrow.ns = "Directions";
219.           arrow.id = 5;
220.           arrow.type = visualization_msgs::Marker::ARROW;
221.           arrow.action = visualization_msgs::Marker::ADD;
222.           arrow.pose.orientation.w = 1.0;
223.           arrow.scale.x = 0.02;
224.           arrow.scale.y=0.05;
225.           arrow.scale.z=0.05;
226.           arrow.color.r=1.0;
227.           arrow.color.g = 0;
228.           arrow.color.b = 1.0;
229.           arrow.color.a = 1.0;
230.
231.               geometry_msgs::Point p1;
232.           p1.x=0;
233.           p1.y=0;
234.           p1.z=0;
```

```
235.            arrow.points.push_back(p1);
236.                geometry_msgs::Point p2;
237.            p2.x=U_dir.getX();
238.            p2.y=U_dir.getY();
239.            p2.z=U_dir.getZ();
240.            arrow.points.push_back(p2);
241.            pub_Arrow.publish(arrow);
242.
243.            int rotate;
244.            double error=0.087266;
245.            if (dir_PHI >= error )
246.            {
247.                rotate=1;
248.            }
249.            else if (dir_PHI <= -error )
250.            {
251.                rotate=-1;
252.            }
253.            else
254.            {
255.                rotate=0;
256.            }
257.
258.            if (abs(x-goal.x)<=0.3 && abs(y-goal.y)<=0.3  && abs(z-
    goal.z)<=0.3 )
259.            {
260.                moveCommand.linear.x = 0.0;
261.                moveCommand.linear.z = 0.0;
262.                moveCommand.angular.z = -0.2;
263.                        pub_vel.publish(moveCommand);
264.                ROS_INFO_STREAM("Arrive goal !!");
265.            }
266.            else
267.            {
268.                if(z <0.5)//rise to 0.5m
269.                {
270.                   moveCommand.linear.x = 0;
271.                   moveCommand.linear.z = 0.2;
272.                   moveCommand.angular.z = 0;
273.                   pub_vel.publish(moveCommand);
274.                   ROS_INFO_STREAM("Rise now !");
275.                }
276.                else
277.                {
278.                    moveCommand.linear.x = 0.2;
279.                    moveCommand.linear.z = 0.2*tan(PI/2-dir_THETA);
280.                    moveCommand.angular.z = 0.2*rotate;
281.                    pub_vel.publish(moveCommand);
282.                }
283.            }
284.
285.        }
286.
287.        int main(int argc, char **argv)
288.        {
289.             ros::init(argc, argv, "command_node");
290.             ControlCommand  controlproject;
291.             ros::spin();
292.             return 0;
293.        }
```

BIBLIOGRAPHY

[1] Anonymous," Unmanned Aircraft Systems Roadmap 2005– 2030", Office of the Secretary of Defense, Washington, DC, USA, 2005.

[2] Defense Update, "MQ-9 Reaper Hunter/Killer UAV". Retrieved from http://defense-update.com/products/p/predatorB.htm on 13 April 2015.

[3] Samuel Gibbs, "Google's Titan drones to take flight within months", The Guardian, Retrieved from http://www.theguardian.com/technology/2015/mar/03/googles-titan-drones-to-take-flight-within-months on 14 April 2015.

[4] Wikipedia, "Boeing A160 Hummingbird", Retrieved from https://en.wikipedia.org/wiki/Boeing_A160_Hummingbird on 14 April 2015.

[5] DJI, "The Phantom3", Retrieved from http://www.dji.com/product/phantom-3 on 16 April 2015.

[6] Alex M. Stoll, Edward V. Stilson, JoeBen Bevirt and Pranay Sinha, "A Multifunctional Rotor Concept for Quiet and Efficient VTOL Aircraft", 14th AIAA Aviation Technology, Integration, and Operations Conference, Los Angeles, USA, August 2013.

[7] Gabriel M. Hoffmann, Haomiao Huang, Steven L. Waslander and Claire J. Tomlin, "Quadrotor Helicopter Flight Dynamics and Control:AIAA Guidance, Navigation and Control Conference and Exhibit, Hilton Head, United States, August 2007.

[8] Mehmet Serdar Guzel and Robert Bicker, "Vision Based Obstacle Avoidance Techniques", Recent Advances in Mobile Robotics, Chapter 5, InTech, December 14, 2011.

[9] EPIX, "Silicon Video 2KS", Retrieved from http://www.epixinc.com/products/sv2ks.htm on16 April 2015.

[10] Dirk Holz, Matthias Nieuwenhuisen, David Droeschel, Michael Schreiber and Sven Behnke, "Towards Multimodal Omnidirectional Obstacle Detection for Autonomous Unmanned Aerial Vehicles", International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XL-1/W2, pp. 201-206, September 2013.

[11] G.J. Iddan and G. Yahav, “3D IMAGING IN THE STUDIO (AND ELSEWHERE...)”, Proceedings of the SPIE 4298: Videometrics and Optical Methods for 3D Shape Measurements, pp. 48-55, 2011.

[12] David Droeschel, Dirk Holz, J¨org St¨uckler, and Sven Behnke, “Using Time-of-Flight Cameras with Active Gaze Control for 3D Collision Avoidance”, In Proceedings of IEEE International Conference on Robotics and Automation (ICRA), Anchorage, USA, pp. 4035-4040, May 2010.

[13] Ian Lenz, Mevlana Gemici, Ashutosh Saxena, “Low-Power Parallel Algorithms for Single Image based Obstacle Avoidance in Aerial Robots”, In International Conference on Intelligent Robots and Systems (IROS), Vilamoura, Portugal, pp. 772-779, October 2012 .

[14] Daniel Magree, John G. Mooney, Eric N. Johnson, “Monocular Visual Mapping for Obstacle Avoidance on UAVs”, in International Conference on Unmanned Aircraft Systems (ICUAS), Atlanta, USA, pp. 471-479, May 2013.

[15] Matthias Nieuwenhuisen, David Droeschel, Marius Beul, and Sven Behnke, “Obstacle Detection and Navigation Planning for Autonomous Micro Aerial Vehicles”, In Proceedings of International Conference on Unmanned Aircraft Systems (ICUAS), Orlando, USA, pp. 1040-1047, May, 2014.

[16] Dirk Holz and Sven Behnke, “Registration of Non-Uniform Density 3D Point Clouds using Approximate Surface Reconstruction”, in Proceedings of the International Symposium on Robotics (ISR) and the German Conference on Robotics (ROBOTIK), Munich, German, pp. 1-7, June 2014.

[17] Henning Lategahn, Andreas Geiger and Bernd Kitt, “Visual SLAM for Autonomous Ground Vehicles”, In International Conference on Robotics and Automation (ICRA), Shanghai, China, pp. 1732-1737, May 2011.

[18] Microsoft, “The Kinect Sensor”, Retrieved from https://www.microsoft.com/en-us/kinectforwindows/ on 15 October 2014.

[19] Johann Borenstein and Yoram Koren. “The vector field histogram-fast obstacle avoidance for mobile robots”, IEEE Journal of Robotics and Automation, Vol. 7, No. 3, pp. 278-288, June 1991.

[20] Iwan Ulrich and Johann Borenstein. “VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots”, in Proceedings of the IEEE International Conference on Robotics and Automation, Vol. 2, Leuven, Belgium, pp. 1572-1577, May 1998.

[21] Iwan Ulrich and Johann Borenstein, “VFH*: Local Obstacle Avoidance with Look-Ahead Verification”, in Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, USA, pp. 2505-2511, April 2000.

[22] Inspector Bots, “The MINIBOT Robotic Platform”, Retrieved from http://www.inspectorbots.com/_Home.html on 20 April 2015.

[23] Hardkernel, “Odroid-U3”, Retrieved from http://www.hardkernel.com/main/products/prdt_info.php?g_code=g138745696275 on October 2014.

[24] Hokuyo, “URG-04LX-UG01”, Retrieved from https://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html on March 2014.

[25] Arduino, “Uno board”, Retrieved from https://www.arduino.cc/en/Main/arduinoBoardUno on February 2015.

[26] Robot Operating System (ROS), Retrieved from http://www.ros.org/ on November 2014.

[27] Gazebo simulator, Retrieved from http://gazebosim.org/ on November 2014.

[28] Open Dynamics Engine, Retrieved from http://www.ode.org/ on November 2014.

[29] Ubuntu Operating System, Retrieved from http://www.ubuntu.com/ on November 2014.

[30] 3DRobotics, “Pixhawk”, Retrieved from http://3drobotics.com/kb/pixhawk/ on May 2014.

[31] Paolo Stegagno, Massimo Basile, Heinrich H. Bülthoff, and Antonio Franchi, “A Semi-autonomous UAV Platform for Indoor Remote Operation with Visual and Haptic Feedback”, IEEE International Conference on Robotics and Automation, Hong Kong, China,pp. 3862-3869. June 2014.

[32] Larry Matthies, Roland Brockers, Yoshiaki Kuwata and Stephan Weiss. “Stereo vision-based obstacle avoidance for micro air vehicles using disparity space”, IEEE International Conference on Robotics and Automation, Hong Kong, China, pp. 3242-3249, 2014.

[33] Lionel Heng, Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys, “Autonomous Obstacle Avoidance and Maneuvering on a Vision-Guided MAV Using On-Board Processing”, IEEE International Conference on Robotics and Automation, Shanghai, China, pp.2472-2477, May 2011.

[34] Haosong Yue, Weihai Chen, Xingming Wu and Jingbing Zhang. “Kinect Based Real Time Obstacle Detection for Legged Robots in Complex Environments”, IEEE 8th Conference on Industrial Electronics and Applications (ICIEA), Melbourne, Australia, pp. 205-210, June 2013.

[35] David Droeschel, J¨org St¨uckler, and Sven Behnke, “Local Multi-Resolution Surfel Grids for MAV Motion Estimation and 3D Mapping”, in Proceeding of 13th International Conference on Intelligent Autonomous System (IAS), Padova, Italy, July, 2014.

[36] Jongho Park and Youdan Kim, “Stereo Vision Based Collision Avoidance of Quadrotor UAV”, International Conference on Control, Automation and Systems (ICCAS), JeJu Island, South Korean, October 2012.

[37] Ashish R. Derhgawen and D. Ghose, “Vision Based Obstacle Detection using 3D HSV Histograms”, Annual IEEE on India Conference (INDICON), Hyderabad, India, pp. 1-4, December 2011.

VITA

Graduate College

University of Nevada, Las Vegas

Lin Zhao

Degrees:

Bachelor of Science in Mechanical and Electronic Engineering, 2013

Zhejiang University City College, China

Thesis title: 3D Obstacle Avoidance for Unmanned Autonomous System (UAS)

Thesis Examination Committee:

Committee Chair: Woosoon Yim Ph.D.

Committee Member: Mohamed Trabia Ph.D.

Committee Member: Kwang J. Kim Ph.D.

Graduate College Representative: Sahjendra Singh Ph.D.