The Pennsylvania State University
The Graduate School

STEREO-CAMERA OCCUPANCY GRID MAPPING

A Thesis in
Aerospace Engineering
by
Wen-Yu Chien

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2020

The thesis of Wen-Yu Chien was reviewed and approved by the following:

Eric N. Johnson
Professor of Aerospace Engineering
Thesis Advisor

Amy Pritchett
Professor of Aerospace Engineering
Department Head of Aerospace Engineering

Jack W. Langelaan
Associate Professor of Aerospace Engineering
Aerospace Director of Graduate Programs

Abstract

Unmanned aerial vehicles (UAVs) are widely used to explore dangerous or unknown areas, i.e., battlefield reconnaissance and collapsed buildings. The risk of operating in these complex environments is high due to the difficulty in performing precision mapping for collision prevention. The primary reason is vehicle hardware limitation, where the performance of the onboard computer and sensors are usually restricted by the maximum vehicle weight. Recently, the rise of mobile devices makes micro sensors much cheaper and lighter, i.e., camera sensors, Global Positioning System (GPS) and Inertial Measurement Unit (IMU). This tendency also benefits UAVs and makes it easier and safer to operate during outdoor flight. Nevertheless, for a GPS-denied environment such as forests or indoor environments, UAVs once again encounter challenges due to weight restriction, lack of GPS data, and complexity of those environments. Unlike large UAVs, micro air vehicles (MAVs), which are more suitable for these complex environments, are restricted by their payload capacity. Laser scanners, which are commonly used for large UAVs, are too heavy for MAVs. Therefore, a lightweight camera system is a better solution to enable these vehicles to fly in GPS-denied, complex areas.

This study presents a framework to show how to enable a drone to map an unknown, complex, GPS-denied environment by integrating two Intel RealSense cameras, which are used to provide the vehicle mapping and localization ability and testing the mapping and localization ability to explore an unknown complex GPS-denied environment. The specific objectives of this research are as follows: reconstruct a 3D occupancy map by extracting depth data from the RealSense depth camera, get pose data from the RealSense tracking camera to localize the vehicle, and give reference to the extracted point clouds, all while running these processes in real-time on the onboard Intel NUC computer. An occupancy grid map is used to store the environment information from the cameras and save memory usage. The inverse sensor model of the RealSense depth camera is studied and implemented based on the camera spec sheets. The performance of two different ray tracing methods, line drawing and voxel ray traversal, are also studied. Furthermore, the map shifting function is developed to allow the 3D map to move as the vehicle moves.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my sincere appreciation to my advisor, Dr. Eric Johnson, for his advising and funding support. I am fortunate to have him as my advisor who gave me freedom on my research and guidance when facing problems. His patience and support, especially on programming, encouraged me to learn OpenGL to visualize the map and build the ground station for the project. Although it took a while to learn due to the steep learning curve, OpenGL sped up the debugging process for the following works.

This thesis was partly supported by DARPA under grant no. D18AP00069. I appreciate my sponsors, DARPA and Dr. L'Afflitto at Virginia Tech. The code integration with Dr. L'Afflitto's students, Julius and Blake, was intellectually satisfying and I learned a lot from them. Also, I learned a lot from Dr. L'Afflitto's students, Julius and Blake, on code integration and data communication between programs.

I also want to thank my labmates in the PSU UAS Research Lab (PURL). I am lucky to be in PURL and know these awesome friends. We teamed up to win the IMAV competition in Spain, and have fun discussing homework and projects. Jake helped me a lot on flight tests and mapping. He is also the best English teacher I have ever met. Ankit is always willing to introduce me to other grad students and shares new features of the RealSense T265 camera he found. Rachel is willing to share her experience in this school and gives us advise based on our situations. Ollie is talkative and willing to help everyone. He used his experience in C++ to teach me how to build a well-organized project, which made my project easier and more convenient to maintain. Venkat shared his previous experience in the aerospace industry and gave me some ideas on testbed selection and design. Jeff is the human library in our lab. He can answer almost any kind of question related to aerospace in detail and recommend a text book or a paper immediately. Henrik taught me how to solder perfectly and helped me order some electronics parts. This technique made the drone more reliable and efficient. Their support really pushed me to improve myself and I am genuinely grateful for being here.

My parents are always supportive toward any decision I make, including studying abroad. They promised to back me up with an Interest-free loan so I could focus on studying English and other more important challenges. Though I was very fortunate to get the school funding and ended up not needing their money.

Chapter 1 Introduction

1.1 Background

Autonomous UAVs have become popular in recent years, with applications being considered in many fields, such as package delivery, agriculture, cinematography, etc. One big advantage of the UAV system is that it is free to move three-dimensionally to perform obstacle avoidance. Thus, it is suitable to explore unknown or dangerous environments, as encountered while performing battlefield reconnaissance and collapsed building rescue, for example. However, these environments are usually GPS-denied and filled with obstacles, which can pose a challenge for UAVs to maneuver and localize. Thus we turn to other sensors to alleviate this deficit. Thanks to the development of sensors for mobile devices, camera sensors, and Inertial measurement units (IMUs), these sensors are cheaper and lighter alternatives for UAVs.

There are existing solutions available which can provide pose estimates necessary for indoor navigation. For example, the VICON system is a motion capture system that provides precise state information. However, VICON is an expensive line-of-sight and range limited system, and impractical for unknown environment exploration. IMUs, on the other hand, are widely used in mobile devices for motion detection but tends to accumulate error and leads to divergent drifts.

Some methods have been invented for unknown environment robot operation in a GPS-denied condition. The most popular technique is Simultaneous Localization and Mapping (SLAM). Localization is critical since a precise pose estimate is needed for both aircraft guidance and mapping. Similarly to localization, obstacle detection and mapping are also important for indoor flying. The restricted space means that UAVs cannot necessarily avoid obstacles by just flying higher. They have to do the more complex task of true three-dimensional obstacle avoidance at close range. For the full

SLAM solution, mapping and localization are used in conjunction with each other (one informs the other), and is used to further support functions like path planning, obstacle avoidance, and environment reconstruction. The map is often constructed through the use of a laser scanner or camera vision, and the range data is converted into point clouds. For aerial robots, computational efficiency is critical since the limited payload only allows vehicles to carry a relatively small computer. However, accessing point clouds is a time-consuming process and sometimes impractical for aerial vehicles in real-time operation. Also, a low-cost laser scanner usually comes with a narrower field of view and lower resolution along the vertical direction.

To autonomously fly in unknown and GPS-denied areas, several tasks need to be resolved. Firstly, the vehicle needs precise knowledge of its state vector. Secondly, the mapping process should be light and fast enough to react to the collision when the vehicle is flying. Therefore, a simplified occupancy grid map is more competitive for an aerial robot in real-time applications.

1.2 Related Works

For the GPS-denied UAV navigation, one of the most popular methods is the SLAM algorithm, which has been extensively studied. Li et al. [1] reviewed some modern real-time SLAM techniques and concluded that unlike 2D SLAM, the problem of 3D SLAM, especially on UAVs, is unsolved. Although the concept of SLAM is straightforward, various SLAM methods are studied based on the various sensors and hardware limitations.

1.2.1 SLAM

LiDAR is a popular topic widely used in self-driving cars and the robotic navigation industry, for its' high accuracy. Despite the accuracy of LiDAR systems, the heavy weight and low vertical data density such as 2D scanner with the tilting mechanism, or even pure 2D scanning [2], make it impractical for MAVs to fly in a complex indoor environment. For multicopters, pitching down is the common way to move forward. However, this maneuver causes the 2D scan plane to deviate from horizontal, thus restricting the ability to see obstacles at similar altitude to the vehicle. Some researchers put LiDAR on UAVs [3–5], through they usually focus on fixed altitude flight, or the use of a relatively large helicopter (which is not suitable for a complex, indoor environment).

The monocular SLAM method, on the other hand, has been another popular research

avenue, due to economy of cost and weight, and has been proposed by Celik et al. [6] and Davison et al. [7]. This method allows robots to estimate the position and pose in space and can reconstruct the outlines of the environment. The most well-known methods are ORB-SLAM proposed by Mur-Artal et al. [8] and LSD-SLAM proposed by Engel et al. [9]. The first method used feature points to extract the depth data to do the mapping and localization. As for the LSD-SLAM, edges were used to detect the depth information. The problem with monocular SLAM is the depth information is non-scaled, which means the calibration has to be done before the SLAM runs. Also, the depth density of monocular SLAM is relatively low and impractical for non-gradient surface detection and obstacle avoidance. The ORB-SLAM can only return sparse depth images and LSD-SLAM can only return semi-dense depth images.

On the other hand, binocular SLAM can measure the depth data directly. Although the weight is slightly heavier than the monocular sensor, the dense depth data is real-time (unlike the one-frame delay of monocular SLAM) thus making obstacle avoidance and environment reconstruction more practical. Endres et al. [10] proposed a way for 3-D mapping with an RGB-D camera and Kerl et al. [11] proposed the keyframe selection for vSLAM with RGB-D cameras. The 3D benchmark for RGB-D cameras was proposed by Sturm [12] and is widely used to check RGB-D SLAM quality. Many RGB-D papers were studied due to the release of the Microsoft Kinect RGB-D camera. The relatively lightweight and easy-to-use API makes the sensor popular at the time. The Intel RealSense depth camera D435 is another lightweight RGB-D camera at 70g with active emitter.

1.2.2 Occupancy Grid Map

Occupancy grid mapping decomposes the environment into voxels, where cells in the map update area are considered either occupied or free. Unlike the point cloud representation, this concept improves the processing speed which decreases computational cost and saves memory usage. The inverse sensor model is used to assign the probability of occupancy for a grid cell based on depth sensor measurements. Ray tracing is used with the inverse sensor model to assign the occupancy along the ray from the sensor to the point clouds. Two classical methods are proposed here, Bresenham's line drawing [13] and fast voxel ray traversal [14] - both of which are workable, depending on the scenario and hardware.

The occupancy grid was first proposed by A. Elfes in 1989 describing how grid cells can be updated using Bayes theorem [15]. Moravec, H. and Roth-Tabak Y et al. first put the concept on a robot and built the environment model by using a depth sensor [16,17].

Andert later proposed an occupancy mapping method by using stereo images with the inverse sensor model, ray tracing, and image pyramid to speed up the process [18]. Evan Kaufman et al. used an exact inverse sensor model to map the accurate occupancy map [19]. In the book Probabilistic Robotics (Thrun et al. [20]), many robotic navigation and SLAM methods are described in detail. Occupancy mapping with a forward sensor model is studied by Thrun et al., but the method is impractical for real-time mapping due to the high-dimensional map and heavy computational load needed for estimate convergence. Obstacle avoidance and tracking with an occupancy map are proposed by Nguyen et al [21]. This method was developed using a limited height 3-D map and clusters grid cells by grouping close occupied cells together; an effective method for obstacle avoidance and path planning.

1.3 Motivation and Objective

Relative to outdoor flying, an indoor flight in a GPS-denied environment is more complicated due to the constrained height and complexity of the space. For aerial robots or drones, reaction time is critical as there is a high collision risk for these vehicles, especially for fixed-wing drones. Even for a hovering drone or helicopter, the inverse thrust-like maneuvers usually induce severe attitude adjustment and instability for a fast flight. Therefore, a fast mapping update rate provides an aerial vehicle with enough reaction time to either avoid obstacles or decide a new path.

The Microsoft Kinect RGB-D sensor has been a game-changer in the mapping research field for its accuracy and light weight. The ZED stereo camera and Intel RealSense depth camera are also becoming popular for both research and the commercial market. The RealSense depth camera is the lightest depth camera among the cameras mentioned above at 70g. The active IR emitter also improves the depth image quality on textureless surfaces and in low-light environments. Similarly, the RealSense tracking camera T265 is a Visual Inertial Odometry (VIO) sensor that provides position and orientation estimation at only 50g.

Most studies have provided a SLAM solution by using one sensor at a time, i.e., one monocular sensor or one binocular sensor. However, the combination of two onboard sensors, one for localization and one for mapping, is rare due to the added weight. The combination of the RealSense cameras, however, is convenient and can speed up the development pace thanks to the well-developed RealSense API. As an added benefit, the Visual Process Unit (VPU) in both cameras can save CPU resource usage. Currently,

many applications use a 2D map because of hardware limitations. With the VPU, the CPU can focus on the mapping task, thus allowing for 3D map reconstruction.

The main goal of this thesis is to develop an accurate real-time 3D occupancy map using only RealSense depth and tracking cameras. To achieve the real-time requirement, the desired update rate is set to at least 15Hz, which can provide enough reaction time for a maneuvering drone. The research also simply lists and introduces the important steps to do real-time 3D occupancy grid mapping so that drone developers can easily implement this occupancy mapping and leverage it for their path planning. The scanned results were tested from the hand-held RealSense depth and tracking cameras in this research.

1.4 Thesis Structure

- Chapter 2 introduces the selection of the hardware and the test arena in detail. The specifications of the sensors are provided. Moreover, characteristics and calibrations of the hardware are presented and tested according to the pros and cons of the hardware.
- Chapter 3 presents the design and methods of the occupancy grid mapping. In addition to basic steps including the inverse sensor model, ray tracing and map update, the resolution downsampling and map shifting are introduced for advanced applications.
- Chapter 4 shows the scanned results of obstacles and an indoor room along with the computational costs.
- Chapter 5 summarizes the results, discusses discovered issues and provides suggestions for future research.

Chapter 2
System Setup

2.1 Complex Environment Setup

For simulating a complex environment, several boxes of different sizes are placed in the test arena. My apartment was then scanned to test the mapping algorithm in a real scene with varing conditions such as brightness, texture, door frame size, and narrow aisles.

2.2 Hardware Selection

2.2.1 Vehicle Selection

Aerial vehicle selection is quite important for system integration. One of the sensors used in this research is a wide field of view camera for estimating vehicle pose (described in section 2.2.3) so the front of the vehicle should be clear of any obstacles obstructing in the camera's field of view. Conventional helicopters and multicopters are much better than fixed-wing aircraft for indoor flight because of their hovering ability. A quadcopter is also more suitable than either a hexacopter or an octocopter since it is easier to get an unobstructed sensor field of view, especially H-Frame configuration quadcopter. The H-frame quadcopter is a modified X-frame F-450 (henceforth referred to as H-450)size frame wheel. The motor and propeller selection is based upon the hardware configuration. The estimated weight of the final configuration is about 1800g, with 2G desired acceleration to ensure adequate performance. The final system is an H450 size quadcopter with 3600g lift, as shown in Fig.2.1.

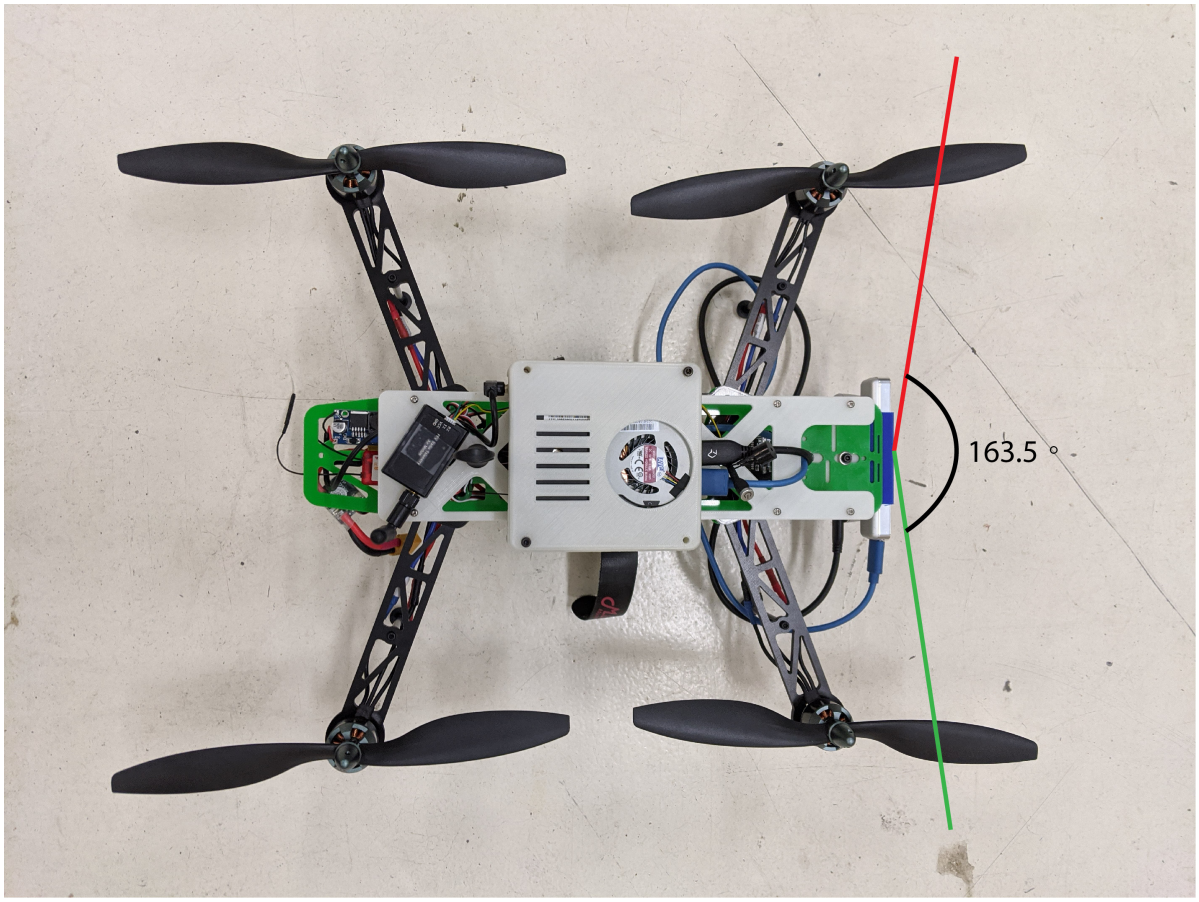


Figure 2.1: Selected Vehicle H-450 with 3600g lift. The cameras are in front of the frame trying to remove the props from the camera's field of view. Although the propellers are still partially in the field of view of the tracking camera, the pose data is negligibly affected by this (compard to the traditional X-frame).

2.2.2 Flight Controller and Computer Setup

The Pixhawk was selected for the system because it is lightweight and easy to integrate with external motion capture data. The Pixhawk [22] is running PX4 firmware which can easily send commands and receive data from the vehicle through Mavlink. For programming, MAVSDK [23] was chose as a higher level API of Mavlink, to communicate between the Intel NUC-i7 board and Pixhawk.

2.2.3 Camera Selection

There are two cameras on this platform: one stereo camera and one tracking camera. The Intel RealSense active stereo camera D435 was chosen for the mapping. The active

IR emitter improves the featureless plane depth detection and performance in low-light environments, which is important for indoor flight. The stereo camera is equipped with global shutter sensors which improves the image quality in fast operation. According to the official specification, the practical range of the depth camera can reach up to 16 meters. The Intel RealSense T265 tracking camera was chosen to provide vehicle pose estimates. The camera comes with two 163±5° fish-eye lenses to detect as many features as possible for the tracking camera. Also, the camera has an onboard mapping function that can correct for drift with the re-localization function enabled. Finally, both cameras are equipped with VPUs to prevent using CPU resources for the computer vision process.

(a) Intel® RealSense™ Depth Camera D435 (b) Intel® RealSense™ Tracking Camera T265

Figure 2.2: Intel® RealSense™ Cameras

2.2.4 Camera Characterization and Calibration

Each camera was tested to know the best settings for mapping. The depth image quality of the D435 active stereo camera is related to brightness, emitter power, texture, and distance. The data accuracy is affected by brightness and texture. High accelerations such as vibrations from motors and hard landings, also affect the IMU. The following section discusses more detailed tests for the depth camera and tracking camera. Table.2.1 and Table.2.2 list some important specs of the two RealSense cameras.

Table 2.1: RealSense D435 specs

Parameter	**Sensor Properties**
Active Pixels	1280×800
Max. Depth Resolution	848×480
Focal Length	1.93mm
Shutter Type	Global Shutter
Horizontal Field of View	91.2°
Vertical Field of View	65.5°
Baseline	50mm

Table 2.2: RealSense T265 specs

Parameter	**Sensor Properties**
Active Pixels	848×800
Shutter Type	Global Shutter
Fisheye Field of View(D)	173°
Baseline	64mm
IMU DOF	6
Acceleration Range	±4g
Accelerometer Sample Rate	62.5Hz
Gyroscope Range	±2000 Deg/s
Gyroscope Sample Rate	200Hz

2.2.4.1 Depth Camera Test

The depth camera was tested with several factors including brightness, emitter power, and texture. To achieve the best result for the depth image, the number of the effective non-zero depth pixels is counted as quality. In general, the greater the number of effective pixels the better.

Calibration

Intel provides calibration software called CalibrationToolAPI and Depth Quality Tool for Intel RealSense cameras. The first software can get the intrinsic and extrinsic matrices from a RealSense chessboard plot to calibrate the distortion of the sensors. The second one is for depth quality calibration. During this step, the camera is directed toward a

perpendicular flat wall at a distance of 2m. According to Intel, they use deep-learning to tune hundreds of parameters to fit the target depth data. During the calibration, we found one of our depth cameras, D435i with IMU, has an image distortion issue, as seen in Appendix-A.

Brightness Test

There is an auto-exposure function for the D435. We tested the camera with different exposure times with the results shown in Fig.2.3. The test used different exposure times for collecting manual exposure data. Then we calculate the mean brightness from every effective pixel in each frame. It shows how the number of effective depth data points vary with brightness. According to Fig.2.3, auto-exposure is reliable in a common environment, but it could still be adversely affected when faced with a high contrast environment due to the sensor limit.

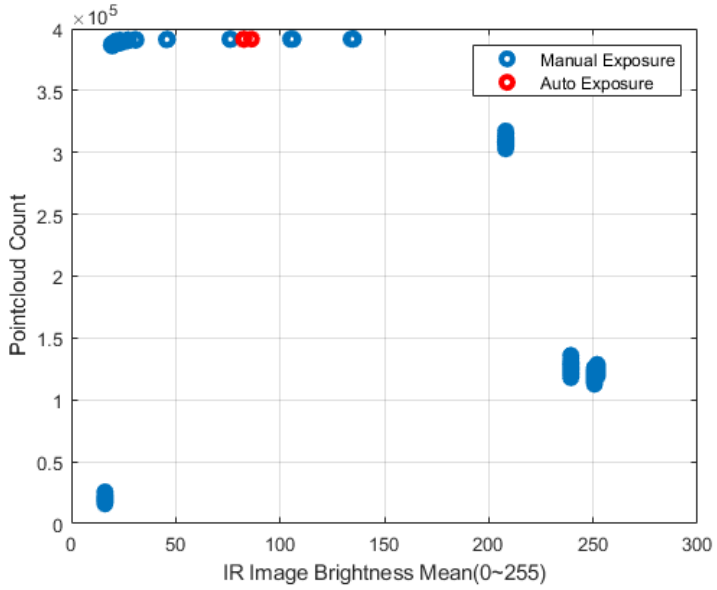


Figure 2.3: Brightness test. The brightness that the auto exposure captured lies in the middle of the highest data sets of the manual exposure. The blue circles are the manual exposure results and the red circles are the results of the auto-exposure.

Emitter Power Test

Emitter power is another important factor for the quality of an active stereo camera. The strength of the emitter is affected mainly by the environment. In a brighter environment light, the emitter power has to be higher to be visible for the camera. Conversely, the power of the emitter can be lower in a darker environment.

The power range of the laser power is from 0 to 360mW. The test consisted of collecting data from the depth camera at a fixed distance with 150 frames. The result counted the number of effective cloud points from the depth camera at different laser powers, as shown in Fig.2.4. The result improves from 0 to 60mW and stays almost the same beyond 120mW in this test. A higher power is required when the texture is more complex or the distance to obstacles is further. However, this increase in power could heat up the camera hardware and induce higher noise. Therefore, the power should not be too high if the quality is acceptable. In this research, the power is chosen as 150mW as the default.

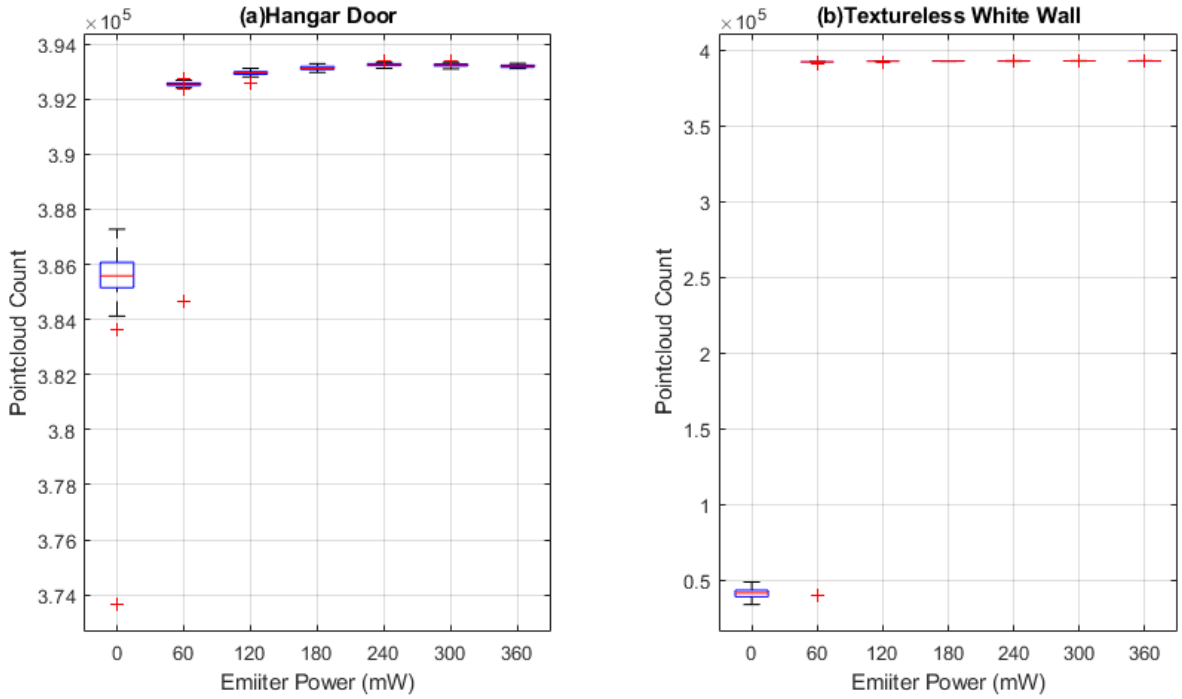


Figure 2.4: Emitter power test direct to different texture. The left image was tested facing a textured surface and the right image was tested facing a non-textured surface. For textured surface to achieve highest number of effective point clouds, the required power is higher than the non-textured surface.

Texture Test

Stereo cameras use feature points or gradients to compare two images to estimate depth data, which implies the texture of the obstacle affects the depth image quality. Since the RealSense D435 is an active stereo camera, it can still calculate the depth of a non-textured surface.

We tested the camera on different textures including a white flat wall, an iron sheet wall, and acrylic. As expected, the stereo camera can barely detect transparent or mesh textures. One of the most severe issues is pointing the camera toward repetitive textures at a further distance. In this case, the depth algorithm would sometimes misidentify the feature points and return incorrect depth data, underestimating the value, as Fig.2.5(c) shows. Repetitive textures are the most common issue for indoor flight since indoor environments are full of repeating patterns. This issue can be fixed by increasing the power of the emitter higher or moving the camera closer to the repeated pattern surface. Another workaround is to tilt the camera up or down, around 20° according to the RealSense Tuning document [24]. The tested results are shown in Fig.2.5. We can see the depth images are improved on both tilting up and down, but tilting down produces the best results.

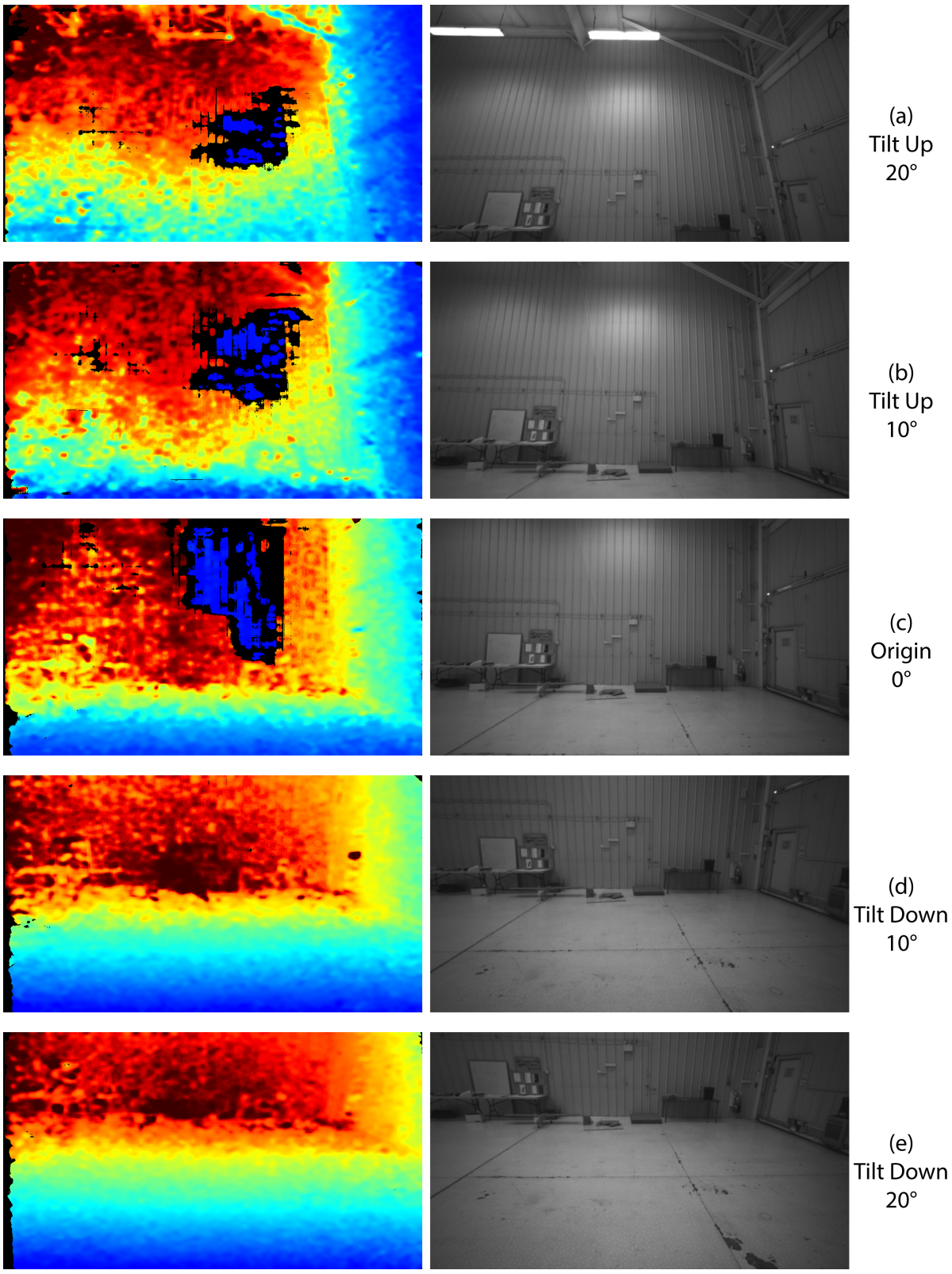


Figure 2.5: Repetitive Texture Test and Workaround

Root Mean Square Error Test

The accuracy of the D435 was tested in front of a flat wall. Here, we tested three different distances: 1m, 4m, and 8m. We then used a curve fit to make further predictions. This will be described in more detail in Chapter 3.

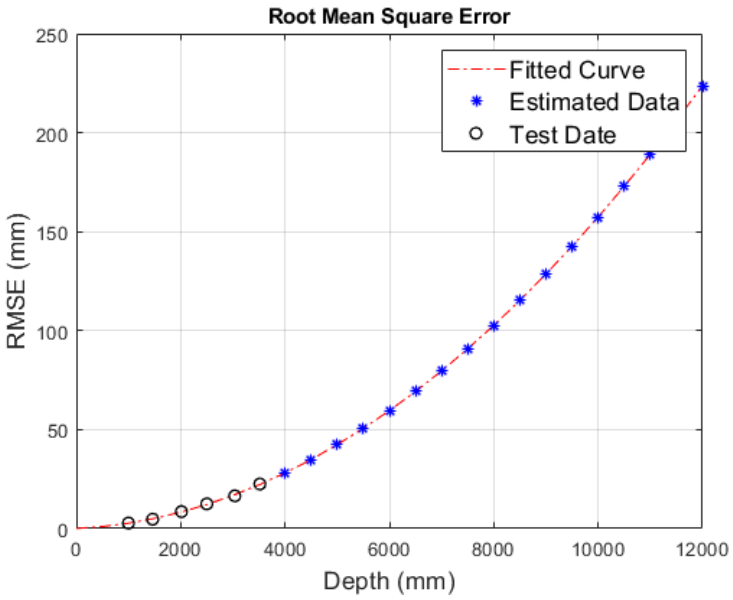


Figure 2.6: Root Mean Square Error at different depth

2.2.4.2 Tracking Camera Calibration and Testing

Calibration

Unlike the depth camera, there is no way to calibrate the T265 using RealSense software, but the camera provides a confidence in the pose data. The confidence is low at the beginning and improves as the camera starts moving. Once the camera starts moving, the camera's onboard application starts its mapping, and the confidence of the pose data increases. Also, the pose will become more precise upon further visits to the same area in our experiments. A maximum achieved accuracy occurs when the camera revisits a scene, which implies that the system is updating the map.

Texture Test

Textures also affect the tracking camera. If it is facing a non-textured surface, like a white flat wall, it would be hard to extract feature points for a pose estimate. However,

the camera is using wide fish-eye lenses, which make it harder to find a featureless plane. The problem with this wide field of view is that it can see the tip of the propellers even with being mounted in front of the H-frame quadcopter as shown in Fig.2.1.

High G Issue and Solution

Vibration and hard landings, which are a common issue for quadcopters, also affect the cameras. High G acceleration in these two cases blur the image on both cameras and affect the feature detection, which affects the depth data and confidence of the pose data. Also, high-G acceleration causes an offset in the IMU. According to the specs, the IMU can only tolerate up to 4G. Therefore, we put a rubber damper between the airframe and the payload/sensor frame, as shown in Fig.2.7.

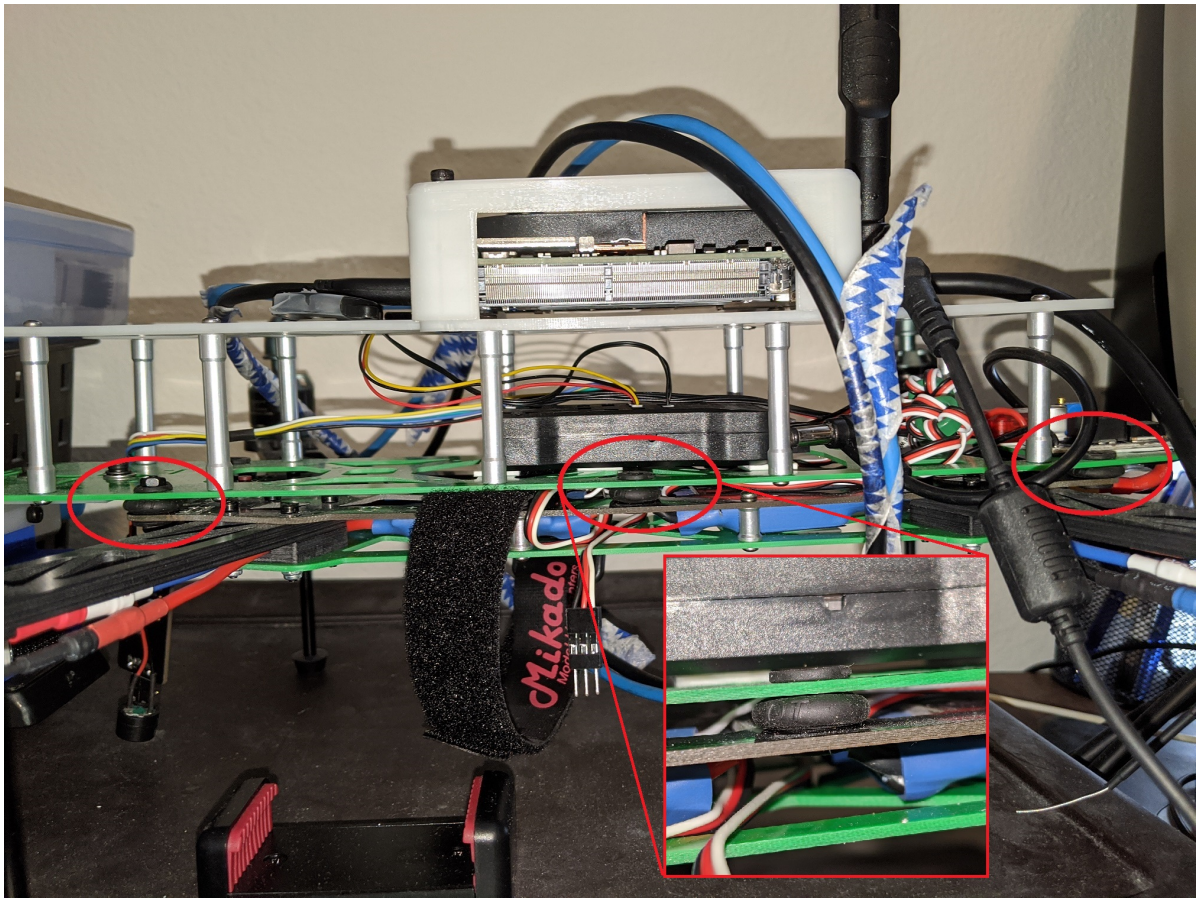


Figure 2.7: Payload frame damper

Chapter 3
Stereo Camera Mapping

3.1 Occupancy Grid Mapping Pipeline

Occupancy grid mapping partitions the world into simple cells. Fundamentally, there are three possible states of any particular cell on the map: unknown, occupied, or free. The grid map recursively updates all measured data from the first frame to the latest incoming frame in a probabilistic form. Each point cloud inside the cell is considered occupied and the occupancy probability increases. Rays are obtained from the position of each point of the camera, which is considered free and thus decreases the occupancy probability.

The flow chart in Fig.3.1 shows the relationship between each step of the mapping method. The method first extracts camera states from the RealSense T265 and converts it into the map coordinate system. Then the RealSense D435 provides the positions of points in the point cloud, where those points will be considered as obstacles in the mapping process. Once the camera states and positions of points are known, the ray tracing engages and starts to assign values to the grid map. Finally, the map can be sent for visualization in a ground station or used for obstacle avoidance.

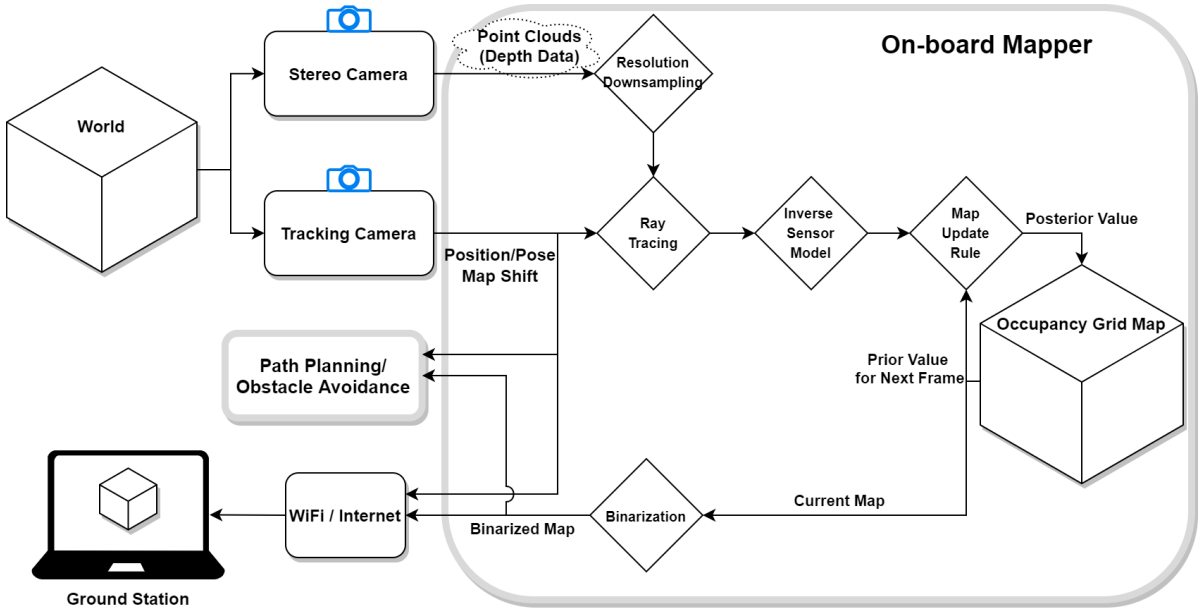


Figure 3.1: Mapping flowchart. The cameras sense the world and return depth and pose date to the onboard mapper. The resolution downsampling reduces the number of points to speed up the mapping process. With the positions of points and the cameras, rays can be calculated and the inverse sensor model can assign a probability to the occupancy grid map. The map update rule handles the previous probability value and the in-coming probability value. The occupancy grid map can then be sent to path planning and the ground station.

To reduce CPU resource usage, we utilize a resolution reduction method before starting mapping. The main purpose of the mapping algorithm is not only to reconstruct the environment for path planning but also to detect obstacles. Reducing resolution but keeping closer points can prevent the program from missing hazardous obstacles.

Finally, the map shift process is also important for unknown environment exploration since the area of interest is usually uncertain in dimension, but the size of the map is fundamentally limited by available memory. With this function, the dynamic zone of the map moves with the vehicle and keeps updating the map. As the cells move outside the mapped region, they can be either deleted or stored (in some other non-volatile/slow memory), as needed.

3.2 Depth Resolution Downsampling

The highest depth resolution of RealSense D435 is 848 × 480, or about 400,000 points in an ideal situation. The mapping process, including ray tracing and occupancy as-

signment, is slower with this full resolution. Reducing the resolution can speed up the process but may also eliminate important details in scenes such as dangerously close obstacles. Therefore, a way to speed up the mapping process and prevent it from losing critical data is to extract the closer depth pixels in each sub-area so that vehicles can still sense the dangers during flight. Fig.3.2 shows the depth resolution downsampling. For this example, the downsampling ratio is 4, which means that it compresses a 4-by-4 pixel area, 16 pixels, into one single pixel with the minimum depth.

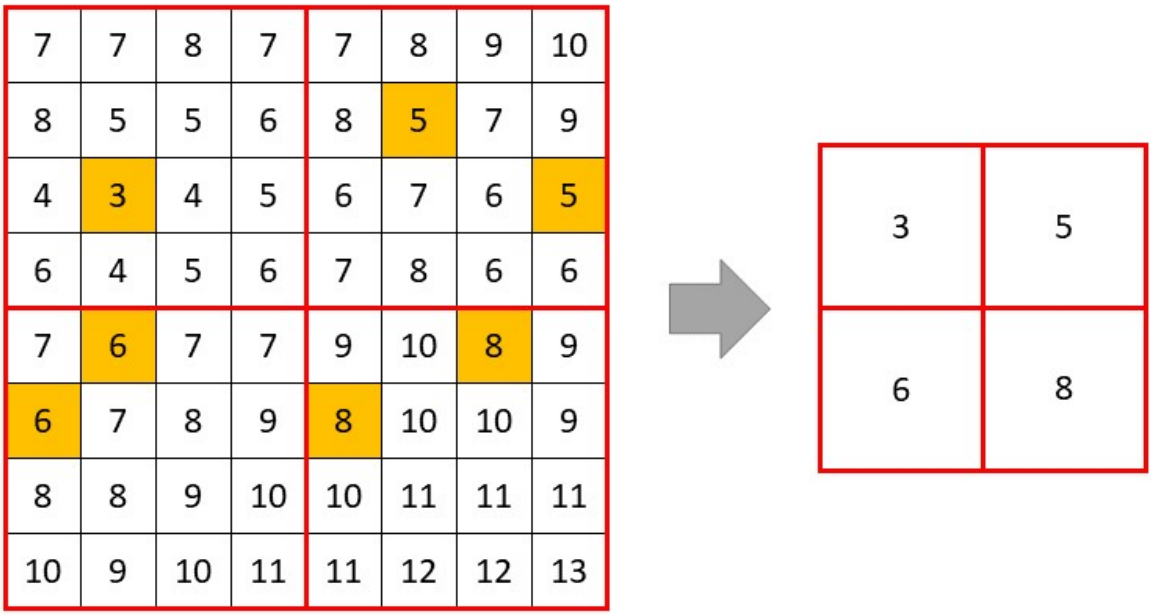


Figure 3.2: Resolution downsampling with 4-by-4 ratio. The value in each cell is depth data and the highlighted cells are the lowest reading in each sub-area.

In Fig.3.2, the red blocks are the new pixels after downsampling, and the orange blocks represent the minimum depth data point in each sample domain. After the downsampling, the new frame would only show the minimum depth in each new pixel.

3.3 Sensor Model

The inverse sensor model assigns the probability of occupancy for a cell based on sensor measurement and sensor state. It is a classic method for occupancy grid mapping update rule, which will be specified in the following section. This is well-documented in Kaufman [25].

The forward sensor model shows the probability of cell occupancy from measured depth given a known map status. It can be considered as a normal physical model in the real world and presented as a probability density function below:

$$p(z_t|m) \tag{3.1}$$

where m is the map and z_t is the measured data at time t. $p(z_t|m)$ represents the probability of the measurements given the map. There are several different models based on sensor and environment mentioned in [26].

The forward sensor model characterizes a depth sensor and can express the accuracy, noise, and maximum range. Fig.3.3 shows the schematic plot of the forward sensor model. The beam emits from the obstacle m to the camera. The 2D map on the left shows the probability density function of the sensor model hit, p_{hit}, with Gaussian distribution. According to Thrun [26], there are also other probabilistic models for the forward sensor model, such as reading from random obstacles, unexpected obstacles, and maximum reading range. Therefore, the forward sensor model characterizes the depth sensor and can express the accuracy and maximum range.

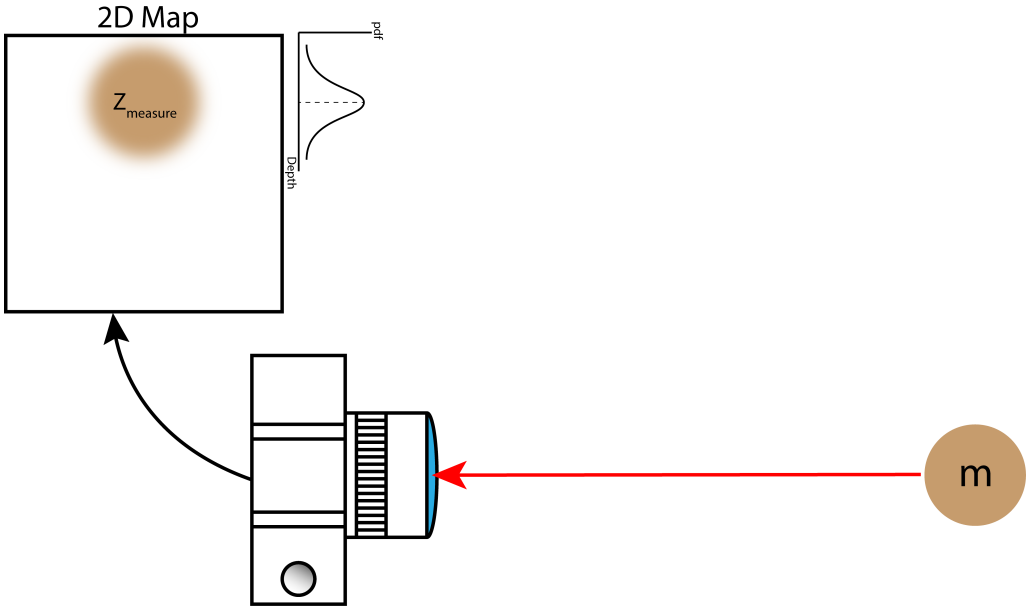


Figure 3.3: Forward Sensor Model. m is the true depth of an object in the world and $Z_{measure}$ is the sensor reading of that object. Due to the sensor noise, the data reading is not a fixed value but a fluctuating value with a Gaussian distribution.

In theory, the forward sensor model is a straight forward method for mapping. However, due to the complexity of the high-dimensional map and the heavy loading of the

computation in real-time, an occupancy map is usually decomposed into one dimension and an inverse sensor model is used [26].

Conversely, the inverse sensor model considers the sensor reading and estimates the probability of occupancy of a cell. Therefore, the arrow in Fig.3.4 is pointing outward. The function can be described as the equation below:

$$p(m|z_t) \tag{3.2}$$

The equation represents the probability of the map with given measurements. Namely, by reading the measurements, the probability distribution of the map can be estimated.

With the model, the mapping update rule is used to revise the estimated probability of the map occupancy from the given sensor reading.

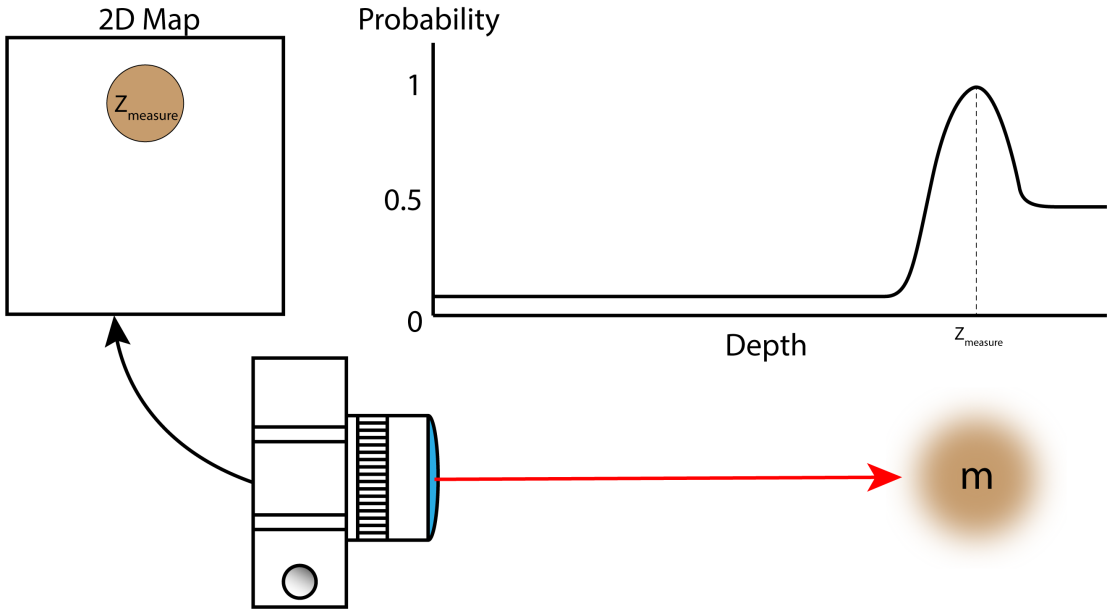


Figure 3.4: Inverse Sensor Model

A simple stereo vision inverse sensor model is proposed in [18]. The modeling combines the linear and Gaussian functions together with an importance factor k and uncertainty σ:

$$p(z|z_t) = p_{occ} + (\frac{k}{\sigma\sqrt{2\pi}} + 0.5 - p_{occ})e^{-\frac{1}{2}(\frac{z-z_t}{\sigma})}, \text{ if } p(z|z_t) \leqslant 1 \tag{3.3}$$

$$\text{where } p_{occ} = \begin{cases} p_{min}, & \text{if } 0 < z < z_t \\ 0.5, & \text{if } z > z_t \end{cases} \tag{3.4}$$

The importance factor k is used to change the amplitude of the probability function and can be tuned to achieve the desired performance. p_{min} is the probability that the sensor returns a wrong value, which can be close to zero for an accurate sensor. The depth error, σ, can be obtained from the camera specs or experimentally determined from the root mean square error. No matter how close the depth is, the probability should not be greater than 1. However, the Eq.3.3 could greater than 1 at some points(e.g., large k with a small σ). Therefore, we added a condition to the equation to prevent it from overflowing. In the RealSense tuning test [24], the theoretical depth error of the D435 can be calculated using the following formula:

$$DepthRMSError(mm) = \frac{Distance(mm) \times Subpixel}{focallength(pixels) \times Baseline(mm)} \tag{3.5}$$

$$\text{where } focallength(pixels) = \frac{1}{2}\frac{X_{res}(pixels)}{tan(\frac{HFOV}{2})} \tag{3.6}$$

A subpixel, a term that expresses the calculated depth image quality, is a floating number which depends on the image variance quality or texture. A subpixel of a well-textured object could reach 0.05 or lower. According to the RealSense test document [24], the subpixel should be no greater than 0.2 or the camera must be recalibrated. This parameter can be checked in the RealSense Viewer application.

Fig.3.5 shows the theoretical root mean square error (RMSE) distribution at different distances. Fig.3.6 shows the theoretical inverse sensor model of RealSense D435 at different distances based on Eq.3.3.

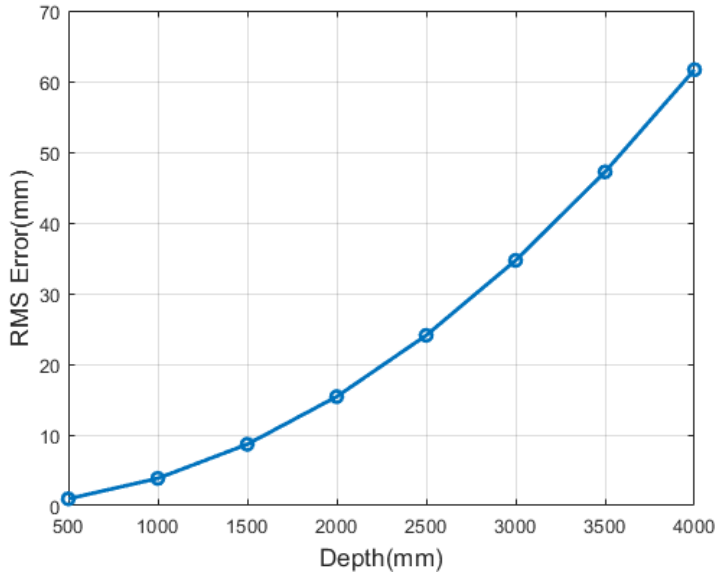


Figure 3.5: Theoretical RMSE of D435

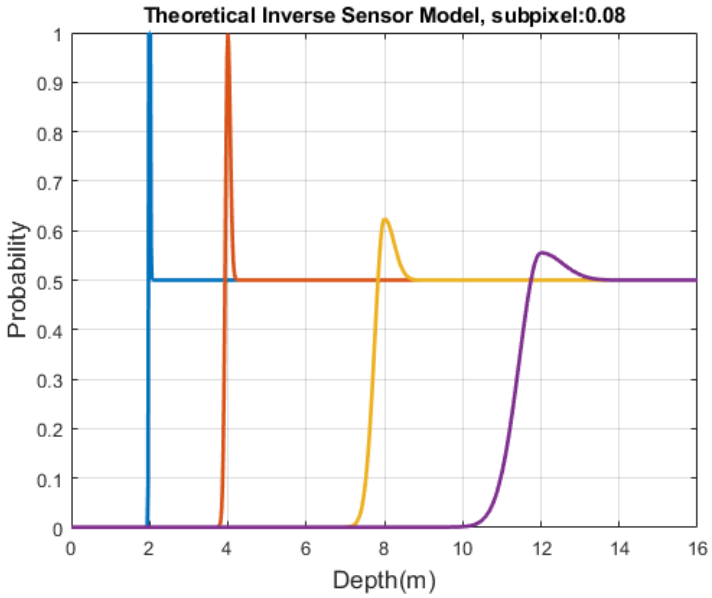


Figure 3.6: Theoretical inverse sensor model

As for the tested RMSE, the region of interest is required due to its wide field of view and there's no proper flat plane for the test. The region of interest is not the full resolution but a 0.6(m)x0.4(m) area along the center of the optical axis. According to RealSense documents, the recommended subpixel should be 0.08 based on their test among various environments and textures. The reason why our tested RMSE is much

smaller than than the theoretical one (0.08) is that we set a 60 percent region of interest in the center. Another possible reason is that the tested wall is covered by newspapers, which can provide plenty of features for high-quality depth images. The tested RMSE is shown in Fig.3.7 and the tested inverse sensor model is shown in Fig.3.8. We conclude that if we could test the depth camera with different textures, the result will be close to the theoretical RMSE with subpixel 0.08. The RMSE changes the width of the Gaussian distribution on the inverse sensor model. Thus, we use the recommended subpixel 0.08 for our inverse sensor model. The comparison of the theoretical RMSE and tested RMSE is shown in Fig.3.9

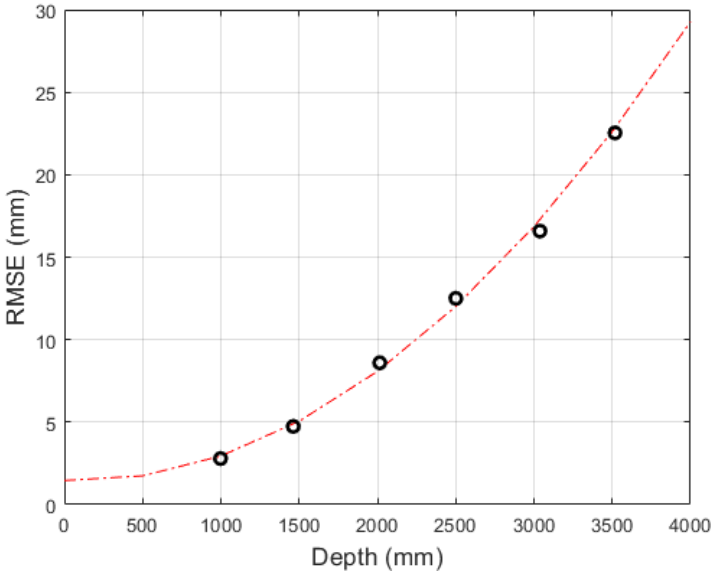


Figure 3.7: Tested RMSE of D435.

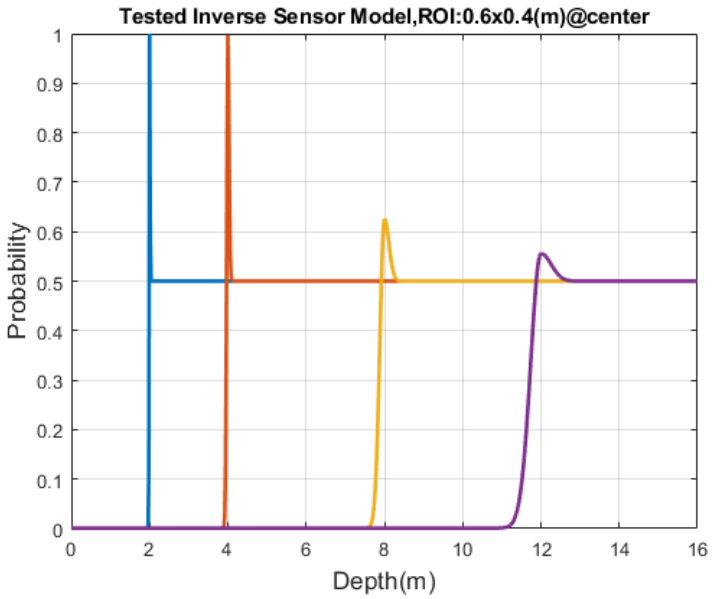


Figure 3.8: Tested inverse sensor model

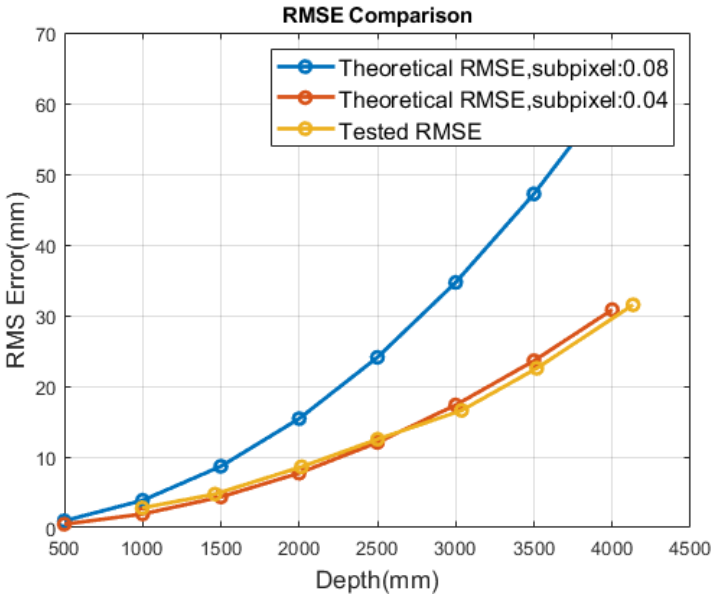
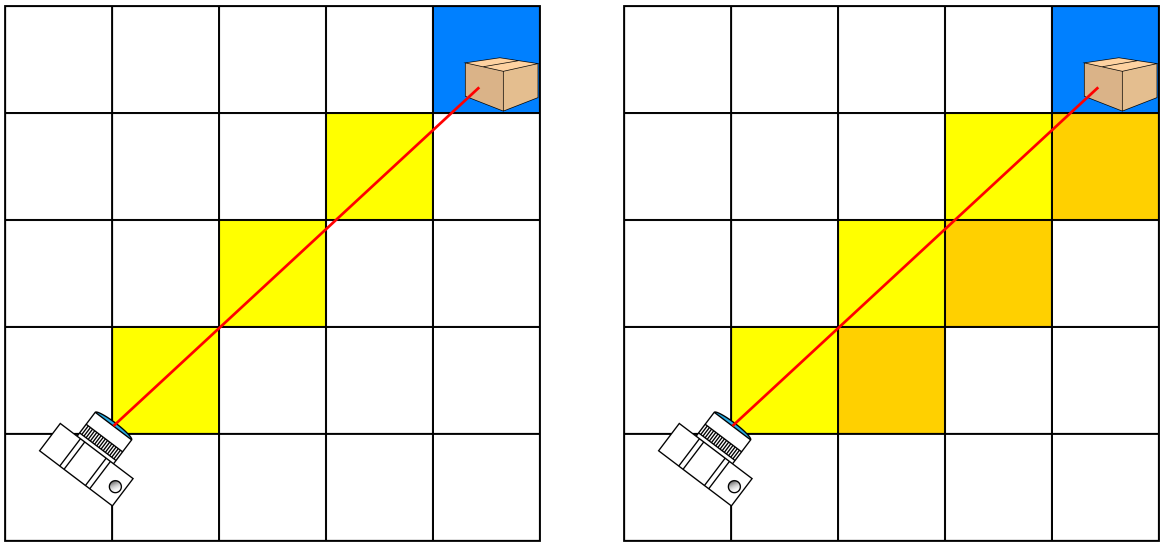


Figure 3.9: RMSE Comparison of D435. The tested result is close to the error with 0.04 subpixel.

3.4 Ray Tracing

Ray tracing not only calculates the direction and distance from obstacles to the camera but also determines the occupancy of each cell, either occupied by an obstacle or free. This step is required for the inverse sensor model in the next section. Here we only consider the two-dimensional case to simplify explanation and demonstration, but the algorithm is in 3D.

To construct these rays, the algorithm has to know the position of the camera and each individual point in the point cloud, setting the origin of the ray at the camera, and the end at the point. There are two classic methods to implement this approach: Bresenham's line drawing algorithm [13] (which was originally designed for a bitmap in computer drawing) and the fast voxel ray traversal algorithm [14]. The difference between the algorithms is shown in Fig.3.10 in 2D to simplify the explanation.



(a) Line Drawing

(b) Fast Voxel Ray Traversal

Figure 3.10: Ray tracing methods. Line Drawing(a) tries to make the line as thin as possible with less effort. Fast Voxel Ray Traversal(b) tries to fill out every cell the ray pass. The difference is shown in orange.

3.4.1 Line Drawing

Like the demonstration above, the Line Drawing algorithm mainly focuses on making a fine straight line instead of filling out all the cells in its path. Therefore, the computational cost is lower than Fast Voxel Ray Traversal but it misses some cell assignments

for rays.

The line drawing here is not exact Bresenham's method but a similar concept. The equation of the ray is $\vec{c} + i\vec{v}$ for $i \geq 0$, where $\vec{c}$ is the current camera position, $\vec{v}$ is the unit vector from camera to point, and i is the step interval of the ray. To find the unit vector $\vec{v}$, the algorithm first checks the slope of the ray. Next, the longest axis of the ray (Δx or Δy in Fig.3.11) is called the primary axis Δ and equal to the cell size. Take the left ray in Fig.3.11 for example, the longer length is y-axis so we set it as the primary axis. Then the unit vector can be calculated by the primary axis and the slope. The unit vector is from the gray dot to the first green dot for each ray in Fig.3.11. The last step is to extend the ray by using the equation $\vec{c} + i\vec{v}$ until the ray reaches out of the occupied cell. An illustration of the line drawing is shown in Fig.3.11.

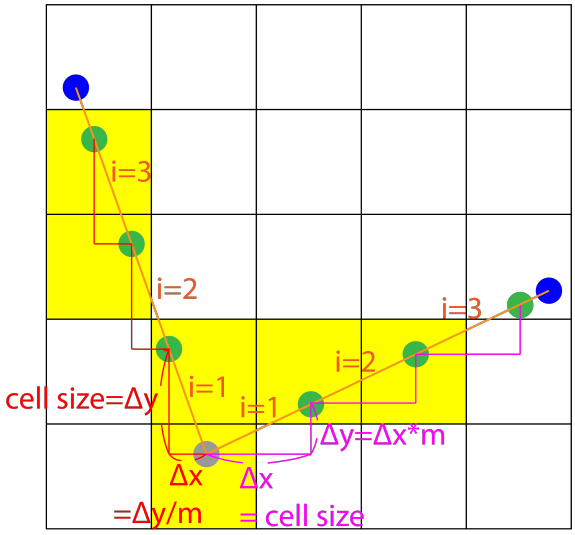


Figure 3.11: Line drawing. Blue dots are the points in the point cloud, the gray dot is the camera, and the green dots are the checkpoints of the ray increment.

Algorithm 1: Line drawing algorithms. l_{cell}:unit cell length

Input: $pos_{pointcloud}$, pos_{camera}

$\overrightarrow{ray} = pos_{pointcloud}$ - pos_{camera} ;

r_{prim} = find primary ray direction ;

$m_{xz} = \overrightarrow{ray}.z$ / $\overrightarrow{ray}.x$;

$m_{zy} = \overrightarrow{ray}.y$ / $\overrightarrow{ray}.z$;

if $r_{prim} == x_{dir}$ **then**

    $dx = l_{cell}$;

    $dz = dx * m_{xz}$;

    $dy = dz * m_{zy}$;

else if $r_{prim} == y_{dir}$ **then**

    $dy = l_{cell}$;

    $dz = dy / m_{zy}$;

    $dx = dz / m_{xz}$;

else

    $dz = l_{cell}$;

    $dx = dz / m_{xz}$;

    $dy = dz * m_{zy}$;

end

$ray_{unit\ length} = sqrt(dx^2+dy^2+dz^2)$;

$ray_{index} = |\overrightarrow{ray}| / ray_{unit\ length}$;

for $i = 0 : 1 : ray_{index}$ **do**

    $\overrightarrow{ray} = d_{x,y,z} * i$;

    $ray_{block} = \overrightarrow{ray} / l_{cell}$;

    assign occupancy value ;

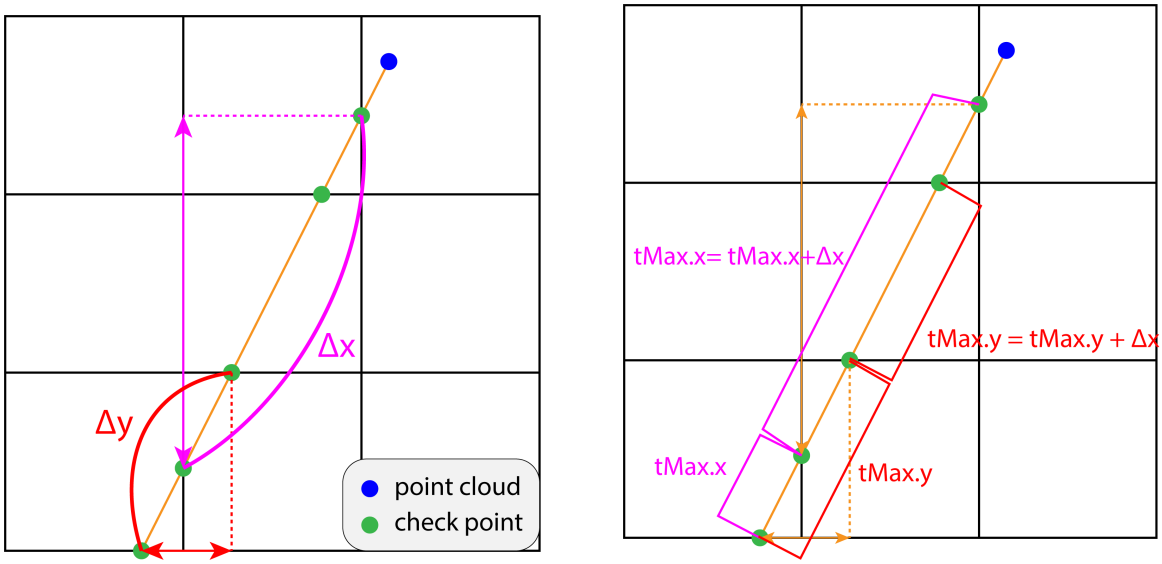
end

3.4.2 Fast Voxel Ray Traversal Algorithm

The Fast Voxel Ray Traversal algorithm is mainly designed for voxel grid mapping. Since some range sensors (such as sonar ranger sensors) have looser data density, this method could ensures all cells along the rays are updated. Due to the high data density of the depth camera, we chose the line drawing method for mapping, due to the computational benefits (as discussed in the next chapter).

The hypotenuse of the distance between two horizontal grid lines along the ray is the unit vector on y-axis, shown as Δy in Fig.3.12(a). Similarly, the pink arc in Fig.3.12(a) is the unit vector on x-axis, Δx.

Finally, the ray moves forward in a loop. In each iteration, the algorithm compares the unit vector container tMax between x-axis and y-axis, and adds one Δ to the minimum tMax. Take Fig.3.12(b) for example: the initial tMax.x is shorter than tMax.y so tMax.x has to add Δx in the second iteration. Thus in the third iteration, tMax.y has to add one Δy since tMax.y is shorter.



(a) Δ Ray (b) Ray Increment

Figure 3.12: Fast voxel ray traversal. Blue dots are point clouds and green dots are checkpoints from ray increment $tMax$. The pink lines show the unit ray increment.

Algorithm 2: Fast voxel ray traversal algorithms

Input: $\text{pos}_{\text{pointcloud}}$, $\text{pos}_{\text{camera}}$
i = x,y,z;
$\overrightarrow{\text{ray}}$ = $\text{pos}_{\text{pointcloud}}$ - $\text{pos}_{\text{camera}}$;
if $ray.i > 0$ **then**
    setp.i = 1;
    map boundary.i = grid.i;
    $\text{cam}_{\text{block}}$.i = $\text{ray}_{\text{block}}$.i + 1;
else
    step.i = -1;
    map boundary.i = -1;
    cam block.i = ray block.i;
end
if $ray.i \neq 0$ **then**
    tmax.i = ($\text{cam}_{\text{block}}$ × l_{cell} - $\text{pos}_{\text{camera}}$)× C_m.i;
    Δ.i = l_{cell} × step.i × C_m.i;
else
    tmax.i = 1000000;
end
while $ray_{block} \neq p_{pointcloud}$ **do**
    if $tmax.x < tmax.y$ **then**
        if $tmax.x < tmax.z$ **then**
            $\text{ray}_{\text{block}}$.x += step.x;
            if $ray_{block}.x == map\ boundary.x$ **then**
                return
            end
            tmax.x += Δ.x;
        else
            $\text{ray}_{\text{block}}$.z += step.z;
            if $ray_{block}.z == map\ boundary.z$ **then**
                return
            end
            tmax.z += Δ.z;
        end
    else
        if $tmax.y < tmax.z$ **then**
            $\text{ray}_{\text{block}}$.y += step.y;
            if $ray_{block}.y == map\ boundary.y$ **then**
                return
            end
            tmax.y += Δ.y;
        else
            $\text{ray}_{\text{block}}$.z += step.z;
            if $ray_{block}.z == map\ boundary.z$ **then**
                return
            end
            tmax.z += Δ.z;
        end
    end
    assign occupancy value;
end

3.5 Map Update

The probability of the map update rule has been documented in Probabilistic Robotics, Thrun [20]. Let m be a three-dimensional map, and the subscripts i, j and k denote indices of the map. For each cell, the value can be occupied, $p(m_{ijk}) = 1$, free, $p(m_{ijk}) = 0$, or unknown, $p(m_{ijk}) = 0.5$. The probability distribution of the map is given by the product over the cells:

$$p(m) = \prod_{t} p(m_{ijk}) \tag{3.7}$$

where $p(m_{ijk})$ is the occupancy of each cell. Let $z_{1:t}$ denote the sensor measurements from time 1 to t. The measurement z could be the distance from sonar, LiDAR, or stereo camera, and pose from feature points estimate from IMU, feature points, or VICON data. Here, the depth data is from the depth camera D435 and the state data comes from the tracking camera T265. Once the depth data is available, the grid map can be determined by calculating the probability of occupancy of each grid cell m_{ijk} given the measurements $z_{1:t}$. The first assumption made for the mapping is that each cell is independent of other cells. The occupancy grid map update to each cell:

$$p(m_{ijk}|z) \tag{3.8}$$

Then, the probability can be seperated via Bayes rule:

$$p(m_{ijk}|z) = \frac{p(z_{1:t}|m_{ijk})p(m_{ijk})}{p(z_{1:t})} \tag{3.9}$$

The terms in the expression, 1:t, can be represented as t with given 1:t-1 condition probability. The three terms in Eq.3.9 are related to time so they can all be represented as:

$$p(z_{1:t}) = p(z_t|z_{1:t-1}) \tag{3.10}$$

$$p(z_{1:t}|m_{ijk}) = p(z_t|z_{1:t-1}, m_{ijk}) \tag{3.11}$$

$$p(m_{ijk}) = p(m_{ijk}|z_{1:t-1}) \tag{3.12}$$

Substitute Eq.3.10, 3.11 and 3.12 into Eq.3.9:

$$p(m_{ijk}|z) = \frac{p(z_t|z_{1:t-1}, m_{ijk})p(m_{ijk}|z_{1:t-1})}{p(z_t|z_{1:t-1})} \tag{3.13}$$

The mapping also assumes that the world is static and the measurement is time-independent. Therefore, we can use the Markov chain here to represent the time-related recursive terms:

$$p(z_t|z_{1:t-1}, m_{ijk}) = p(z_t|m_{ijk}) \tag{3.14}$$

$$p(z_t|z_{1:t-1}) = p(z_t) \tag{3.15}$$

Applying Bayes rule again to Eq.3.14, Eq.3.13 becomes:

$$p(m_{ijk}|z) = \frac{p(z_t|m_{ijk})p(m_{ijk}|z_{1:t-1})}{p(z_t)} = \frac{p(m_{ijk}|z_t)p(z_t)p(m|z_{1:t-1})}{p(m)p(z_t)} \tag{3.16}$$

Log-odds is commonly used in mapping and can simplify the problem to one addition and subtraction, as follows:

$$\begin{aligned} log\frac{p(m_{ijk}|z_{1:t})}{\bar{p}(m_{ijk}|z_{1:t})} &= log\frac{p(m_{ijk}|z_t)p(m_{ijk}|z_{1:t-1})\bar{p}(m)}{\bar{p}(m_{ijk}|z_t)\bar{p}(m_{ijk}|z_{1:t-1})p(m)} \\ &= log\frac{p(m_{ijk}|z_t)}{\bar{p}(m_{ijk}|z_t)} + log\frac{p(m_{ijk}|z_{1:t-1})}{\bar{p}(m_{ijk}|z_{1:t-1})} + log\frac{\bar{p}(m_{ijk})}{p(m_{ijk})} \end{aligned} \tag{3.17}$$

Letting L denote $log\frac{p}{\bar{p}}$, Eq.3.17 can be represented as:

$$L_t(m_{ijk}|z_{1:t}) = L(m_{ijk}|z_t) + L(m_{ijk}|z_{1:t-1}) - L(m_{ijk}) \tag{3.18}$$

The term on the left-hand side is the new log-odds value of the current frame and can be considered as a new value, $L_{posterior}$. The first term on the right-hand side is the log-odds value based on the current measurement. This term is also called the inverse sensor model, which has been mentioned in the previous section. The second term is the recursive log-odds value from the first frame to the latest frame. Due to the time-independent assumption, it can be considered as an accumulation of past values, L_{prior}. The last term is the initial log-odds value and is 0 (probability = 0.5) in general cases and can be denoted as L_0.

$$L_{posterior} = L_{inverse\_sensor\_model} + L_{prior} - L_0 \tag{3.19}$$

To reduce memory usage, we choose the smallest data type in c, unsigned char (with value from 0 to 255), as the container of the occupancy grid map. This means the value of the map has to be between 0 to 255 and the log-odds value has to offset to fit in this region. L_0 offsets to 127 as the initial value of the map, 255 for occupied cells, and 0 for free cells. The program regards a map value of 127 to correspond to a cell of unknown state. The smaller range of log-odds value means it can respond to dynamic objects in the field of view faster than an unsigned integer type with the range from 0 to 65535.

As for converting into a binary map for further usages such as obstacle detection and visualization, the threshold can be tuned based on specific environmental factors. For example, in an environment with many small objects, the occupied threshold has to be lower. In this research, we chose 180 as the occupied threshold for 3-D visualization in the next chapter. The final map update rule is shown below and the distance weighting is described in the following subsection.

$$L_{posterior} = L_{prior} + DW \times L_{inverse\_sensor\_model} \tag{3.20}$$

$$\begin{aligned} where \quad & L_{posterior} = \text{logodds of current frame} \\ & L_{prior} = \text{logodds of previous frame, initial value:0} \\ & \text{DW} = \text{distance weighting} \end{aligned}$$

3.5.1 Distance Weighting

Distance weighting is used to compensate for the lower pixel density of further cells. This term is similar to the contribution factor in Nguyen [21], but the factor here is applying to log-odds values instead of probability directly. As the distance increases, the density of points within a cell reduces. This factor increases the weighting for further cells.

Fig.3.13 shows the distance weighting mechanism. Just as the luminosity of a point light source is inversely proportional to the squared distance from it, so too is the point cloud density.

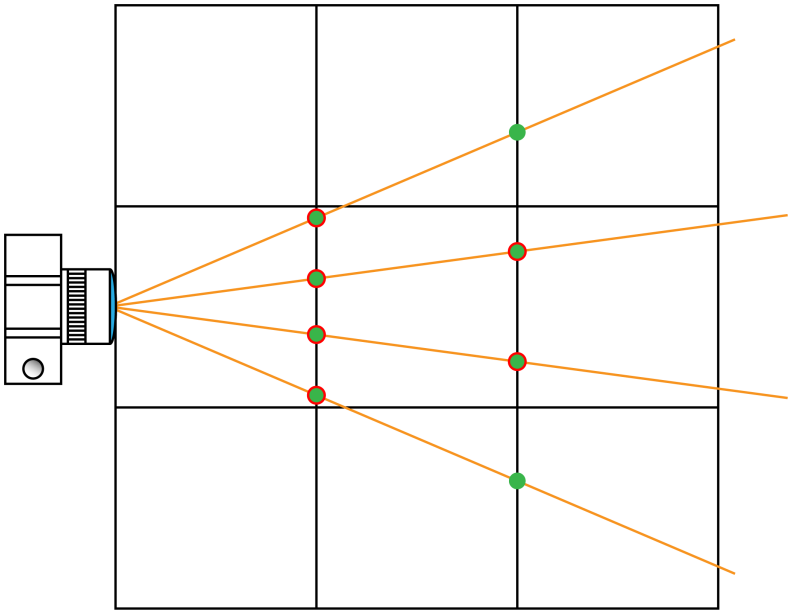


Figure 3.13: Distance Weighting. The dots can be considered as the projection pixel and those with red boundary are the pixels stay within the same row.

Based on our test, there are 86 × 86 pixels within 0.2 m^2 square region at 1 meter, which is the default cell size of the mapper we used. This can be easily transformed into different cell sizes by geometry. Also, the resolution downsampling decreases the raw resolution so the pixel density has to be added back to the distance weighting. The equation for distance weighting can be expressed as:

$$DistanceWeighting(DW) = \frac{distance^2 \times downsamplingratio^2}{pixeldensity^2} \tag{3.21}$$

Algorithm 3: Map update rule

Input: baseline, focallength(pixel), subpixel, len_{ray}, $\text{z}_{\text{measured}}$ Δ d

Δ z = $\text{z}_{\text{measured}}{}^2$ $\times$ Δ d / baseline / focallength;

$\sigma = \Delta \text{ z} \times \text{len}_{\text{ray}}/\text{z}_{\text{measured}}$;

if $z_{ray} \leq z_{measured}$ **then**

| $\text{P}_{\text{occu}} = \text{P}_{\text{min}}$

else

| $\text{P}_{\text{occu}} = 0.5$

end

$\text{P}_{\text{ISM;x,y,z}} = \text{P}_{\text{occu}} + (\text{k}/\sigma\ \sqrt{2\pi})+0.5\text{-P}_{\text{occu}})\times\exp(\text{-}0.5(\text{z}_{\text{ray}}\text{-z}_{\text{measured}})^2/\sigma^2)$;

$\text{L}_{\text{ISM;x,y,z}} = \text{P}_{\text{ISM;x,y,z}}/(1\text{-P}_{\text{ISM;x,y,z}})$;

$\text{L}_{\text{posterior;x,y,z}} = \text{L}_{\text{prior;x,y,z}} + \text{K}_{\text{distance}} \times \text{L}_{\text{ISM;x,y,z}}$;

3.6 Map Shifting

This is an optional function depending on the dimension of the environment. For an area of unknown size, this function would prevent the map from growing unboundedly, which would quickly make the mapping task intractable.

The first step is to set up a trigger boundary in the dynamic region like the red frame in Fig.3.14 shown. The dynamic region is in a local frame that will move with the vehicle. This will be explained later. The trigger boundary is a fence in three-dimensional space which when the camera passes through it, prompts the map to shift. Once the camera hits the trigger boundary, the map shifting process will activate and the shifted map will be saved into an array. The shifting array will be explained later.

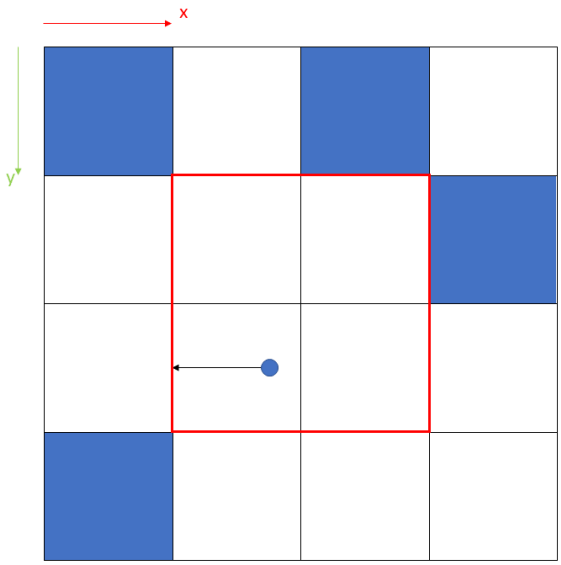


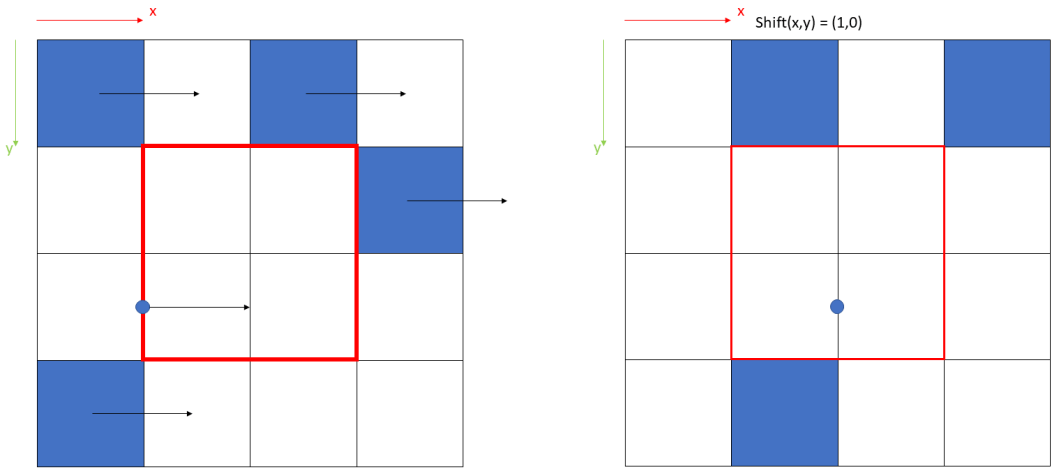
Figure 3.14: Map Shifting trigger region

Next, the whole grid map shifts one unit in the opposite direction of the boundary. For example, if the left boundary is hit, the whole occupancy grid map will shift toward the right and a new empty column will be shown on the leftmost side of the map as shown in Fig.3.14.

Fig.3.16(a) is the same map as Fig.3.14 but in the global frame perspective, where capital X and Y are global frame coordinates. The yellow region is the dynamic region that will move with the vehicle. Namely, this is the only region that could update the map.

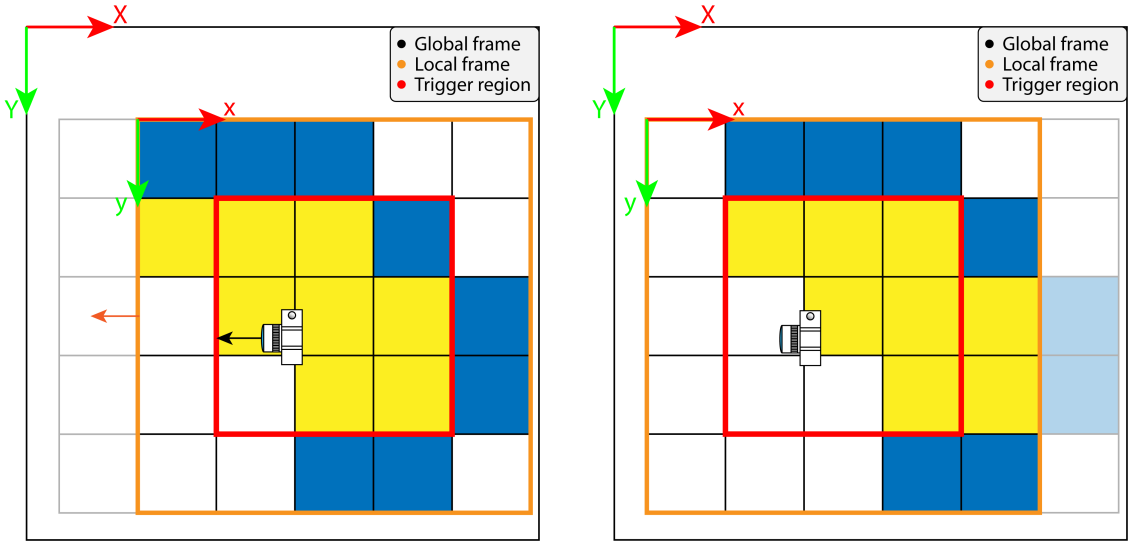
After the map shift happens, as per the process above, the dynamic region shifts

with the vehicle. The cells outside of the region are eliminated from the local map and can be stored in another memory location, or removed permanently (depending on the specific requirements). The eliminated cells are shown as light blue boxes in Fig.3.16(b).



(a) Map Shifting step.2 (b) Map Shifting step.3

Figure 3.15: Map shift in local frame. The blue cells are the occupied cells. The red square is the trigger region. The blue dot is the camera.



(a) Map shifting step.1 (b) Map shifting step.2

Figure 3.16: Map shift. The red square is the trigger region. The uppercase is the global frame coordinates and the lowercase is the local frame coordinates.

Chapter 4
System Integration and Test

4.1 System Coordinates

The coordinate system used for mapping corresponds to that of the D435, namely, right-down-forward. Here, the positive x-axis points to the right, the positive y-axis points down and the positive z-axis points forward.

The coordinate system for the T265 is the same as that defined in OpenGL. The positive x-axis points to the right, the positive y-axis points up and the positive z-axis points backward. Here, we convert the coordinates of the T265 to the D435 default coordinates for mapping. The coordinates of the two cameras are shown in Fig.4.1.

Initially, the local and global frames coincide. The initial position of the camera system is offset to the center of the occupancy map. The trigger region of the map shifting centers at the camera with the dimension of 1-by-1-by-1 meter (0.5m to trigger the virtual fence in all directions).

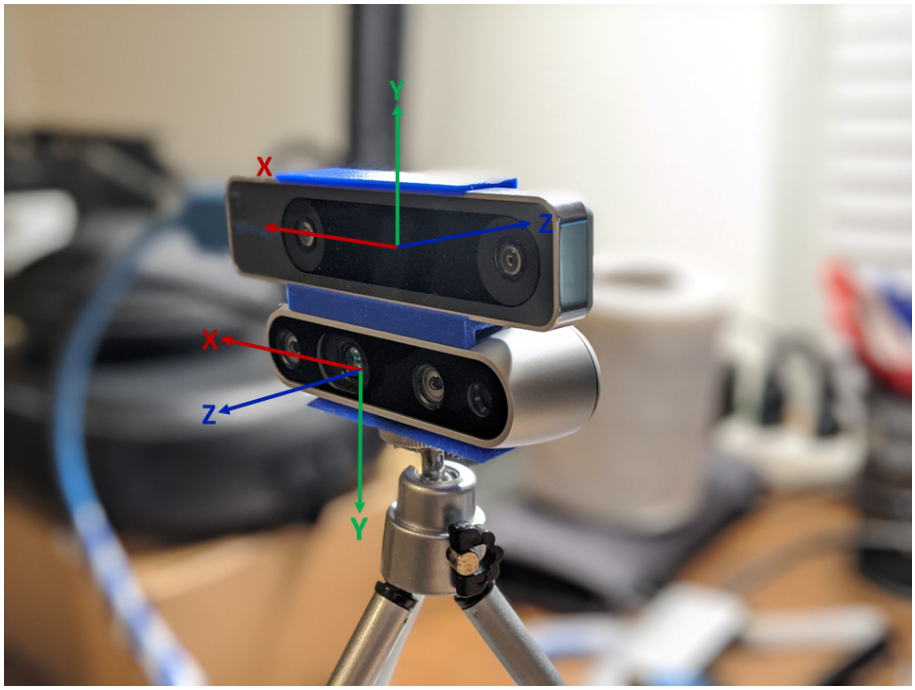


Figure 4.1: Coordinates of the D435 and the T265

4.2 Hardware Setup and Parameter settings

The Intel NUC computer is equipped with an 8th-gen i7-8550U CPU, 8GB RAM, and 128GB SSD. The cameras are Intel RealSense D435 and RealSense T265 using the librealsense API v2.31.0, and they are connected to the NUC with USB3.0.

The mapping parameters of the following experiment are listed in Table.4.1.

Table 4.1: Mapping parameter

Parameter	**Value**
Map Dimension	100×30×100 cell 20×6×20 m
Cell Dimension	0.2 m 5 pixel (ray tracing)
Map Shift Trigger Region	±1 m
D435 Resolution	848×480 pixel
D435 Depth Stream Rate	30 fps
Depth Resolution Downsampling Ratio	8 (default)

4.3 Experiments

4.3.1 Obstacle Detection Experiment

The obstacle experiment was tested with different obstacle dimensions, as Fig.4.2 shows. The true distance between the boxes and cameras was measured with a tape measure.

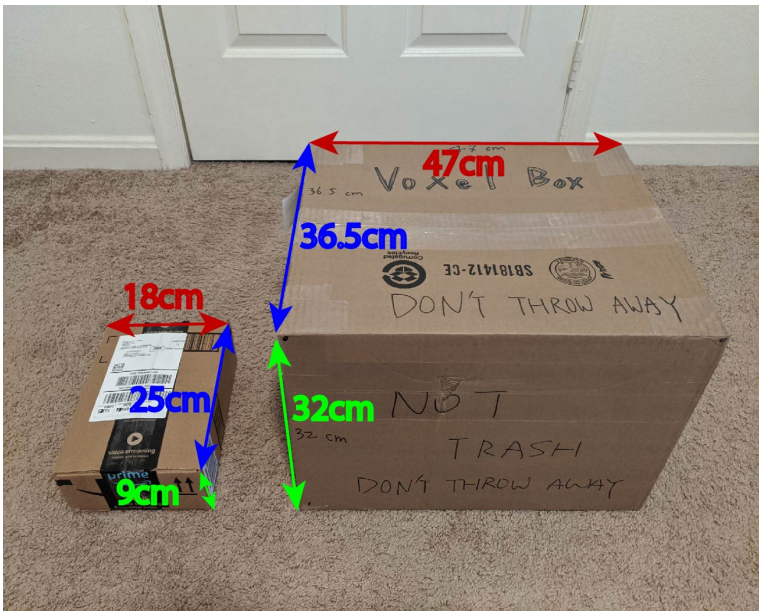


Figure 4.2: Tested obstacles

The experimental setup is shown in Fig.4.3, where the obstacles were placed 2m in front of the cameras. In this test, both cameras and the obstacles are at fixed positions and the maps are updated at least 100 times.

In Fig.4.4, the RealSense cameras, D435 and T265, are at the front end of the H-450 framewheel quadcopter, the Intel NUC is at the rear side of the vehicle to balance the C.G, the Pixhawk flight controller is in the middle of the vehicle, and the battery is on the bottom side of the vehicle to lower the C.G for stability.

4.3.2 Indoor Scene Experiment

The second experiment was implemented in an indoor environment which contained many obstacles. In this experiment, the precision of the vehicle state estimate from the T265 is critical since it has to continuously update the attitude and position of the vehicle precisely to ensure accuracy of our map. We checked the resultant occupancy map and compared it to the floor plane.

Figure 4.3: Obstacles setup in scene

Figure 4.4: Mapper test bed

4.4 Results

4.4.1 Obstacle Detection

4.4.1.1 Large Box

The camera view is shown in Fig.4.5 and the 2D mapping result is shown in Fig.4.6. The red dot is the camera's true position and the green dot is the box's true position. Since both the width and height of the box is larger than our defined grid cell size, 0.2m, the background wall is fully obstructed by the box and has no update results. The top

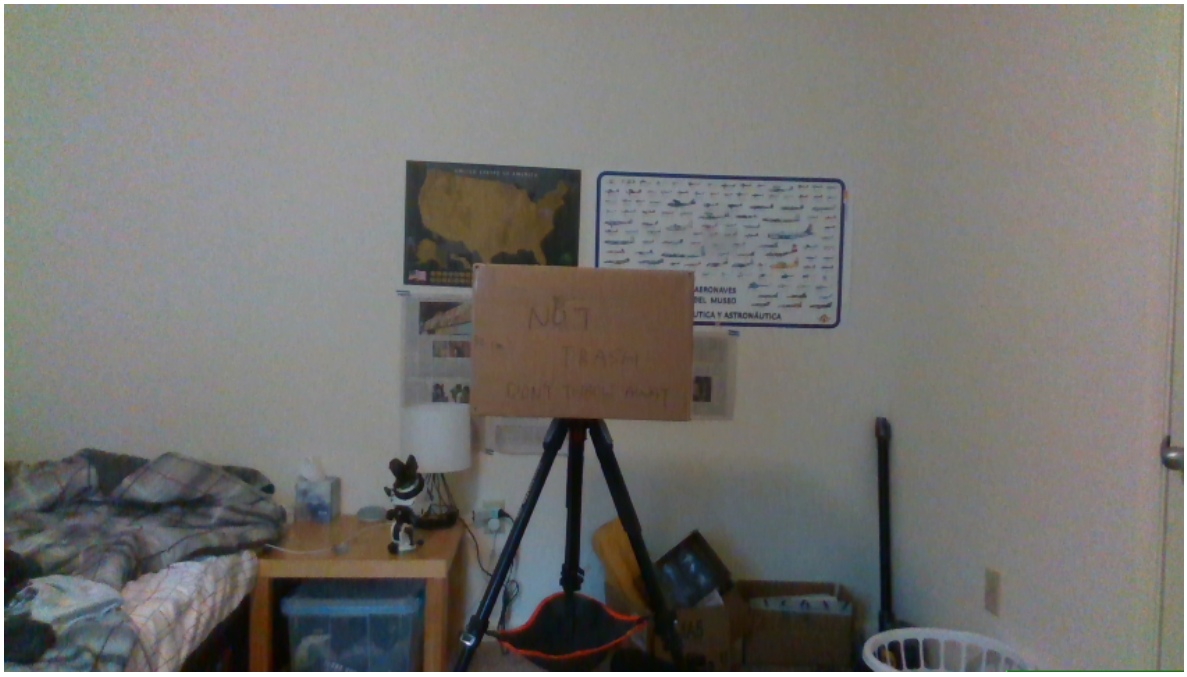


Figure 4.5: Large box setup (47×36.5×32 cm^3)

right figure is the log-odds distribution at the optical axis and it shows the result of iterative updates by the inverse sensor model. The bottom figure shows the 2D log-odds distribution at the camera height layer. The cells outside of the field of view are white (top-left figure) or cyan color in the bottom figure (corresponding to a log-odds value of 127).

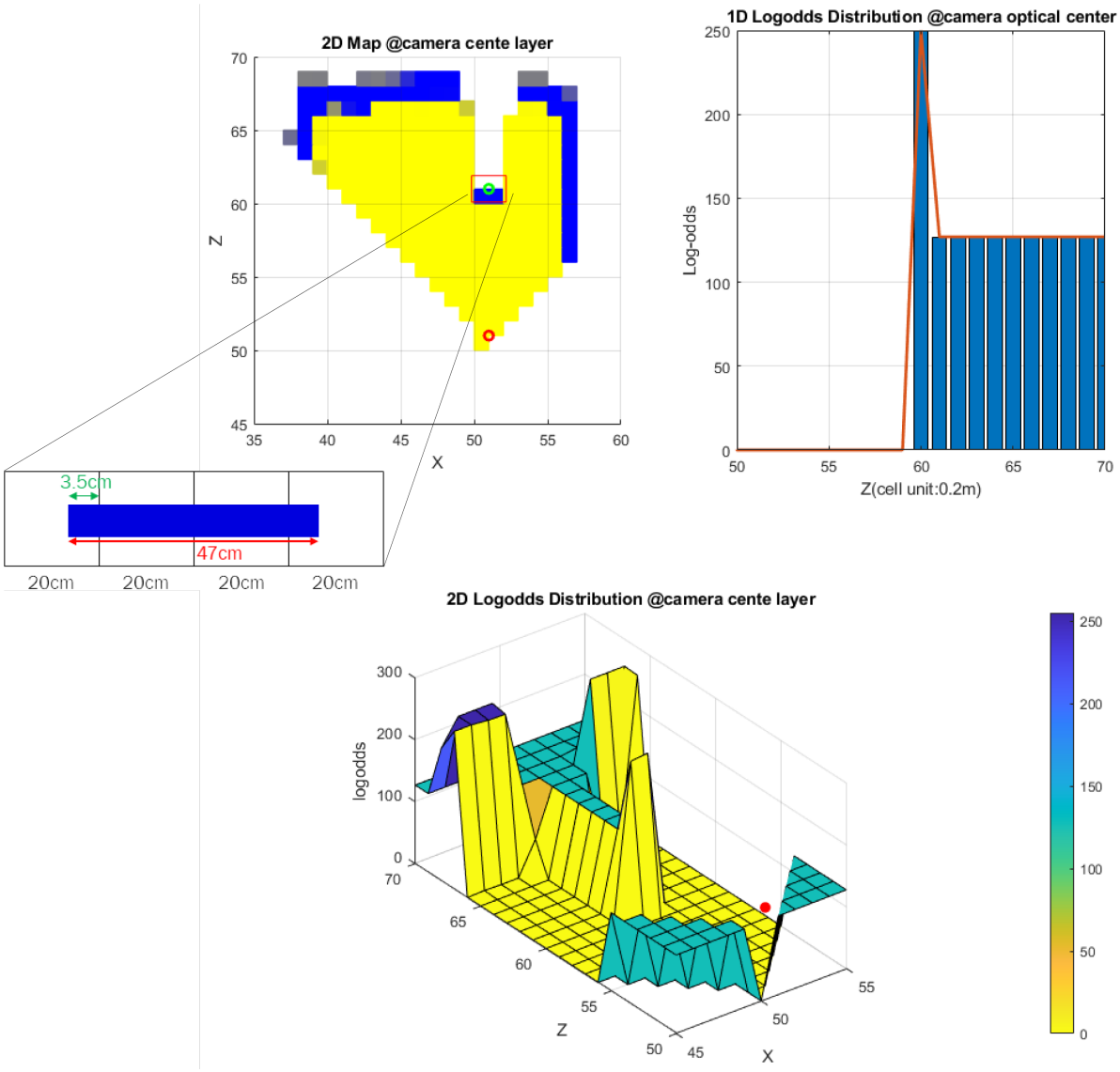


Figure 4.6: Large box mapping result. The top left figure is the 2D map, the blue square cells are occupied, yellow are unoccupied, and white are unknown. The red circle is the camera position and the blue circle with the red rectangle is the box position and its dimension. The top right figure is the log-odds distribution along the optical axis. The bottom figure is 2D log-odds distribution, where the vertical axis represents the corresponding log-odds value. The reason that the box occupies two cells is shown in the top left figure. There are only 3.5cm of the box within the first cell on the left and the rest of the spaces of the cell are dominated by rays.

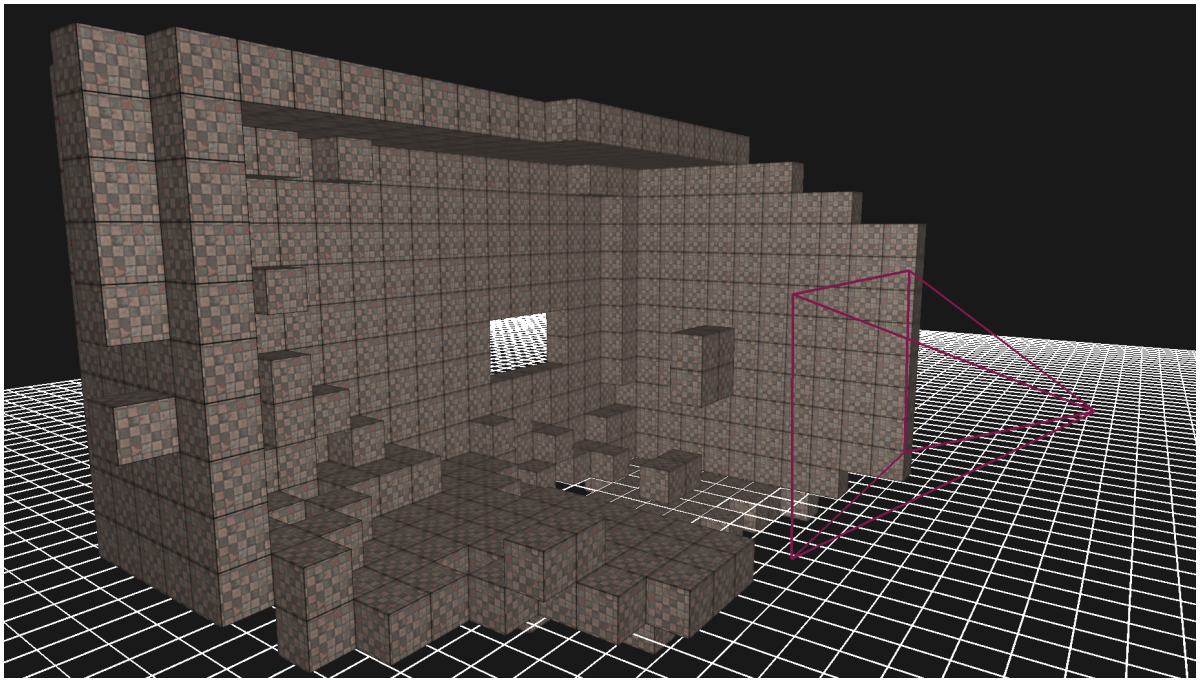


Figure 4.7: Large box 3D Map, red lines:field of view with 1m depth

4.4.1.2 Small Box

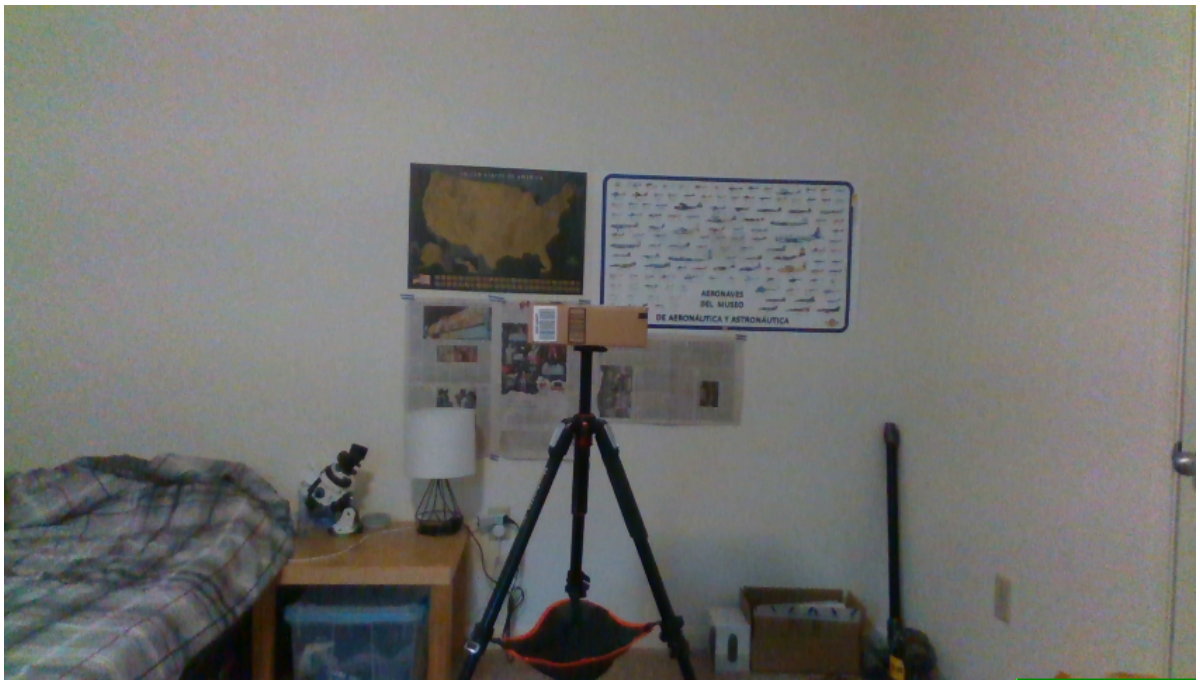


Figure 4.8: Small box setup(25×18.5×9 cm^3)

The camera view is shown in Fig.4.5 and the mapping result is shown in Fig.4.6. Unlike the large box, only the width of the small box is slightly larger than the cell size. Therefore, the box can barely block the background wall to update those cells on the map. Fortunately, the map used two occupied blocks to show the small box and we have enough confidence to detect the small box.

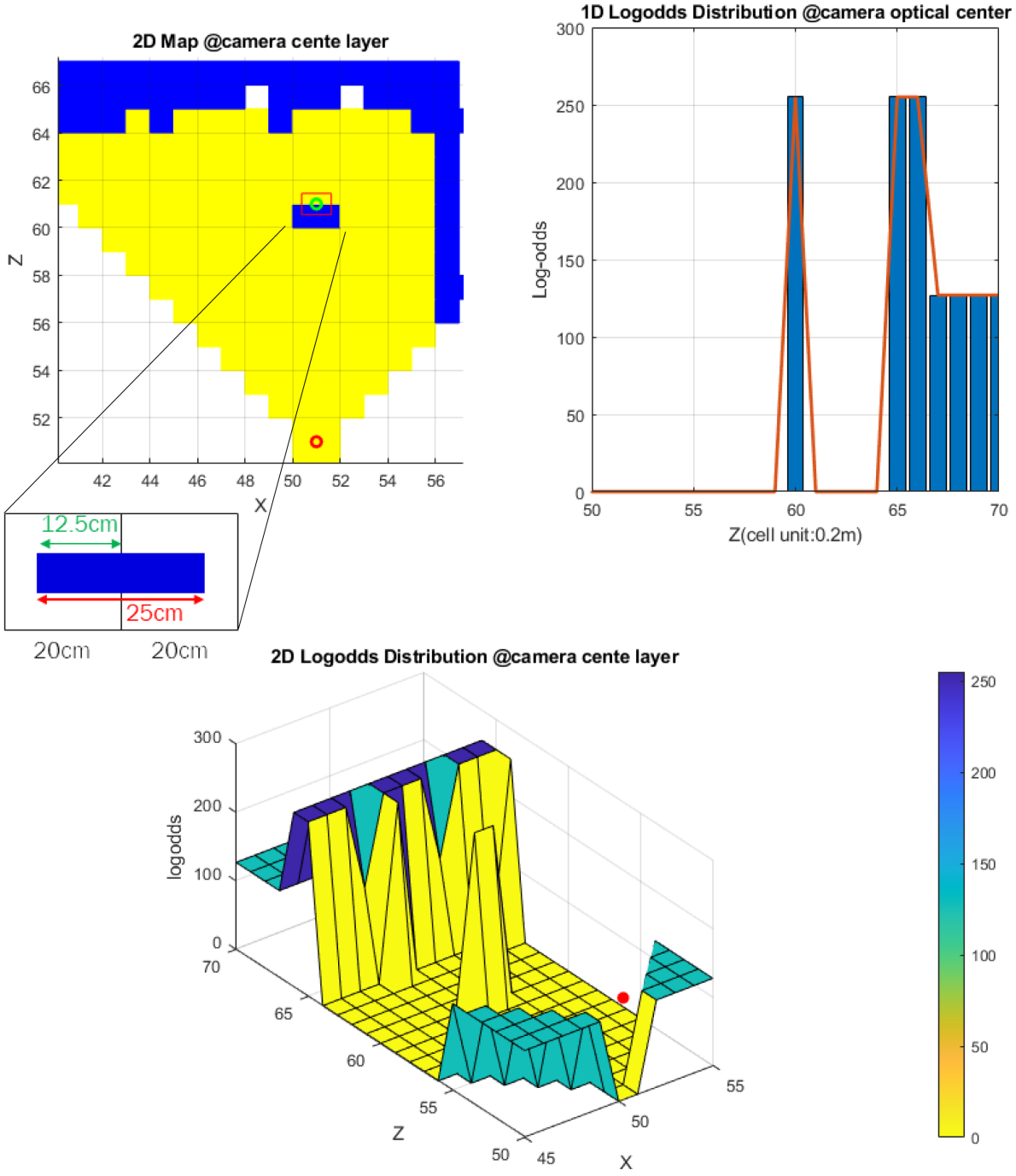


Figure 4.9: Small box mapping result. The notation is the same as Fig.4.6. The small box also occupies two cells even though the dimension is 22cm smaller. The reason is shown in the top-left figure that 12.5cm of the small box is within the cells on each side, which is more than half of the cells. Therefore, the cells eventually register as occupied.

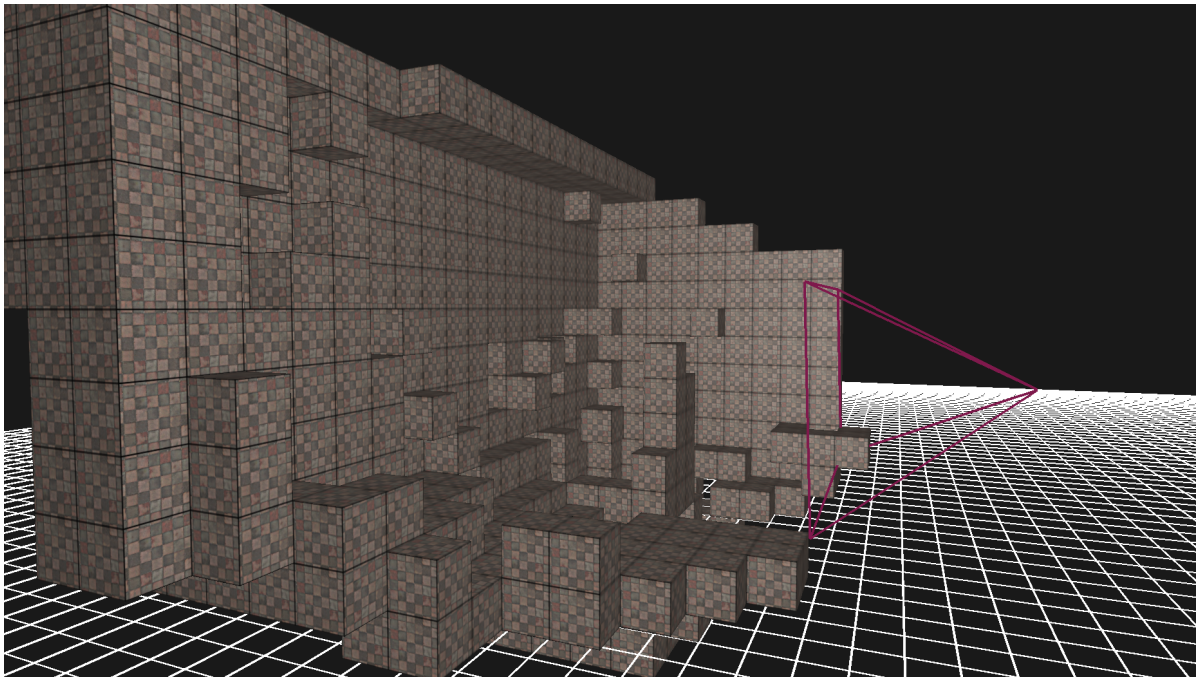


Figure 4.10: Small box 3D Map, red lines:field of view with 1m depth

Comparing the above two cases, we can see that the dimension of the box and the cell affect the final occupancy value. For the large box with 47cm width, it cannot occupy a total of four cells (as seen in Fig.4.6). Each side of the box is 23.5cm in width and slightly larger than a cell. That is to say, the box fully occupied the two center cells, but only partially occupies the two outer cells (with 3.5cm of its width). This means that those outer cells are deemed unoccupied, as rays overwhelm the slight obstacle occupancy in those cells. Conversely, the small box is only 25cm in width but can still occupy two cells. That is because half of the width of the small box, 12.5cm, is greater than half of the cell, 10cm. Therefore, the volume of an obstacle inside a cell contributes to the occupancy result of the cell. This characteristic means that margins must be used in path planning algorithms.

4.4.2 Scene Mapping

4.4.2.1 Apartment

The apartment was scanned to test the combination of the position estimation of the T265 and point cloud position of the D435. Since the attitude of the cameras were constantly changing during the mapping, we would expect mapping errors to increase. However, the result (shown in 4.11) looks precise due to the on-board SLAM and re-

localization function of the T265. The walls are solid and straight and the corner is 90 degrees. The floor plan of the apartment is shown in Fig.4.11 as well to compare with the scanned map. [27]

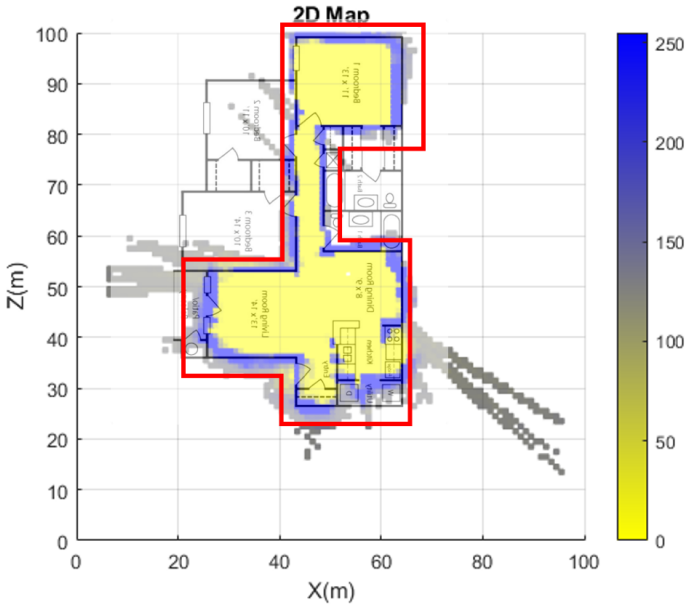


Figure 4.11: Scanned apartment map. The color bar on the right shows the occupancy in log-odds, 255 is occupied and 0 is free.

4.4.2.2 Aisle

During the test, we found that the resolution downsampling value affected aisle-like scenes a lot. Since the algorithm only considers the closest depth data in each sub-area, the further depth data of the aisle might be filtered out. The comparison is shown in Fig.4.12.

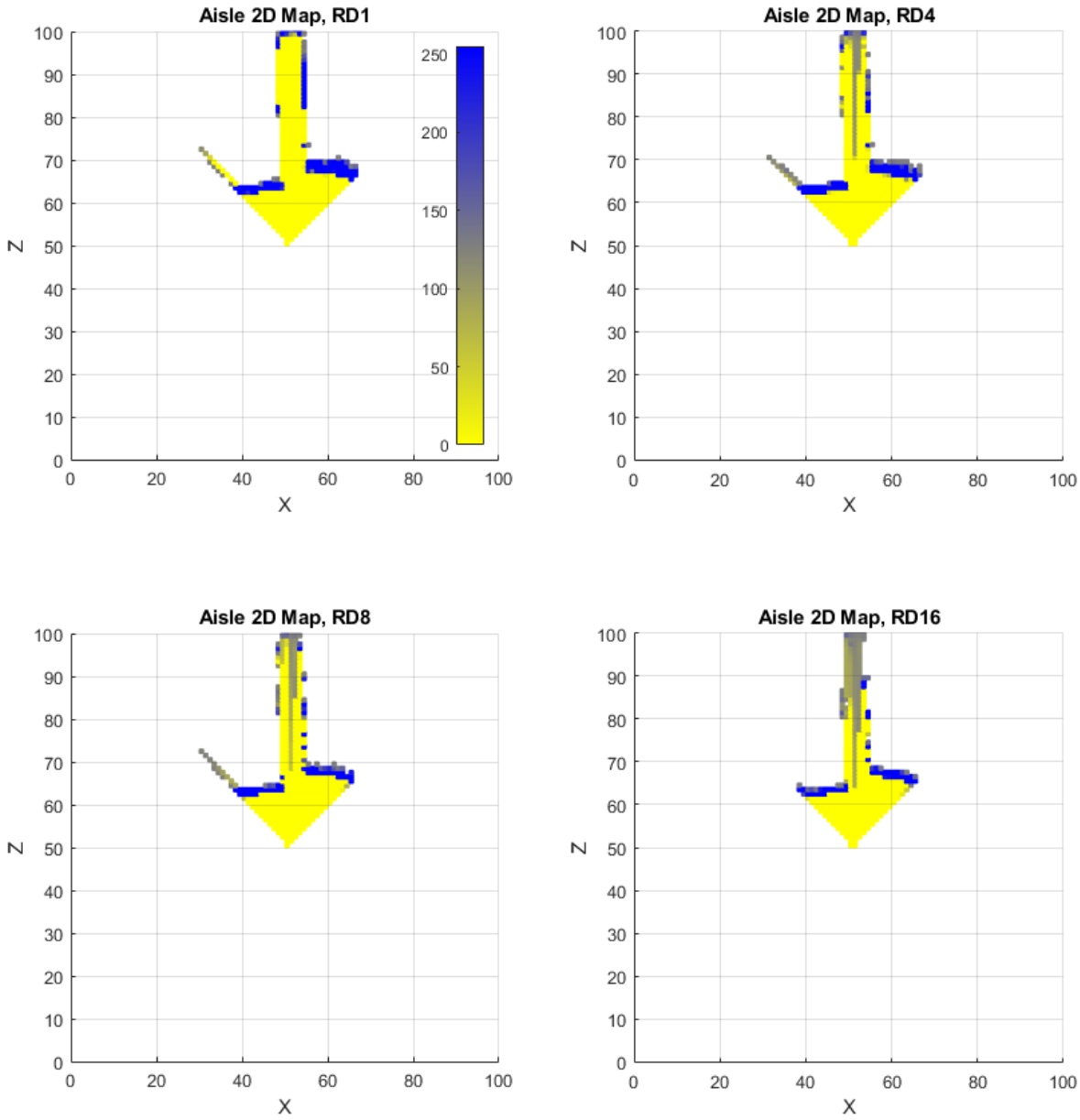


Figure 4.12: Aisle scanned with different resolution downsampling. The further cells are unable to be updated frequently with a higher resolution downsampling ratio.

4.4.3 Mapping Frame Rate

There are two different ray tracing methods mentioned above, so we test the time cost of both methods in different distances and conditions. The local mapping dimension affects the update rate.

The dimension of the tested map is $20\times6\times20\text{m}^3$ with a 0.2m cell size, $100\times30\times100$

cells. The raw resolution is 848×480 at 30 fps stream rate and the resolution downsampling is 8-by-8 pixels.

Table.4.2 lists the average time cost of each component of the mapper, and Table.4.3 lists the mapping time cost with different conditions on the Intel 8th-gen i-7 NUC. The sampling size is 250 frames and the outliers from the first couple frames are removed due to the cost of initialization.

Table 4.2: Functions time cost

Function	**Time Cost(ms)**
Resolution Downsampling (10x10)	5.48
Pointcloud Extraction	19.16
Colorized FilterDepth Image (Optional)	21.79
Fast Voxel Ray Traversal + Map Update	10.49
Line Drawing + Map Update	9.77

Table 4.3: Mapping time cost

Mapping Condition	**Time Cost(ms)**	**Frame rate(fps)**
3D + 2D Visualization	52.075	19.203
3D Visualization only	40.205	24.872
2D Visualization only	51.384	19.461
10×10 Resolution Downsampling (Line Drawing + No Visualization)	28.475	35.118
8×8 Resolution Downsampling (Line Drawing + No Visualization)	36.517	27.384
Full Resolution (Line Drawing + No Visualization)	665.320	1.503

Chapter 5
Conclusion

The main target of this study is to reconstruct a 3D map in real-time by using the RealSense depth camera and tracking camera for aerial robots. According to the results, the mapping accuracy is practical for detecting obstacle positions but not exact dimensions. Therefore, a one cell margin is required for path planning and obstacle avoidance. As for the performance, the mapping update rate is faster than expected. The map update rate with the settings mentioned in Table.4.3 allows for faster and safer indoor flights. Again, this research was tested hand-held rather than running the mapping on a vehicle for safety considerations. However, we did test the hardware on the vehicle to make sure the quadcopter is capable of carrying the chosen devices.

The RealSense D435 and T265 were selected for this research because of their light weight and accuracy, which improves our micro unmanned aerial vehicle's ability to operate autonomously. Although the stereo baseline of the D435 is only 50mm, the maximum depth range could reach up to 65m for obstacle avoidance and distances within 16m are accurate for mapping. Also, the active emitter of the D435 can perform well during indoor flights, even if the environment is relatively dim and featureless. For an environment with repetitive textures, tilting the camera slightly downward can solve the mismeasured depth reading. The high accuracy and re-localization feature of the T265 make the drift issue of state estimation barely noticeable when mapping. However, the precision of T265 is not constant. The initialization of the on-board mapping causes higher pose estimation error but the pose estimation improves after several scans (exploring the same area several times). The ease of use of the RealSense API also increases the development pace.

From Table.4.3, the resolution downsampling speeds up the mapping dramatically from 1.5fps to 35fps with a 10×10 ratio. This step is one of the most important parts in this research for achieving real-time mapping and a frame rate high enough such

that the map update rule enables us to detect moving objects. This approach brings a new challenge that has to be considered: the downsampling ratio affects the mapping distance on a narrow scene as shown in Fig.4.12.

The performance of the two ray tracing methods, line drawing and fast voxel ray tracing, were similar, with a difference of 0.1ms between them. At greater distances, the time difference may be greater, although the practical range of the D435 limits this distance such that the time difference would not be too significant. The line drawing method is faster but would miss some free data assignment. As for the fast voxel ray traversal method, it is slower but makes every cell along the rays is assigned correctly. For stereo cameras, unlike sonar sensors, the difference of the result is quite small due to the higher data density of the camera sensor. The missed cells can still be updated by other rays except for the rays close to the edge of the field of view.

The box detection test shows that there could be a one cell error around an obstacle. The volume of occupied cells could be either one cell larger or one cell smaller depending on the volume of an obstacle within a cell. This characteristic means that a one cell safety margin is required for path planning and obstacle avoidance.

The map shifting function is important for exploring a complex environment of unknown dimensions. Although it is hard to present the map shifting result, it helps the fixed size map move to scan further areas.

5.1 Suggestions for Future Research

Various ways have been proposed to improve the traditional occupancy grid map. From a UAV's point of view, robustness is one of the most important factors for filtering out outliers such as bad quality depth images and low confidence state estimation.

A lighter weight single board computer such as Raspberry Pi allows smaller MAVs to carry this hardware. Due to the lower computational power processor, it saves memory usage and reduces calculation loading from mapping. These properties make MAVs reconnaissance possible and practical in real-life applications. A way to reduce memory usage is to use octomap. The octree occupancy grid map starts from a large cell and recursively subdivides into eight octants. The advantage of this method is that it can save memory by condensing a large obstacle into a single cell. Also, the recursive update characteristic is similar to the map update rule and could work well together.

Also, the fixed resolution and fixed size map have to sacrifice either resolution for larger mapping area or map size for higher resolution. The dynamic size occupancy grid

map proposed in [28] can change the size of the map based on the velocity for different scenarios. This is workable for our system as well since the T265 can provide not only position and orientation data but also translational and rotational velocity.

The map shifting trigger region can be dynamic based on the velocity as well. For example, when the vehicle is moving forward quickly, moving the trigger region backward to have a further front clearance/field of view allows UAVs to have a longer reaction time for path planning and obstacle avoidance.

When exploring an indoor environment, finding the middle of the room and scanning could save time and battery life. This has been done on a multicopter with sonar sensors [29]. The high pose accuracy T265 can benefit from this approach by yawing to find the center of a space. The RealSense API allows for four depth cameras with 90 degrees field of view in each of them to work together to simulate a 360-degree range scanner in real-time. This multi-camera setup can speed up both finding the center of a room and the mapping process, though the weight would be heavier.

An RGB-D benchmark was proposed in Sturm [12]. A threshold using a similar RGB-D benchmark can filter out bad depth images from overexposure, blurred images, miscalculated depth data, etc., and improve the robustness for UAVs SLAM.

Appendix A
Depth Camera Distortion

During testing, we found one of the D435 depth cameras suffered from significant image distortion. The distortion is from the difference between the installed angle of the two camera sensors. Thus, the normal calibration tools Intel provides cannot solve the issue and a factory recalibration is required. The issue might happen to any RealSense depth camera with less severe distortion. In that case, we propose a simple method to calibrate the depth distortion by collecting depth distribution and fitting the curve to calibrate it.

A.1 Depth Camera Calibration

Some parameters of the stereo camera can be calibrated via RealSense software and uploaded to the onboard processor. For optical calibration, there is a software called Dynamic Calibrator, which is used to calibrate the optical parameters on each sensor like intrinsic, extrinsic, and distortion coefficients. The RealSense Depth Quality Tool uses deep learning to tune the algorithm parameters to match the true distance data and refine depth quality.

However, none of the software tools calibrate the depth distortion due to the installation angle of the sensors. For example, the depth image in this D435 Camera has a depth barrel distortion. The distance on the edge is around 10% further than the central area like Fig.A.1 shows.

Two identical cameras were tested, and one of them has a depth distortion issue. When pointing toward a flat wall, the camera has a barrel distortion. Although the problem only happens on one camera, this is still a possible issue for the toe-in installed stereo camera [30]. Except for this section, the rest of this work used the "good" camera rather than the heavy-distorted camera. It is interesting that the "poor" camera only

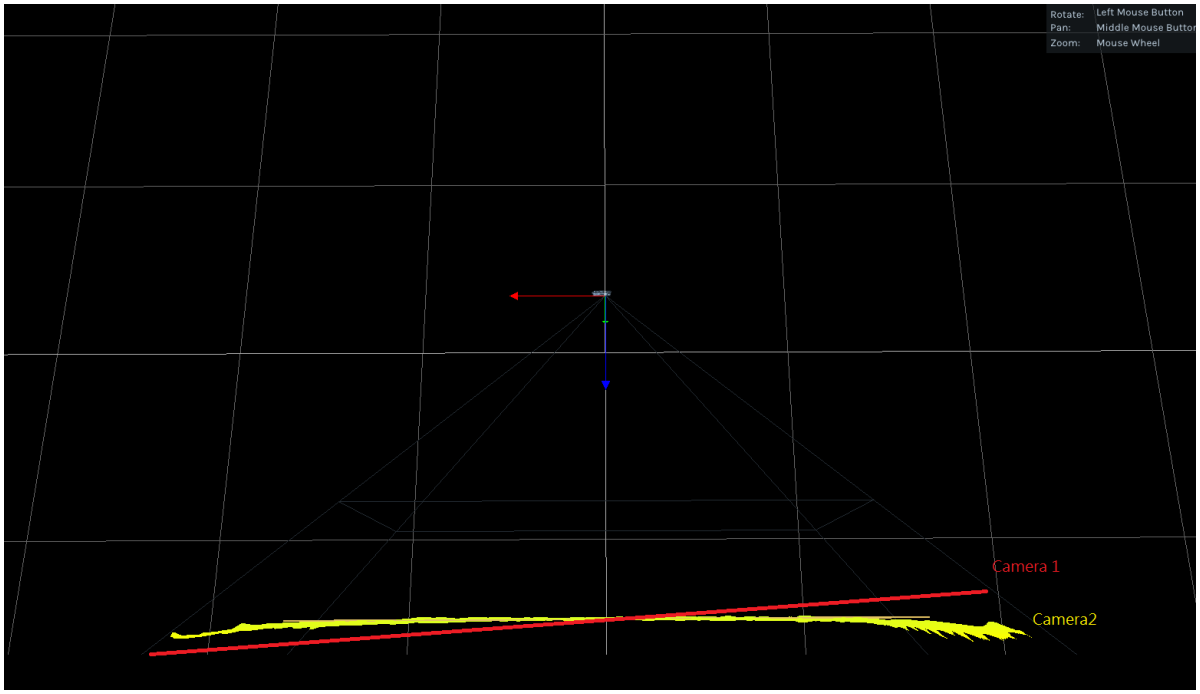


Figure A.1: Depth distortion against flat wall

has obvious depth distortion on the x-axis, so we only have to calibrate the x-z plane.

The first step of the depth calibration is pointing the camera toward a flat wall and measuring the true distance z_{true}. In this research, the calibration true distance is 1.4m away from the wall. Next, the calibration extracts the coordinates of points and calculates the 2$^{\text{nd}}$ order linear regression Eq.A.1:

$$Z_m = p_{1r}X_m^2 + p_{2r}X_m + p_{3r} \tag{A.1}$$

X_m and $Z_m(depth)$ are the raw coordinates of the points and p_{3r} is the offset of the linear regression vertex. Once the regression mode is calculated, the inverse of regression direction can calibrate the Z to the calibration plane.

The concept of calibration is to pull the linear regression curve back to a straight line to better match the true distance. The difference of p_{3r} and true distance, $p_3 = p_{3r} - z_{true}$ is the new 0 order coefficient. The new first and second-order coefficients can be obtained by switching the sign of the regression coefficients. Then the calibration formula can be expressed as:

$$\Delta Z = p_1X_m^2 + p_2X_m + p_3 \tag{A.2}$$

ΔZ is the offset of the $Z_{measure}$.

Finally, the calibrated depth $Z_{calibrated}$ can be obtained by substituting ΔZ into Z_m. The final equation for calibration in run-time code can be expressed as:

$$\begin{aligned} Z_{calibrated}(X_m, Z_m) &= Z_m - \Delta Z \\ &= Z_m - (p_1 X_m^2 + p_2 X_m + p_3) \end{aligned} \tag{A.3}$$

The calibrated result can be seen in Fig.A.2 and Fig.A.3. The tested calibration distance is 1.4m against a flat wall. There are three different sets of data in the result, including the raw depth data, the calibrated data based on current raw data, and the calibrated data based on the 1.4m calibration. A histogram is a useful way to show the distortion distribution (see Fig.A.2).

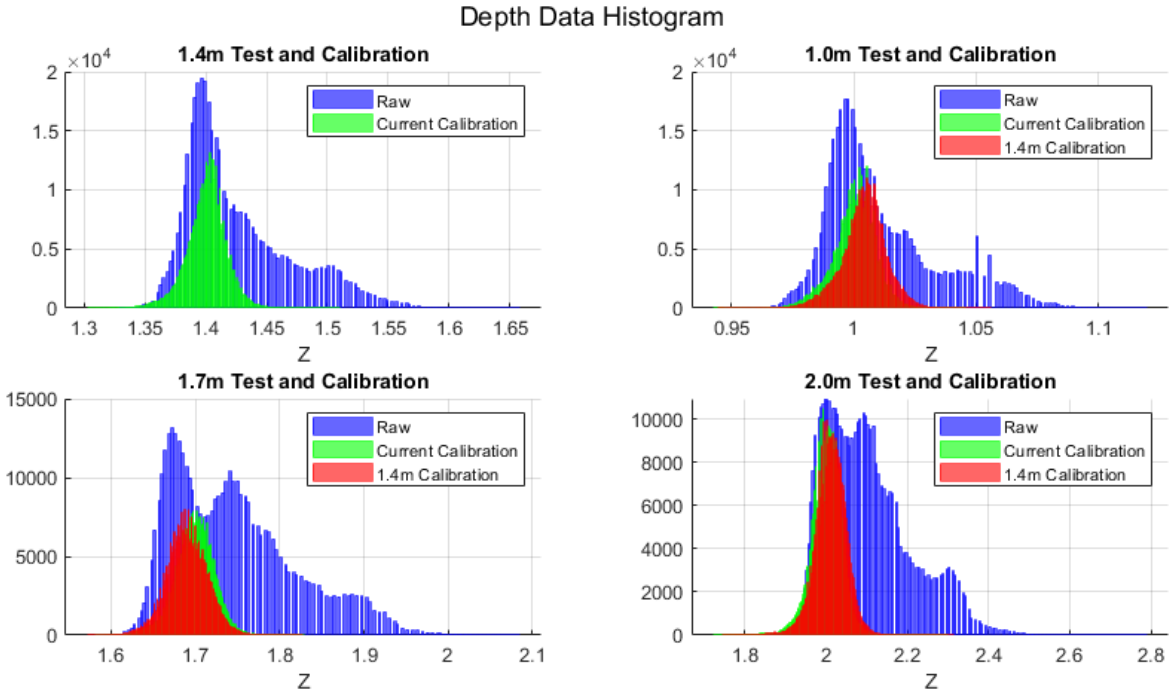


Figure A.2: Depth Data Histogram

Although the distribution of pre-calibrated data is slightly off from the current-calibrated data, the pre-calibrated data is still better when compared to the raw data distribution. Fig.A.3 is the top view of the point clouds. The result of the current calibrated and pre-calibrated data almost overlap with each other and the wall is flat enough for the following mapping.

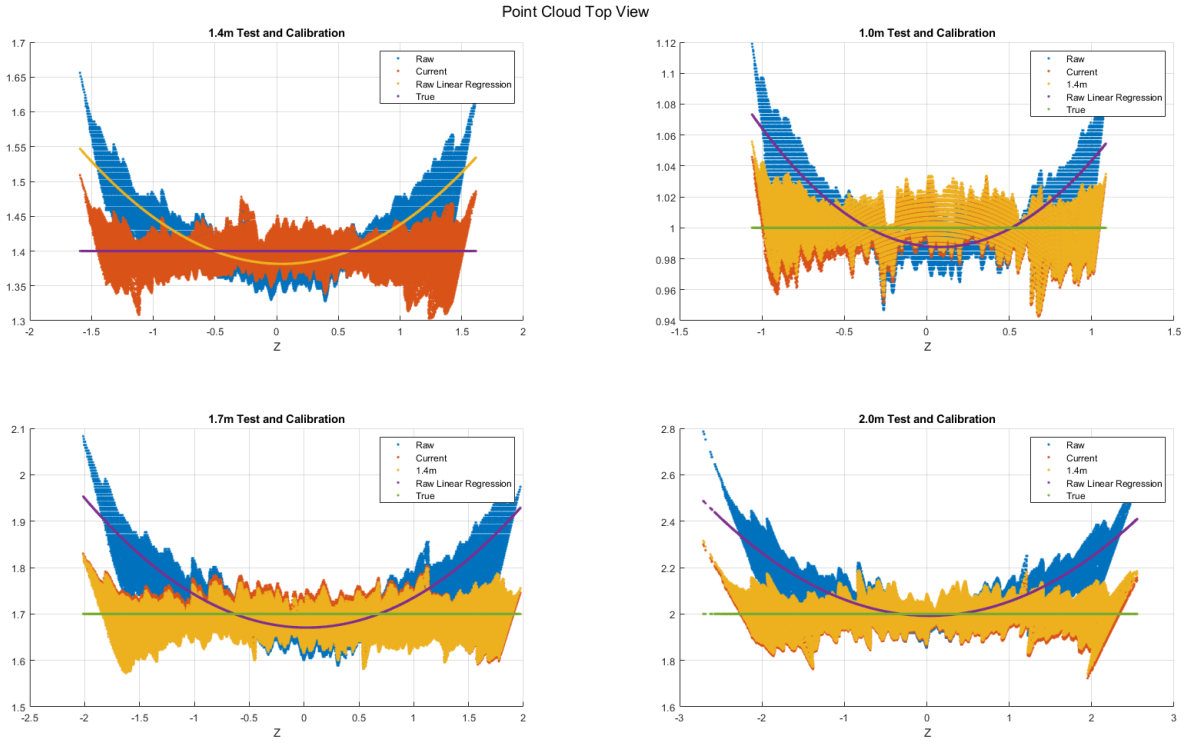


Figure A.3: Point Cloud Distribution, Top View

The depth distortion is more complicated and cannot just use a single linear regression to fixed all distance distortion. The distortion gets worse as the distance increases [30].

Bibliography

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

[2] Hess, W., D. Kohler, H. Rapp, and D. Andor (2016) "Real-time loop closure in 2D LIDAR SLAM," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp. 1271–1278.

[3] Li, R., J. Liu, L. Zhang, and Y. Hang (2014) "LIDAR/MEMS IMU integrated navigation (SLAM) method for a small UAV in indoor environments," in *2014 DGON Inertial Sensors and Systems (ISS)*, IEEE, pp. 1–15.

[4] Peng, S., W. Xinhua, and Y. Yurong (2017) "Real-time onboard mapping and localization of an indoor MAV using laser range finder," in *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*, IEEE, pp. 1612–1617.

[5] Lin, Y., J. Hyyppa, and A. Jaakkola (2010) "Mini-UAV-borne LIDAR for fine-scale mapping," *IEEE Geoscience and Remote Sensing Letters*, **8**(3), pp. 426–430.

[6] Celik, K., S.-J. Chung, and A. Somani (2008) "Mono-vision corner SLAM for indoor navigation," in *2008 IEEE International Conference on Electro/Information Technology*, IEEE, pp. 343–348.

[7] Davison, A. J., I. D. Reid, N. D. Molton, and O. Stasse (2007) "MonoSLAM: Real-time single camera SLAM," *IEEE transactions on pattern analysis and machine intelligence*, **29**(6), pp. 1052–1067.

[8] Mur-Artal, R., J. M. M. Montiel, and J. D. Tardos (2015) "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE transactions on robotics*, **31**(5), pp. 1147–1163.

[9] Engel, J., T. Schöps, and D. Cremers (2014) "LSD-SLAM: Large-scale direct monocular SLAM," in *European conference on computer vision*, Springer, pp. 834–849.

[10] Endres, F., J. Hess, J. Sturm, D. Cremers, and W. Burgard (2013) "3-D mapping with an RGB-D camera," *IEEE transactions on robotics*, **30**(1), pp. 177–187.

[11] Kerl, C., J. Sturm, and D. Cremers (2013) "Dense visual SLAM for RGB-D cameras," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, pp. 2100–2106.

[12] Sturm, J., N. Engelhard, F. Endres, W. Burgard, and D. Cremers (2012) "A benchmark for the evaluation of RGB-D SLAM systems," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, pp. 573–580.

[13] Bresenham, J. E. (1965) "Algorithm for computer control of a digital plotter," *IBM Systems journal*, **4**(1), pp. 25–30.

[14] Amanatides, J., A. Woo, et al. (1987) "A fast voxel traversal algorithm for ray tracing," in *Eurographics*, pp. 3–10.

[15] Elfes, A. (1989) "Using occupancy grids for mobile robot perception and navigation," *Computer*, **22**(6), pp. 46–57.

[16] Roth-Tabak, Y. and R. Jain (1989) "Building an environment model using depth information," *Computer*, **22**(6), pp. 85–90.

[17] Moravec, H. (1996) "Robot spatial perceptionby stereoscopic vision and 3d evidence grids," *Perception*.

[18] Andert, F. (2009) "Drawing stereo disparity images into occupancy grids: Measurement model and fast implementation," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, pp. 5191–5197.

[19] Kaufman, E., T. Lee, Z. Ai, and I. S. Moskowitz (2016) "Bayesian occupancy grid mapping via an exact inverse sensor model," in *2016 American Control Conference (ACC)*, IEEE, pp. 5709–5715.

[20] Thrun, S. (2002) "Probabilistic robotics," *Communications of the ACM*, **45**(3), pp. 52–57.

[21] Nguyen, T.-N., B. Michaelis, A. Al-Hamadi, M. Tornow, and M.-M. Meinecke (2011) "Stereo-camera-based urban environment perception using occupancy grid and object tracking," *IEEE Transactions on Intelligent Transportation Systems*, **13**(1), pp. 154–165.

[22] (2020 (accessed June 2, 2020)), "Pixhawk," Available at `https://pixhawk.org/`.

[23] (2020 (accessed June 2, 2020)), "Introduction MAVSDK," Available at `https://mavsdk.mavlink.io/develop/en/index.html`.

[24] Grunnet-Jepsen, A., J. N. Sweetser, and J. Woodfill (2018) "Best-Known-Methods for Tuning Intel® RealSense™ D400 Depth Cameras for Best Performance," *Intel Corporation: Satan Clara, CA, USA*, **1**.

[25] Kaufman, E., Taeyoung Lee, Zhuming Ai, and I. S. Moskowitz (2016) "Bayesian occupancy grid mapping via an exact inverse sensor model," in *2016 American Control Conference (ACC)*, pp. 5709–5715.

[26] Thrun, S. (2003) "Learning occupancy grid maps with forward sensor models," *Autonomous robots*, **15**(2), pp. 111–127.

[27] (2020 (accessed June 1, 2020)), "Park Crest Terrace Floor Plane," Available at `https://parkcrestterrace.com/property#gallery`.

[28] Marlow, S. and J. Langelaan (2010) "Dynamically sized occupancy grids for obstacle avoidance," in *AIAA Guidance, Navigation, and Control Conference*, p. 7848.

[29] Sobers, D., G. Chowdhary, and E. Johnson (2009) "Indoor navigation for unmanned aerial vehicles," in *AIAA Guidance, Navigation, and Control Conference*, p. 5658.

[30] Woods, A. J., T. Docherty, and R. Koch (1993) "Image distortions in stereoscopic video systems," in *Stereoscopic displays and applications IV*, vol. 1915, International Society for Optics and Photonics, pp. 36–48.