

OCTREE 3D VISUALIZATION MAPPING BASED ON CAMERA INFORMATION

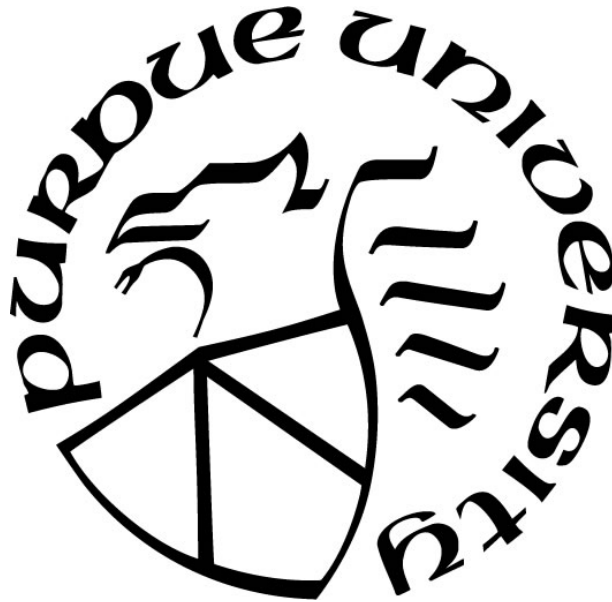
by
Benhao Wang

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Electrical and Computer Engineering



Department of Electrical and Computer Engineering

Hammond, Indiana

May 2020

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Nasser Houshangi, Chair

Department of Electrical and Computer Engineering

Dr. Dave Kozel

Department of Electrical and Computer Engineering

Dr. Donald Gray

Department of Electrical and Computer Engineering

Approved by:

Dr. Xiaoli Yang

Dedicated to my parents Wei Wang and Liwei Fu

ACKNOWLEDGMENTS

Here, I would like to thank my academic advisor Dr. Nasser Houshangi for his patient guidance and scientific insights. I have learned a lot and a lot of research methods since I started my thesis under his guidance. I believe this will be beneficial to my future career. Through this thesis project, I not only study engineering expertise but also study the morality, dedication, and enthusiasm of research.

I would also like to thank my other committee members for their feedback. I would also like to thank all the team members in my research for their help and brainstorming.

TABLE OF CONTENTS

LIST OF TABLES.....	7
LIST OF FIGURES	7
ABSTRACT.....	9
1. INTRODUCTION	10
1.1 Background.....	10
1.1.1 Simultaneous Localization and Mapping (SLAM).....	10
1.1.2 Visualization for Surface Reconstruction.....	12
1.1 Motivation and Objective	15
1.2 Organization of Thesis.....	16
2. VISUALIZATION USING OCTOMAP.....	17
2.1 Mapping in SLAM.....	18
2.2 Preliminary Data Processing.....	19
2.2.1 Pass-through Filter.....	19
2.2.2 Statistical Removal Filter	20
2.3 Point Cloud Downsampling.....	22
2.4 Increased Sampling.....	23
2.5 Octomap.....	24
2.5.1 Introduction to Octree.....	25
2.5.2 Occupancy Representation of Octree	27
3. SIMULATION	29
3.1 Gazebo	29
3.2 Build the Environment.....	30
3.3 Visualization of Map Points from SLAM Using RGB-D Camera	33
3.4 Interactive Markers Controlling the Mobile Robot	34
3.5 Simulation Result.....	35
4. VISUALIZATION EXPERIMENTS USING CAMERA INFORMATION.....	44
4.1 Experimental Setup.....	44
4.1.1 Hardware Platform.....	45
4.1.2 Software Platform.....	46

4.2	Visualization of the Right-angle Corridor Environment	46
4.2.1	Map Points Collection Using Camera	47
4.2.2	Map Points Data Processing	49
4.2.3	Publishing and Visualizing Point Cloud Data	52
4.3	Visualization of the Square Corridor Environment	55
4.4	Visualization of the Stairwell Environment.....	57
5.	CONCLUSIONS AND FUTURE WORK.....	59
	APPENDIX A. SOFTWARE ENVIRONMENT	61
	APPENDIX B. OFFLINE DATA VISUALIZATION	75
	REFERENCES	80

LIST OF FIGURES

Figure 1.1 A part of A Sunday Afternoon on the Ile de la Grande Jatte.	14
Figure 1.2. Minecraft.	15
Figure 2.1 Process for visualization.....	17
Figure 2.2 The Gaussian distribution.....	21
Figure 2.3 The principle of voxel filtering.....	22
Figure 2.4 Octomap with different resolutions.	25
Figure 2.5 Octree cube diagram.....	25
Figure 2.6 Octree statistical model.	26
Figure 3.1 3D scene of the second-floor corridor.	31
Figure 3.2 3D scene of the complex indoor environments.	31
Figure 3.3 Model of the jackal robot.	32
Figure 3.4 Details of the jackal robot.....	32
Figure 3.5 The corridor with the robot.....	33
Figure 3.6 Rviz with the robot.	34
Figure 3.7 The initial state of the simulation experiment. Figure (a) shows the position of the robot in Gazebo at the beginning of the experiment, (b) shows the scene seen by the RGB-D camera and the field of view of the RGB-D camera, and (c) shows the point cloud obtained by the ORB-SLAM2 method, (d) shows an Octomap constructed using point clouds.....	36
Figure 3.8 Intermediate state. Figure (a) shows the position of the robot in Gazebo in the middle of the straight corridor, (b) shows the point cloud, (c) shows the camera view field and the Octomap constructed using point clouds.....	37
Figure 3.9 Map building behind the corner. Figure (a) shows the movement of the robot in Gazebo, (b) shows the camera view field and the Octomap.	38
Figure 3.10 Octomap obtained from the simulation experiment.	38
Figure 3.11 Visualization errors that occur while turning.	39
Figure 3.12 Starting state of the simulation process for a more complicated environment in Rviz.....	40
Figure 3.13 The midpoint visualization for a complex room in Rviz.....	40
Figure 3.14 The final visualization for the complex room in Rviz.....	41
Figure 3.15 Targeted observation of people.	42

Figure 3.16 The comparison between the visualization results obtained from targeted observations of people(a) and the model in Gazebo (b).	42
Figure 4.1 The Jackal robot shown with both a bumblebee camera and an RGB-D camera.	45
Figure 4.2 The right-angle corridor on the second floor of the Potter Building.	47
Figure 4.3 The experimental results from RGB-D cameras in the right-angle corridor.	48
Figure 4.4 The left image is an overview of the map point cloud, and the right image is the front view of the point cloud.	48
Figure 4.5 The Octomap visualized based on the map point.	49
Figure 4.6 Map point cloud after preliminary processing.	50
Figure 4.7 Point cloud after downsampling.	51
Figure 4.8 Point cloud after increase sampling processed.	51
Figure 4.9 Octomap. (a) shows the result of a cube side length of 0.1m, (b) shows the result of a cube side length of 0.03m, (c) shows the result of a cube side length of 0.05m.	53
Figure 4.10 (a)The SLAM experimental results from RGB-D cameras in the square corridor (b)The map points obtained for the square corridor environment.	55
Figure 4.11 The point cloud from the square corridor environment after processing.	56
Figure 4.12 Visualization of the square corridor environment using Octomap.	57
Figure 4.13 The left image is the point cloud view after desampling. The right image is the result of the visualization using Octomap.	58

ABSTRACT

Today, computer science and robotics have been highly developed. Simultaneous Localization and Mapping (SLAM) is widely used in mobile robot navigation, game design, and autonomous vehicles. It can be said that in the future, most scenarios where mobile robots are applied will require localization and mapping. Among them, the construction of three-dimensional(3D) maps is particularly important for environment visualization which is the focus of this research.

In this project, the data used for visualization was collected using a vision sensor. The data collected by the vision sensor is processed by ORB-SLAM2 to generate the 3D cloud point maps of the environment. Because, there are a lot of noise in the map points cloud, filters are used to remove the noise. The generated map points are processed by the straight-through filter to cut off the points out of the specific range. Statistical filters are then used to remove sparse outlier noise. Thereafter, in order to improve the calculation efficiency and retain the necessary terrain details, a voxel filter is used for downsampling. In order to improve the composition effect, it is necessary to appropriately increase the sampling amount to increase surface smoothness. Finally, the processed map points are visualized using Octomap. The implementation utilizes the services provided by the Robot Operating System (ROS). The powerful Rviz software on the ROS platform is used. The processed map points as cloud data are published in ROS and visualized using Octomap.

Simulation results confirm that Octomap can show the terrain details well in the 3D visualization of the environment. After the simulations, visualization experiments for two environments of different complexity are performed. The experimental results show that the approach can mitigate the influence of noise on the visualization results to a certain extent. It is shown that for static high-precision point clouds, Octomap provides a good visualization. The simulation and experimental results demonstrate the applicability of the approach to visualize 3D map points for the purpose of autonomous navigation.

1. INTRODUCTION

This project explores how to visualize the environment from information collected by the robot using vision cameras. The geographic information is obtained using the SLAM technique to localize the robot and map the environment. The ORB-SLAM2^[49] is used specifically in this project as applicable for monocular, stereo, and RGB-D cameras. ORB-SLAM2 is a very accurate SLAM real-time solution that can be applied in various environments. In this chapter, the literature search is covered in Section 1.1, discussing the development of Simultaneous Localization and Mapping (SLAM) and the current related research in the area, followed by the developments in the visualization of 3D SLAM. The motivation and objective of this research are discussed in Section 1.2. The thesis organization is explained in Section 1.3.

1.1 Background

1.1.1 Simultaneous Localization and Mapping (SLAM)

SLAM means to estimate the position and orientation of mobile robots and to construct a model of the environment, essential and critical for autonomous navigation which is used in a large number of applications. The SLAM technique can improve the performance of localization using map information that is made by map building. SLAM provides the means to make a robot truly autonomous^[3].

In the past, the simplest method to find the robot position was to use a wheel encoder-based odometer^[35]. The amount of wheel rotation is combined with the motion model of the robot to find the current position of the robot relative to the global reference frame. Wheel odometers have some major limitations. Since the positioning based on the previously estimated position is incremental, the measurement error can accumulate over time, causing the estimated robot to pose to deviate from its actual position. Many factors cause errors as indicated in^[36].

To overcome these limitations, other positioning methods have been proposed, such as the use of ultrasonic, Global Positioning System (GPS), Inertial Measurement Unit (IMU)^[46], Light Detection and Ranging (LIDAR)^[37], and visual camera. Visual SLAM(VSLAM) is based on visual information, which means that the camera is the only sensor used. In the field of augmented

reality and robotics^[40], visual SLAM is an active research area. The reliability and robustness of current visual SLAMs can be trusted, even with large environmental dynamics and odometer errors^[39].

After the 21st century, scholars began to use Structure From Motion (SFM) to solve the problem of visual SLAM^[41]. This approach has achieved some success and has dominated the field of visual SLAM. In the SFM algorithm, the spatial and geometric relationship of the target is determined by the movement of the camera, which is a common method for 3D reconstruction. The SFM algorithm is an offline algorithm for 3D reconstruction based on various collected out-of-order pictures. First extract the focal length information from the picture, extract the image features, and calculate the Euclidean distance between the feature points of two adjacent frames of images to match the feature points. For each image matching pair, the epipolar geometry is calculated. There are feature points that can be passed down in a chain in such matching pairs and detected all the time, then a trajectory can be formed.

G. Klein and D. Murray proposed Parallel Tracking and Mapping (PTAM)^[42] in 2007. PTAM is a milestone in the field of visual SLAM. It is the first to propose and implement the parallelization of the tracking and mapping process. This is the first time in the visual SLAM to distinguish between the front-end (tracking and mapping) and the back-end (optimization), resulting in the design of many visual SLAM systems. Moreover, PTAM is the first approach to use nonlinear optimization rather than using traditional filters as a back-end solution. After PTAM, visual SLAM research has gradually turned to a back end dominated by nonlinear optimization.

PTAM has made great progress in monocular SLAM. But there are still problems caused by monocular cameras. Such as scale ambiguity, which is a common problem. It is caused by the triangulation calculation when the depth is obtained. The solution could be to add other sensors for optimization, such as IMU^[43]. Also, when the camera is rotated into position, the depth cannot be calculated correctly. Therefore, it cannot obtain an accurate trajectory.

In 2014, LSD-SLAM: Large-Scale Direct monocular SLAM^[44] introduces an algorithm suitable for large-scale environments. LSD-SLAM is a monocular SLAM algorithm based on a direct tracking method, which uses grayscale information from image pixels for positioning. It successfully overcomes the limitations of the feature point extraction method used by PTAM and can use all the information on the image. This method can achieve higher positioning accuracy

and robustness in an environment with few feature points and provide more environmental geometry information.

All the solutions mentioned above use only one camera as a sensor. The paper "CoSLAM: Collaborative visual slam in dynamic environments" ^[7] was published in 2013 and the vision-based SLAM problem of using multiple cameras in a dynamic environment was solved. These cameras can be moved independently and can be mounted on different platforms. All cameras work together to build a global map that includes the 3D position of the static background point and the trajectory of the pre-sport attraction. Experimental results show that this system can operate stably in a highly dynamic environment and produce more accurate results in a static environment. SLAM will always use several different types of sensors, and the powers and limits of various sensor types have been a major driver of new algorithms ^[18].

In the field of visual SLAM visualization, ORB-SLAM2 achieves the SLAM process with a different type of camera. The system works in real-time on standard CPUs in a wide variety of environments from the small indoor environment, to drones flying in industrial environments and cars driving around a city. The evaluation of the ORB-SLAM2 method achieves state-of-the-art accuracy, being in most cases the most accurate VSLAM solution ^[49]. Information obtained from a SLAM process is usually expressed as a series of distances and angles. After the coordinate system is established, these data map points can be displayed as points in the coordinate system space. A large number of points are merged to form a point cloud. In order to make better use of these point clouds, 3D visualization is usually required. In the field of SLAM, the realization of 3D map construction is a popular area explored by some researchers.

1.1.2 Visualization for Surface Reconstruction

In 2005, 3D SLAM technology based on Extended Kalman Filter(EKF) was researched ^[45], It uses planar features extracted probabilistically from dense 3D point clouds generated by a rotating 2D laser scanner. In the surface reconstruction part, It starts by decomposing the space into regular cells. After every raw data point has been associated with its corresponding cell, a plane is fitted to the points contained in every cell by using a Ransac algorithm ^[55] for segmentation with subsequent least-square fitting. The Ransac algorithm was chosen due to its simplicity and its robustness to outliers. However, the disadvantage of Ransac is that it does not have an upper limit on the number of iterations for calculating parameters; if you set an upper limit on the number

of iterations, the results obtained may not be optimal, and may even get wrong results. Ransac only has a certain probability to get a reliable model, and the probability is proportional to the number of iterations. Another disadvantage of Ransac is that the algorithm needs to manually set a threshold to determine whether the data points are suitable for the surface reconstruction model. At the same time, in terms of surface reconstruction, this algorithm uses square slices without thickness for reconstruction, and the size of the square is limited by the side length of the sample cube, and the accuracy is not high.

In 2006, Poisson surface reconstruction is explored ^[53]. Poisson surface reconstruction directly defines the approximate surface by defining that the value inside the model is greater than zero and the value outside the model is less than zero and then extracting the incremental zero-valued surface. The Poisson surface reconstruction algorithm combines the advantages of hierarchical and local methods. The implicit fitting method is used to transform the Poisson equation to obtain the implicit equation represented by the surface information described by the point cloud model. The equivalent value is that the reconstructed model has watertight closed features and has good geometric surface characteristics and detailed characteristics. However, Poisson surface reconstruction is often used to reconstruct dense point clouds of small objects. Poisson surface reconstruction is not efficient for sparse points in large scenes.

In 2016, Japanese researchers used the Robot Operating System (ROS) ^[5] platform and Unity3D to solve the simulation problem before the actual flight of UAV ^[25]. But cross-platform data communication often brings unstable connections. At the same time, Unity3D, like a game engine, needs to mount many additional modules when used as a visualization platform, resulting in inefficient use of computing power. After Rviz ^[6] came out, it became possible to complete all visual content only in ROS.

By 2018, researchers in India used the Greedy Triangulation Algorithm to reconstruct coastal dune surfaces ^[54]. When dealing with the geographic features of high-resolution point data, it is often more efficient to generate triangular surfaces. This study evaluated the performance of coastal dune areas using point-cloud data acquired from LiDAR sensors based on the Greedy Triangulation Algorithm. This method uses a spatial region growth algorithm. This method selects a sample triangle as the initial surface, continuously expands the surface boundary, and finally forms a complete triangular mesh surface. Finally, the original three-dimensional surfaces are determined according to the connection relationship of the projected point cloud. The topological

connection between the points, the resulting triangular mesh is the reconstructed surface model. However, the algorithm is suitable for the case where the sample point cloud comes from a continuous smooth surface and the density of the point cloud is relatively uniform.

Because the point cloud obtained through VSLAM in an indoor environment sometimes not dense and there may be discontinuities on the surface. Therefore, a new visualization method is needed. It is advantageous for the method to have high robustness and low requirements on host computing power. Point cloud data can only roughly show the environment information, but the advantage is that the point cloud can be optimized. In order to visualize the processed geographic information more vividly, the concept of the artistic mosaic is used in this project to optimize and voxel the scattered point clouds.

Mosaic visualization is very popular both in 2D map construction and 3D map construction. This stems from the artistic aesthetics of mankind. Impressionist paintings in the Renaissance actually used mosaics of countless color points to build the entire painting. Its representative is *A Sunday Afternoon on the Ile de la Grande Jatte*^[38]. As shown in Figure 1.1, the painting is made up of millions of points and is now housed in the Academy of Fine Arts in Chicago, USA.



Figure 1.1 A part of A Sunday Afternoon on the Ile de la Grande Jatte.

In three dimensions, the game Minecraft is a representative. The game world is composed of rough 3D objects—mainly cubes, fluids and commonly called "blocks"—representing various materials, such as dirt, stone, ores, tree trunks, water, and lava. The core gameplay revolves around picking up and placing these objects. These blocks are arranged in a 3D grid, while players can move freely around the world. Players can "mine" blocks and then place them elsewhere, enabling them to build things^[47]. As shown in Figure 1.2, the game uses a myriad of colorful cubes to create a vivid and beautiful picture.



Figure 1.2. Minecraft.

The map formed by SLAM are usually points cloud. When the points cloud map used not only for general purposes such as navigation, the maps need to be visualized in 3D. These 3D maps often require different accuracy when they are used for different applications. Fortunately, Octomap^[4] provides theoretical support for visualization. In this project, Octomap will be used as a theoretical basis to implement a cube-based map construction.

1.2 Motivation and Objective

Map visualization plays an important role in the current development of robotics. The vivid 3D map can not only bring a good control feedback experience for robot operators but also play an important role in areas such as automatic navigation. In order to make the point cloud obtained through VSLAM more vividly represent the indoor environments, point cloud data can be used to build a 3D map. Many real-time implementations of 3D SLAM have been implemented by Unity3D^[24], but Unity3D has strict requirements on needed computational power^[25]. In previous research, visualization methods using Unity3D were often used in desktop workstations. In the case of the Intel Xeon E5-2630 processor^[52], visualization on Unity3D still consumes more than 85% of the computing power.

The objective of this project is to create a method that can be applied to mobile platforms while saving computing power but can realize the visualization of the indoor environment. In this experiment, the visualization is done in the ROS environment. To achieve this, the camera information needs to be converted into corresponding map points cloud data, and the initially obtained point cloud data is filtered and processed multiple times. Then, Octomap is used to visualize the environment.

1.3 Organization of Thesis

The structure of this thesis is as follows. Chapter 2 describes the visualization approach using 3D SLAM map points. In Chapter 3, the simulation of SLAM using an RGB-D camera is implemented and results are discussed. Chapter 4 will introduce the experiment about visualizing offline data using the approach described in Chapter 2. Finally, conclusions and future research are discussed in Chapter 5.

2. VISUALIZATION USING OCTOMAP

The modern visual SLAM system can be divided into two parts: the front end and the back end. The front end extracts the sensor's data, which is used to build the preliminary model and perform state estimation and the back end does optimization based on the data provided by the front end. After the data has been properly optimized, it can be used for 3D visualization. The front end of the visual SLAM is related to the camera movement between adjacent images^[22]. Based on the image differences, locate the camera trajectory in the constructed map and extract the feature point positions required for map construction. Due to equipment and algorithm limitations, there is potential for errors in the constructed map.

The content of this chapter, as process for visualization, will be presented according to the following flowchart as shown in Figure 2.1 .

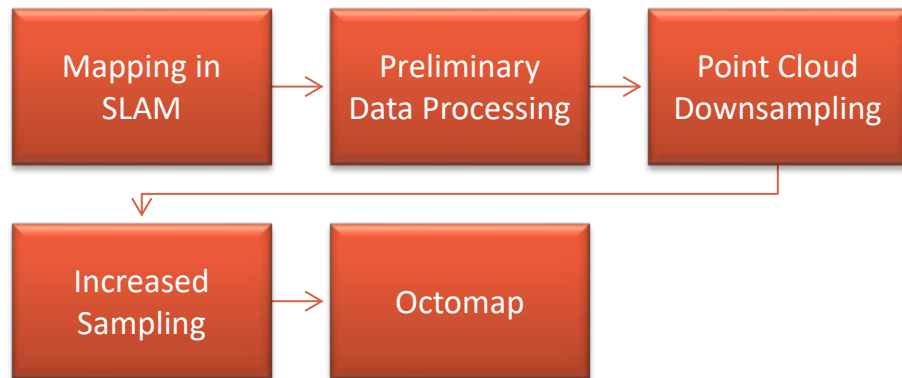


Figure 2.1 Process for visualization

The purpose of this project is to build an application that can visualize the environment from map points derived from the SLAM process. The data obtained by the experimenters through SLAM will be processed to make it suitable for 3D visualization. Therefore, the filtering stage is added to properly handle the possible residual noise and organize the point cloud structure. When the invalid information in the data is removed and the point cloud is at a suitable density, the point cloud is visualized using Octomap.

2.1 Mapping in SLAM

A map is a description of the environment. The form of the map has different suitable expressions depending on the application of the SLAM. Generally, it can be divided into two types: metric map^[17] and a topological map^[21]. The metric map emphasizes the precise representation of the positional relationship of objects on the map. The metric maps are the fixed-sized, occupancy-grid map. Topological maps emphasize the relationship between map elements compared to the accuracy of metric maps. It relaxes the map's need for precise location and removes the details of the map, yet it is not as good at expressing maps with complex structures.

Topological graphs are mostly used when the environment is very large to indicate the positional relationship of various small partitions in a large range. In this project, all experiments were performed in one environment. Therefore, no topology diagram is needed.

The Octomap in the ROS environment used in this study is more inclined to metric maps. Its advantage is that the density of the built model can be easily adjusted and converted. Another team is responsible for implementing the SLAM process with ORB-SLAM2 method and providing point clouds for visualization to this research. The RGB-D camera is used in this experiment. Its advantage is that the RGB-D camera can directly obtain depth information without the need for initialization like a monocular. The system initialization is more convenient, and the first frame is the key frame. Knowing the depth information, the initial 3D map can be constructed directly from the point.

The advantage of using ORB-SLAM2 is that it can quickly extract features and match them in real-time and can realize the functions of map reuse, loopback detection and relocation. ORB-SLAM2 is a method based on feature extraction. All the operations of the system are based on the detected features from images.

After determining the map points using ORB-SLAM2 based on RGB-D camera, it is often necessary to perform data processing on point cloud data to reduce the impact of noise. Octomap is used to visualize the processed data. The following section discusses the approach taken for preliminary data processing.

2.2 Preliminary Data Processing

Although the data obtained from the camera can roughly represent the approximate shape of the detected space, the accuracy is first limited by equipment used, operator experience, and environmental factors. There are also some theoretical limitations that cause the data obtained by the camera need to be filtered before being visualized^[19]. These phenomena may appear at the junction of the wall with low contrast or high reflection coefficient. Some noise points will inevitably appear in the point cloud data, which is a random error.

Also, due to external interference such as occlusion of obstacles, obstacles, and other factors, there are often discrete points in the point cloud data that are far from other points, that is, outliers. These outliers needed to be removed before the visualization phase.

The presence of these noises means that proper data processing is extremely valuable. As the first step of point cloud processing, the filtering process is to remove noise points, outliers, holes, and sometimes it is necessary to cut out areas that do not need to be visualized. The second step is to allow the point cloud data to be better processed by subsequent applications, which may require to downsample the point cloud.

Each point in the point cloud data set expresses a certain amount of information, and the denser a particular area, the higher the amount of useful information. The amount of isolated outliers is small, and the amount of information expressed is negligible. These processing methods not only make the data more readable but also reduce the computational load on the hardware to a large extent as the number of map points decreases. In this research in a preliminary processing phase, the collected data is mainly processed using two filters, a pass-through filter, and a statistical removal filter as explained next.

2.2.1 Pass-through Filter

The pass-through filter is the simplest type of filter that eliminates points that are not in the given range of values in the specified environment dimensions^[12]. The implementation of the pass-through filter is as follows: First, specify dimensions and the range of values for the environment. When processing data obtained from indoor SLAM, the filtering range is often set to the height estimated for indoor. After that, through each point in the point cloud, the point value is determined on whether the specified dimension value range. If the value of the map point is not in the range,

the point will be deleted. Finally, these remaining points become the new filtered point cloud. pass-through filters are not only simple but effective. This method is suitable for eliminating background noise.

For example, use a camera to scan an indoor environment to collect point clouds. Although the height of the indoor room is usually determined, the position of the ceiling is often not accurately showed in point clouds. The reason may be that the ceiling is too far away, the camera is not accurate, and the light intensity is insufficient. For these reasons, the collected points are often distributed on the altitude axis beyond the actual room height range. A pass-through filter can be used. The points within the range of the height axis will be retained, and the outliers can be quickly removed. After applying the pass-through filter, the point cloud data is processed again using a statistical removal filter as discussed in the next section.

2.2.2 Statistical Removal Filter

The primary function of the statistical removal filter is to remove sparse outlier noise. In the process of collecting point clouds, due to the influence of measurement noise, the noises points are sparsely distributed in the point cloud space. When visualizing point clouds, these noises can cause errors, such as unknown floating objects in the middle of empty corridors.

The main idea of the statistical filter is to assume that the average distance of all points in the point cloud and its nearest k neighbor points satisfy the Gaussian distribution. Gaussian distribution is a convenient model for quantitative phenomena in the natural and behavioral sciences. A variety of psychological test scores and physical phenomena such as photon counts were found to be approximately obeying a normal distribution. Although the root causes of these phenomena are often unknown, it is theoretically possible to prove that if many small effects are added together as a variable, then this variable obeys normal distribution. The normal density function is shown in equation (2-1),

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \quad (2-1)$$

where μ is the mean, and σ is the standard deviation of the normal density function. When the average distance between a point and its nearest k points is greater than the preset threshold, the point is determined as an outlier and deleted.

The implementation principle of the statistical filter is as follows: First, traverse the point cloud to calculate the average distance between each point and its nearest k neighbor points; secondly, the distance threshold can be determined from the mean and standard deviation. Calculate the mean μ and standard deviation σ of all the average distances between each point and its nearest neighbor points, then the distance threshold d_{\max} can be expressed as

$$d_{\max} = \mu + \alpha \times \sigma \quad (2-2)$$

where α is a constant scaling factor, which depends on the number of neighbor points. The Gaussian distribution is shown in Figure 2.2.

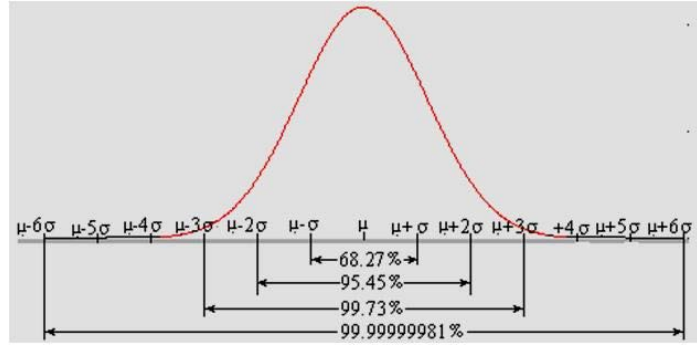


Figure 2.2 The Gaussian distribution.

As shown in Figure 2.2, 95.45% of the area is within the range of two standard deviations of (2σ) around the mean. 99.73% of the area is within the range of three standard deviations of (3σ) around the mean value. 99.99% of the area is within the range of four standard deviations of (4σ) and 99.9999% of the area is within the range of four standard deviations of (6σ) around the mean values.

Depending on the detection accuracy, α usually takes the value of 2, 3 or 4. It is generally believed that the denser the original point cloud, the smaller the value of α . In this project, the α is set to a value of 2. Finally, the point cloud is traversed again and the points with an average distance of k adjacent points greater than d_{\max} are eliminated.

After data preprocessing, a voxel mesh filter is used to downsample the point cloud as discussed next.

2.3 Point Cloud Downsampling

If a point cloud is collected using a device such as a high-resolution camera, the point cloud tends to be dense. The excessive number of point clouds can make it difficult for subsequent segmentation work^[29]. The voxel mesh filter can achieve downsampling without destroying the geometry of the point cloud itself^[30]. Figure 2.3 approximates the principle of voxel filtering.

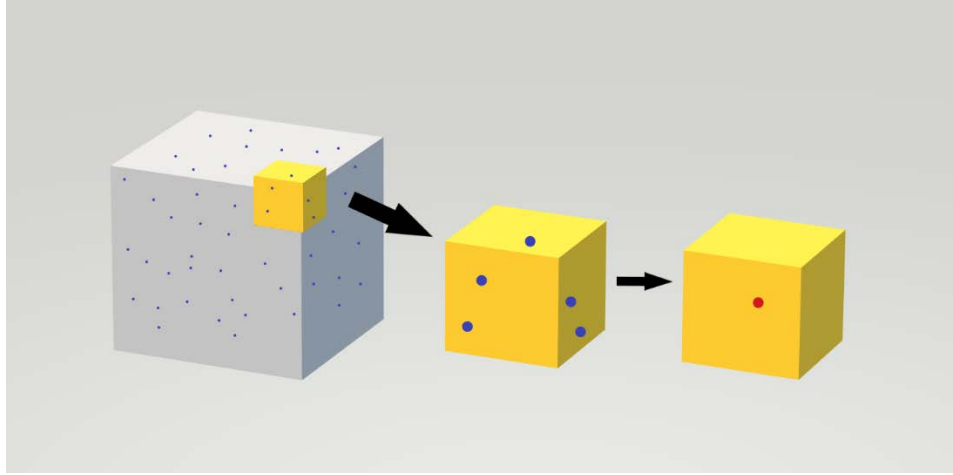


Figure 2.3 The principle of voxel filtering.

The concept of voxels is similar to pixels. Pixels are two-dimensional points, while voxels are small 3D spaces. As shown in Figure 2.3, the gray cube indicates that a 3D voxel grid is created on the input point cloud data. The points in the gray cube indicate the position of the points in the original data. Set a certain voxel value, that is, the entire space, that is, the gray cube, is divided into a number of small cube spaces with side lengths that are voxel values. The yellow cube is considered one of the spaces in all the tiny cube spaces. In a yellow cube, all existing points will be approximated by their centroids and represented by red points. Replacing points with centroids are slower than directly replacing points with voxel centers, but it can maintain macro geometry more accurately.

2.4 Increased Sampling

After downsampling using voxel filtering, the surface-displayed by the point cloud data is sparse. However, the lower point cloud density is very inconvenient when using Octomap for surface reconstruction.

The voxelization downsampling process will lead to uneven distribution of the point cloud. Direct visualization with the point cloud will make the surface not smooth or have holes. In order to build a complete wall model, the surface needs to be smoothed and the hole repaired. This problem can be solved by data reconstruction. In this part, based on the existing data, Moving Least Squares (MLS) is used to solve the problem of data resampling. The resampling algorithm reconstructs the missing surface by performing high-order polynomial interpolation on the surrounding data points^[48].

Compared with the traditional least squares method, the moving least squares method has two major improvements:

- (1) Least squares is a mathematical optimization technique. It finds the best function match of the data by minimizing the sum of squared errors. If the amount of discrete data is relatively large and the shape is complex, piecewise fitting and smoothing are also required. Using moving least squares for curve fitting can overcome the above disadvantages.
- (2) The concept of compact support is introduced. The value y at point x is only affected by nodes in the subdomain near x . This subdomain is called the influence area of point x . Nodes outside the influence area have no value for x influences. Define a weight function $w(x)$ on the area of influence. If the weight function is taken as a constant in the whole area, the traditional least square method is obtained.

By calling the calculation function of the MLS method, the point cloud library can directly implement incremental sampling of the point cloud. In the process of increasing sampling, it is very convenient to adjust the constant coefficients at will. Increasing the coefficient will cause the density of the points in the point cloud to increase. Although the increase in density will make the point cloud smoother, the details will also become more blurred. This aspect will be described in detail in Section 4.2.

2.5 Octomap

Using the point cloud to visualize the environment is simple, but there are some obvious flaws. The first flaw, the map form is not compact. Point cloud maps are usually huge, so a Point Cloud Data(pcd) file will be large. For a corridor-sized indoor area, thousands of space points are generated(Section 4.2). When performing high-resolution SLAM on complex indoor environments, nearly a million map point clouds are generated requiring a large amount of storage space(Section 4.4). Even after some filtering, the pcd file is still large. And the annoyance is that such a large map data is not required. The point cloud map provides a lot of unnecessary details. For dense point clouds on a large flat surface, these things are not worth paying special attention to, only a few points are needed to describe the plane features. Placing them on the map wastes space.

The second problem with using point cloud data is the way cloud data processing overlaps. When building a point cloud, in many cases, the robot will directly combine the detection results in the estimated pose. Especially in the process of camera mapping, when there is an error in the pose, it will cause a distinct overlap of the map. For example, if a chair becomes two. The way to deal with overlapping areas should be better.

The third issue is the difficulty of navigation. In many cases, when constructing a point cloud, the robot will directly combine the detection result with the estimated pose. Especially during the camera mapping process, the same scene will be repeatedly detected. The point clouds in some places are too dense.

The Octomap is designed for visualizing the environment. It can elegantly compress, update the map, and the resolution is adjustable. It stores maps in the form of octrees, which saves a lot of space compared to point clouds. The map created by Octomap has different resolutions as shown in Figure 2.4 from left to right, the side length of the cube in the figure is 0.08m, 0.64m, 1.28m^[4].



Figure 2.4 Octomap with different resolutions.

Due to the Octomap, its ground image is composed of many small squares (much like the game Minecraft). When the resolution is high, the square is small; when the resolution is low, the square is large. Each square represents the probability that the cell is occupied. Therefore, the user can query whether a certain area is occupied by a cube to determine whether the area can be "passed" to achieve different levels of navigation. In short, when used for navigation, lower resolutions provide smaller throughput rates, while higher resolutions provide higher throughput rates. Octomap is designed based on octree. Before discussing Octomap, it is very important to elaborate on the expression of the octree.

2.5.1 Introduction to Octree

Octree is a tree data structure in which each internal node can have at most 8 children. As for why it is divided into eight sub-nodes, imagine the three sides of a square block cut one by one, see Figure 2.5^[13].

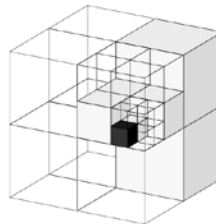


Figure 2.5 Octree cube diagram.

It is a common practice to model 3D space into many squares (or voxels). If each side of a cube is cut into two pieces on average, then this cube will become eight small cubes of the same size. This step can be repeated until the final block size reaches the highest accuracy of the model. In this process, the matter of "dividing a small square into a jack of the same size" is regarded as "expanding a node into an attachment child node". Then, the entire process of dividing from the largest space to the smallest space is an octree. Figure 2.6 shows the octree statistical model^[14].

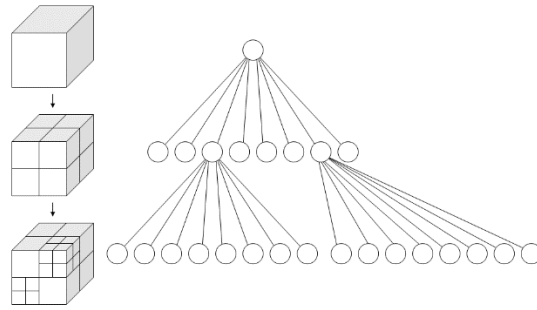


Figure 2.6 Octree statistical model.

In the actual data structure, the root of the tree continues to expand downwards, divided into eight branches at a time, until the leaves. Leaf nodes represent the highest resolution. For example, if the resolution is set to 0.01m, each leaf is a small square with a length of 1cm. Each small square has a number describing whether it is occupied. If the cube size of the leaf node is 1 cm^3 and the octree is limited to 10 floors, the total volume that can be modeled is about 8^{10} cm^3 , about 1073m^3 , which is enough to model a house. Due to the exponential relationship between volume and depth, the modeling volume will grow very quickly when larger depths are used.

In the simplest case, two values of 0-1 can be used to indicate the occupation of a node. 0 means unoccupied and 1 means occupied. However, due to the influence of noise, you may see that a point will be 0 for a while and 1 for a while or 0 for most times and 1 for small times. In addition to these two cases, there is a possibility of an "unknown" state. Usually, a floating-point number between 0 and 1 is used to indicate the probability of being occupied and 0.5 means uncertainty. The larger the number is, the more likely it is to be occupied, and vice versa. Because it is an octree, the eight blades of a node have a certain possibility of being occupied or unoccupied. If it is continuously observed that it is occupied, then let this probability value continue to increase.

Conversely, if it is continuously observed that it is blank, then let the probability value continue to decrease. The obstacle in the map can be dynamically modeled in this way.

Benefits of using a tree structure: If further description is not needed (child nodes), it is sufficient for the octree to retain a thicker block (parent node). This can save a lot of storage space. In an octree, the probability value x is used to indicate whether the leaves are occupied. If you let x increase or decrease continuously, it may run outside the $[0, 1]$ interval, which causes inconvenience in processing, so instead of directly using probability to describe a node being occupied, a logarithmic value of the probability (Log-odds) is used. The occupancy representation of octrees will be introduced in detail in the next section.

2.5.2 Occupancy Representation of Octree

The implementation of octree requires Log-odds to describe the probability of a node being occupied, as follows

$$y = \text{logit}(x) = \log\left(\frac{x}{1-x}\right) \quad (2-3)$$

where y is the logarithm of the probability of value x , and x is the probability between 0 and 1. Its inverse transform is shown as

$$x = \text{logit}^{-1}(y) = \frac{\exp(y)}{\exp(y)+1} \quad (2-4)$$

It can be seen that when y changes from -1 to $+1$, x changes from 0 to 1 accordingly, and when y is 0, x is 0.5. Therefore, it is better to store y to indicate whether the node is occupied.

When "occupancy" is continuously observed, let y increase by one value; otherwise, let y decrease by one value. When querying probability x , use inverse logit transform to convert logarithm of probability y .

In mathematical form, let us consider modern node be n and the observation data be z . Then the logarithm of the probability of a node n is occupied or not from the beginning to time $T-1$ is L , then the logarithm of the probability $L(n|z_{1:T})$ of a leaf node n to be occupied given the sensor measurements $z_{1:T}$ at time T is shown in the equation as^[4],

$$L(n|z_{1:T}) = L(n|z_{1:T-1}) + L(n|z_T) \quad (2-5)$$

It's a bit more complicated if written in a probabilistic form instead of a log-probabilistic form. But probabilistic models are easier to understand as shown in the following equation,

$$P(n|z_{1:T}) = \left[1 + \frac{1 - P(n|z_T)}{P(n|z_T)} \frac{1 - P(n|z_{1:T-1})}{P(n|z_{1:T-1})} \frac{P(n)}{1 - P(n)} \right]^{-1} \quad (2-6)$$

This updated formula depends on the current measurement z_T , a prior probability $P(n)$, and the previous estimate $P(n|z_{1:T-1})$. The term $P(n|z_T)$ denotes the probability of voxel n to be occupied given the measurement z_T . This value is specific to the sensor that generated z_T ^[4].

The occupancy probability of the parent node in the octree can be calculated based on the value of the child node. The simpler is to take the average or maximum. The octree is rendered according to the occupation probability, the uncertain block is rendered transparent, and the occupied occupation is rendered opaque.

In the chapter, an approach for visualization of map points was proposed. The next chapters will discuss the simulation and implementation of the approach utilizing ROS.

3. SIMULATION

Before the actual implementation of the approach, it is usually necessary to prove the feasibility of the approach through simulation. Simulation refers to the use of models to reproduce the necessary processes that occur in actual systems and to study existing or designed systems by experimenting with system models. The scope of the simulation includes electrical, mechanical, chemical, hydraulic, thermal, and other scientific fields^[1]. Simulation can also be used in social, economic, ecological, and management areas. Simulation is a particularly effective means of research. Simulation is suitable when the system under study is expensive, and the risk of experimentation is high. Sometimes it takes a long time to understand the consequences of system parameter changes. At this time, the simulation can significantly speed up the experimental process.

In this project, computer technology is used to simulate physical experiments^[2], which can avoid various risks that may occur in robot hardware. These include but are not limited to network communications, wireless control of robots, environmental impact, and machine kinematics control. The simulation process includes two main steps: establishing a simulation model and performing a simulation experiment for visualization. In this project, Gazebo is used to build the environment. In this virtual environment, the main limitation comes from the CPU computational power, RAM speed and video card power of the computer. In order to more effectively implement the construction of the environment, this project uses Gazebo under the ROS platform as the simulation tool. Firstly, the simplified interior corridor environment was rebuilt. Import the Clearpath Jackal model in Gazebo and install the camera VSLAM package. Control the robot in Rviz to cruise in the simulated corridor to get a real-time map to be visualized by Octomap. After proving that Octomap can visualize simple empty indoor environments, it is necessary to further examine the ability of Octomap to visualize a more complicated environment with furniture and people.

3.1 Gazebo

In the simulation, Gazebo will be used to build the environment. Simulation refers to the creation of a model to characterize its principal features or behaviors/functions based on the

purpose of experimentation or training. The virtual model is systematized and formulated to simulate critical features.

Gazebo is a powerful 3D physics simulation platform with a powerful physics engine, high-quality graphics rendering, and graphical interface^[32]. The most important thing is that Gazebo is open-source. The robot model in the Gazebo has the same 3D appearance as the model used in Rviz^[33], but the physical properties of the robot and the surrounding environment (such as mass, friction coefficient, and spring constant) are added to the Gazebo model. The sensor information of the robot can also be added to the simulation environment through the form of a plug-in to display visually.

3.2 Build the Environment

Before start using Gazebo to create an environment based on real scenes, it is necessary to install ROS installed with the corresponding version of Gazebo^[34]. Specifications of the computer system used in this simulation are shown in Table 3.1.

Table 3.1 Computer systems specifications.

Operating System	Ubuntu 16.04 LTS
Memory	6.8 Gbyte
Processor	AMD Ryzen 5 2500u With Radeon Vega Mobile Gfx × 8
Graphics Card	AMD Raven (Drm 3.23.0 / 4.15.0-64-Generic, Llvm 6.0.0)
ROS Version	Kinetic

If the full desktop ROS is already installed, Gazebo does not need to be installed separately. Please refer to Appendix A.3.1 for how to reinstall Gazebo and Map Editor. Gazebo is to create a similar 3D scene, modeled after the actual environment on the second-floor corridor of the Potter Building on campus. In order to reduce computational pressure, the model is made as simple as possible. Figure 3.1 shows the details of the simulated environment.

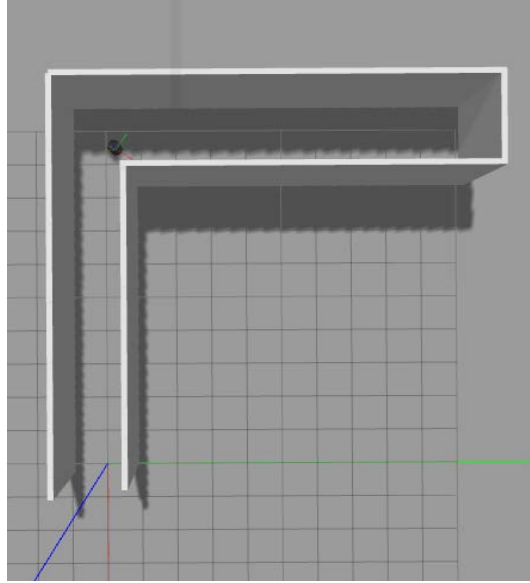


Figure 3.1 3D scene of the second-floor corridor.

When an environment is built in the simulation, it usually needs to be saved as an attachment. In this way, it is convenient to call up the saved environment later. To test the applicability of the approach to a more complicated environment, the simulation scene was replaced in Gazebo to rebuild a square room environment with three bookshelves, a coffee table, and a standing person as shown in Figure 3.2.

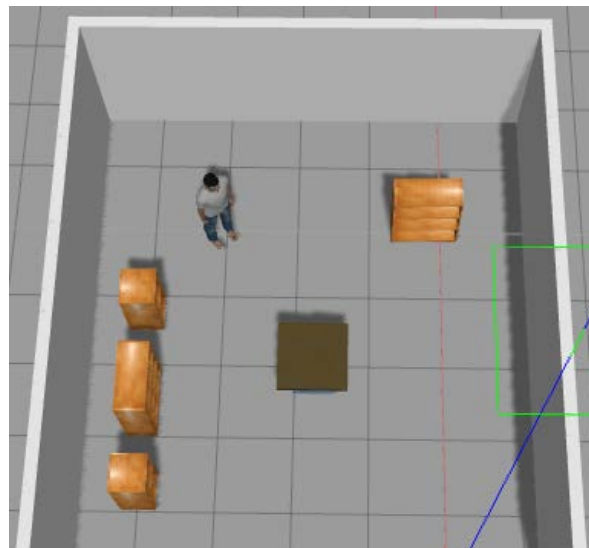


Figure 3.2 3D scene of the complex indoor environments.

After building the environments, the next stage is to simulate the robot in the environments. To how to simulate the Jakal robot, refer to Appendix A.3.2. The loaded robot model is shown in Figure 3.3.

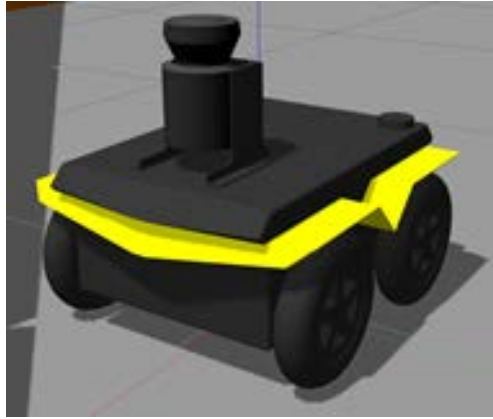


Figure 3.3 Model of the jackal robot.

This robot has a complex physical structure in order to achieve a state of motion like the actual robot. When the robot model is selected, the model will become transparent and the coordinates of the robot will be displayed as shown in Figure 3.4. In the Jackal model, the four wheels and the fuselage have separate models and independent coordinates, which allows the wheels to rotate in real-time, showing the true attitude of the robot. At the same time, the various components of the robot have their physical properties, including friction coefficient, mass inertia, and collision volume.

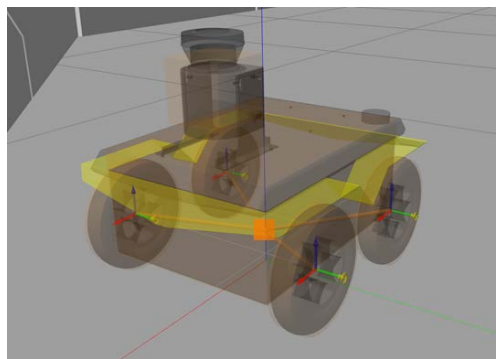


Figure 3.4 Details of the jackal robot.

After completing the environment construction and the robot model in Gazebo, it should be noted that the Gazebo can only simulate the environment and the robot, and is not a visualization platform. In order to ensure the consistency of simulation with real experiments, the robot will be equipped with an RGB-D camera as explained next.

3.3 Visualization of Map Points from SLAM Using RGB-D Camera

In order to simulate the actual situation of data collection using RGB-D cameras in the simulated environment, it is necessary to install the RGB-D camera on the robot in the Gazebo. Refer to Appendix A.5 for installation details.

Although Rviz is similar in appearance to Gazebo, the use of Rviz is quite different. Unlike Gazebo, which shows the reality of the simulation world, Rviz demonstrates the robot's perception of its world, whether real or simulated. Rviz plays a key role in the visualization in this research. Appendix A.4 provides more details on RVIZ.

To start the simulation of the robot moving inside the corridor, set the robot at the beginning of the corridor as shown in Figure 3.5.

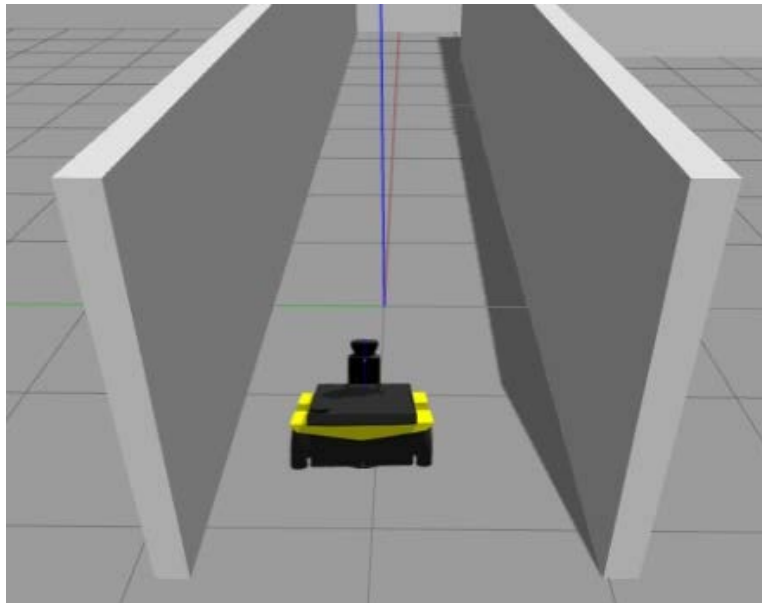


Figure 3.5 The corridor with the robot.

Using launch invocation to start Rviz with standard Jackal shown in Figure 3.6:

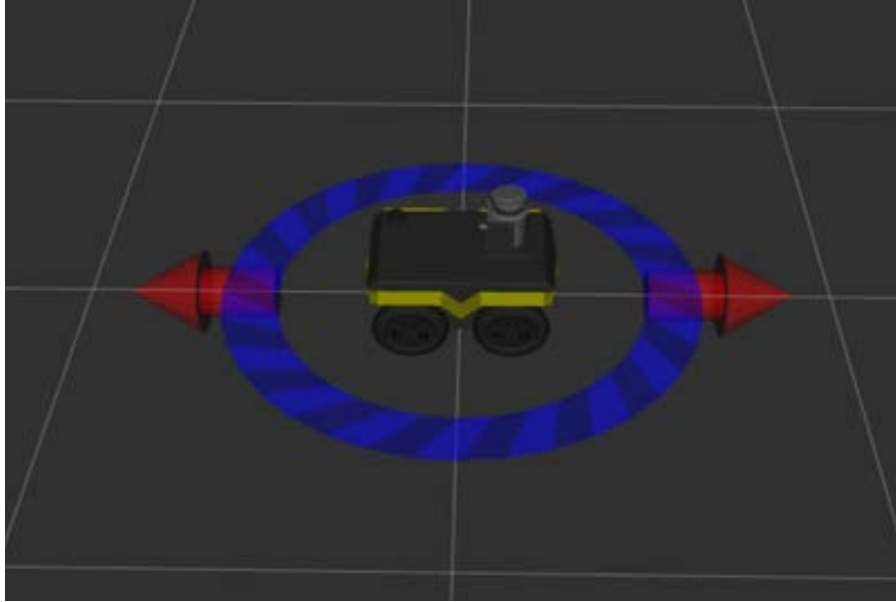


Figure 3.6 Rviz with the robot.

The Rviz display only shows what the robot knows about its world, which presently, is nothing. Because the robot doesn't yet know about the barriers which exist in its Gazebo world, the barriers not shown in Figure 3.6.

At this stage, the Rviz software required for the simulation has been correctly set up, and the map data can be read from the camera. The problem that needs to be solved at this time is the manipulation of the robot. The robot can be moved through the game controller, or by inputting commands to adjust the robot speed and steering angle by inputting commands. Interactive Markers offer another solution. The robot can be operated very intuitively. Different operation methods have their advantages. This project used Interactive Markers to move the robot.

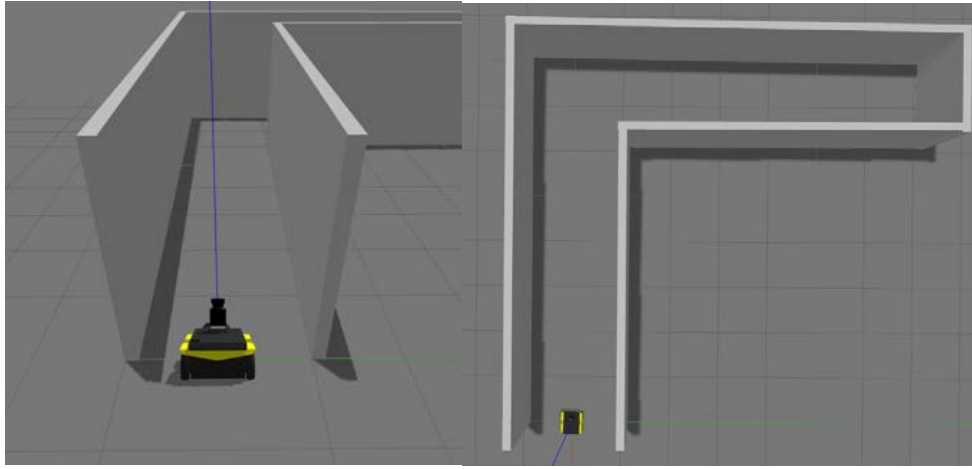
3.4 Interactive Markers Controlling the Mobile Robot

As shown in Figure 3.6, the blue circle and red arrows are Jackal's interactive markers. These are the simplest way to command the robot to move around. Drag the red arrows in Rviz to move in the linear x and the blue circle to move in the angular z . Rviz shows the Jackal moving relative to its odometric frame, but it is also moving relative to the simulated world supplied by Gazebo. If clicking over to the Gazebo window, the Jackal moving within its simulated world at the same time.

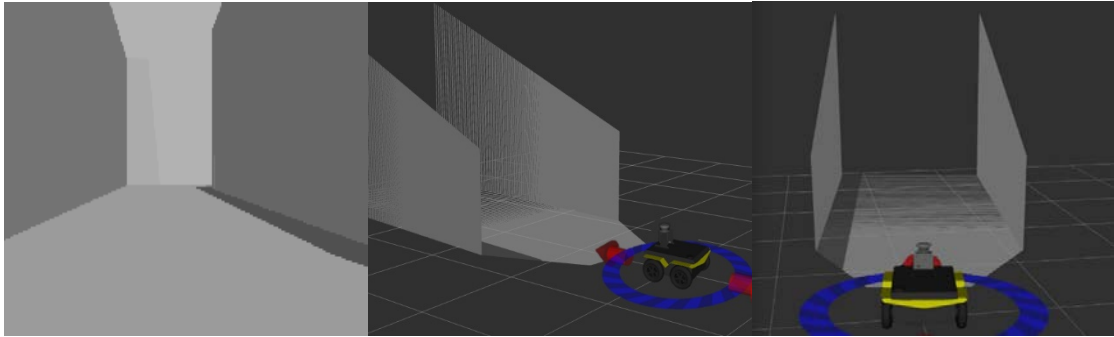
At that moment, we can control the robot with Interactive Markers^[20] in Rviz, the robot model will also show the movement in Gazebo. With the robot traveling in the simulation world, the 3D Octomap can be built at the same time based on the RGB-D camera in Rviz.

3.5 Simulation Results

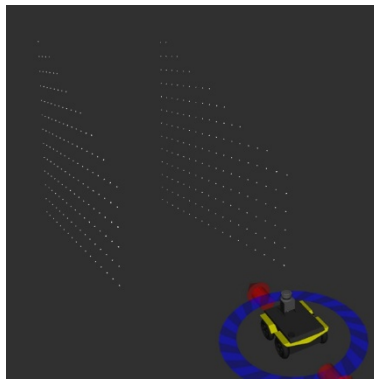
At the beginning of the simulation, the robot position was preset at the beginning of the corridor. At this time, the data obtained by the RGB-D camera is published to the receiving node. Octomap can be directly displayed in Rviz on the left. The resolution (small cube side length) of the Octomap obtained at this stage is set to 10 cm. The initial state is shown in Figure 3.7 on the next page.



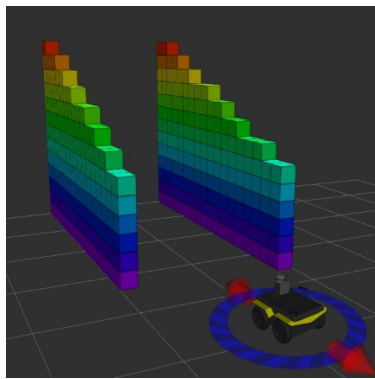
(a)



(b)



(c)



(d)

Figure 3.7 The initial state of the simulation experiment. Figure (a) shows the position of the robot in Gazebo at the beginning of the experiment, (b) shows the scene seen by the RGB-D camera and the field of view of the RGB-D camera, and (c) shows the point cloud obtained by the ORB-SLAM2 method, (d) shows an Octomap constructed using point clouds.

Figure 3.7 (b) (c) and (d) are all displayed in Rviz. The robot's movement in the simulated environment can be controlled by the Interactive Marker in Rviz. Pull the arrow in front of the robot in Rviz to move the robot forward. From Figure 3.7(b), it can be seen that due to the camera perspective, the slope of the wall will appear during the initial stage of map construction. Some walls outside the viewing angle will not be displayed. In Figure 3.7(d), in order to express the height of the wall in the visualization, the gradient from blue to red is used to express from low to height. The map will be constructed during the movement and the situation during the detection as shown in Figure 3.8.

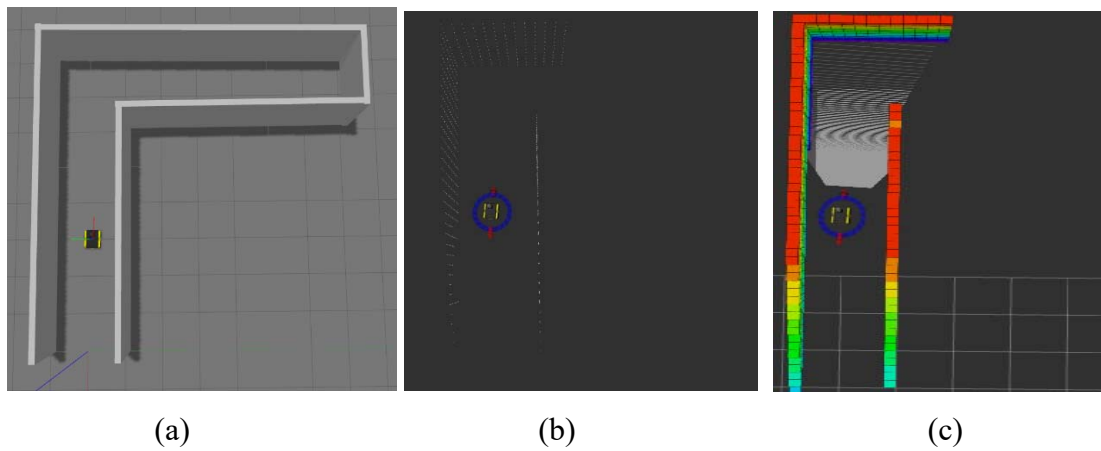
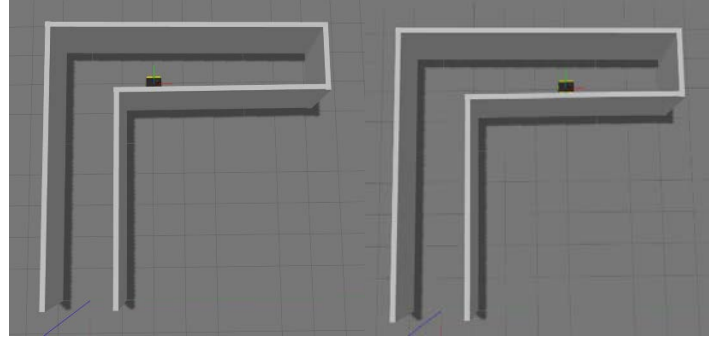
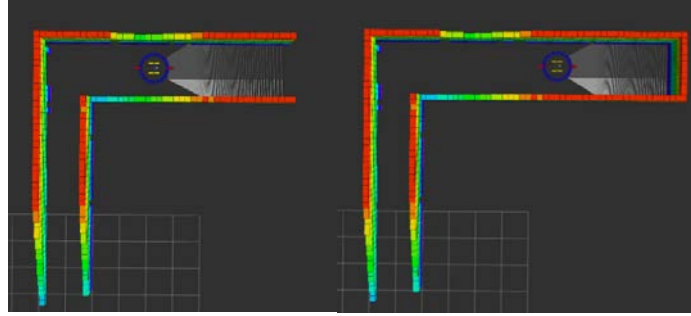


Figure 3.8 Intermediate state. Figure (a) shows the position of the robot in Gazebo in the middle of the straight corridor, (b) shows the point cloud, (c) shows the camera field of view and the Octomap constructed using point clouds.

When the robot reaches the end of the corridor, turn the blue disk around for the robot in Rviz to make a right turn. At this time, the camera can detect the terrain behind the corner. After the turn, the robot continued to go straight until the entire corridor was completely explored, as shown in Figure 3.9.



(a)



(b)

Figure 3.9 Map building behind the corner. Figure (a) shows the movement of the robot in Gazebo, (b) shows the camera view field and the Octomap.

Since the end of the corridor has been displayed, the robot can stop moving to complete simulation. The visualization obtained from the simulation experimental results is compared with Gazebo. The comparison is shown in Figure 3.10.

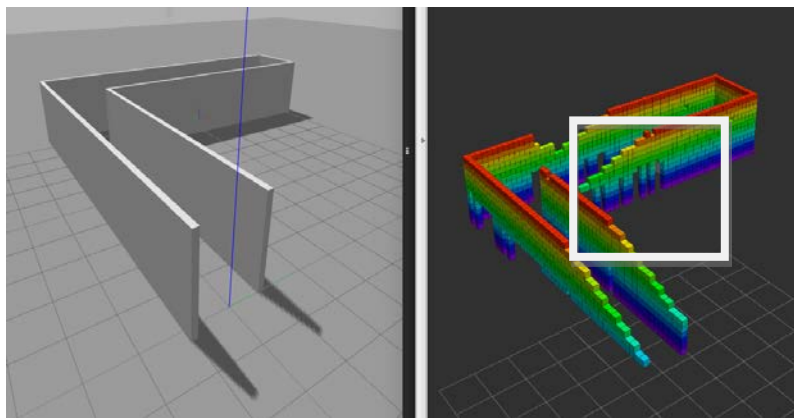


Figure 3.10 Octomap obtained from the simulation experiment.

It can be seen that, in the simulation, to facilitate the observation of the robot's movement and reduce the computation, the detected corridor did not design the roof and did not add materials to the ground, so there was no roof or floor in the visualizations. As shown in the white square in Figure 3.10, it can be seen that when turning, a detection error similar to the beginning of the experiment occurs, and a part of the wall surface is stepped. This can be explained from the point cloud and camera field of view at the end of the turn. Visualization errors due to turning are shown in Figure 3.11.

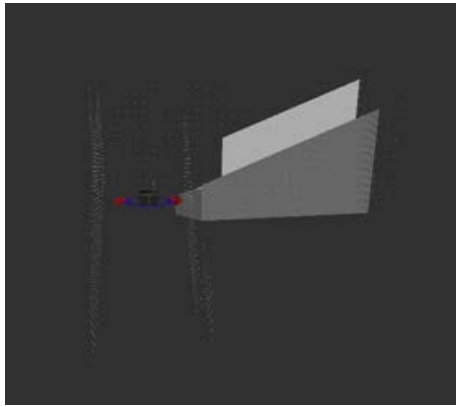


Figure 3.11 Visualization errors that occur while turning.

It can be seen that the field of view of the simulated RGB-D camera is limited. In this experiment, the default field of view of the RGB-D camera kit is used. During the turning process of the robot, the details of the wall cannot be fully captured, similar to the initial state in the simulation. Also, there are some errors in the lower wall height. Trying to repeat the simulation in the same area can reduce these errors. But in general, simulation has successfully achieved the real-time construction of the environment using Octomap. The simulation results show the applicability of using Octomap for visualizing the map points.

The next simulation is to test the visualization capability of the Octomap of complex environments. The environment constructed in Gazebo is transposed as shown in Figure 3.2. It contains a square room environment with three bookshelves, a coffee table, and a standing person. In order to improve the visualization, the resolution of the Octomap (small cube side length) obtained at this stage is set to 1 cm. Due to the increase in the complexity of the room, the increase in resolution, and the real-time visualization as much as possible, it is necessary to reduce the

computation to prevent Rviz to forcibly terminated. The way to reduce the computation is not to display the model of the Jackal robot in Rviz. The other settings remain unchanged. Figure 3.12, shows the starting state of the simulation process in Rviz.

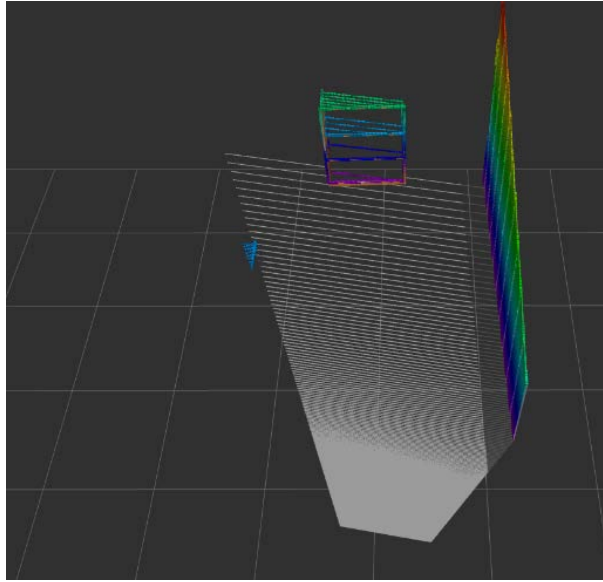


Figure 3.12 Starting state of the simulation process for a more complicated environment in Rviz.

Figure 3.13 shows the state at the midpoint of the simulation process in Rviz. The red arrow indicates the path of the robot's movement.

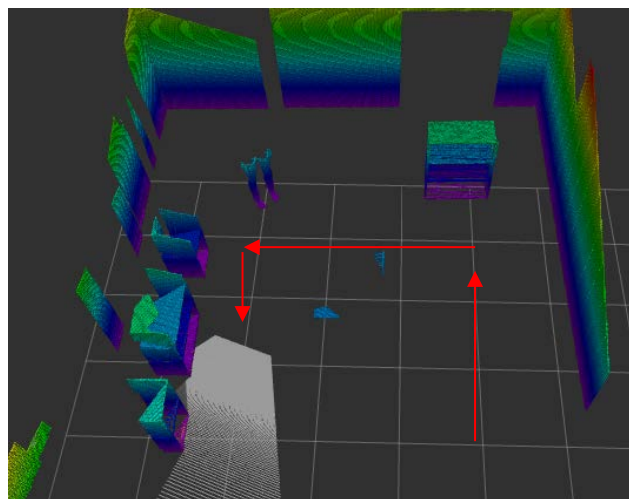


Figure 3.13 The midpoint visualization for a more complicated environment in Rviz.

Figure 3.14 below shows the visualization of the end of the simulation in Rviz. The red arrow indicates the path of the robot's movement.

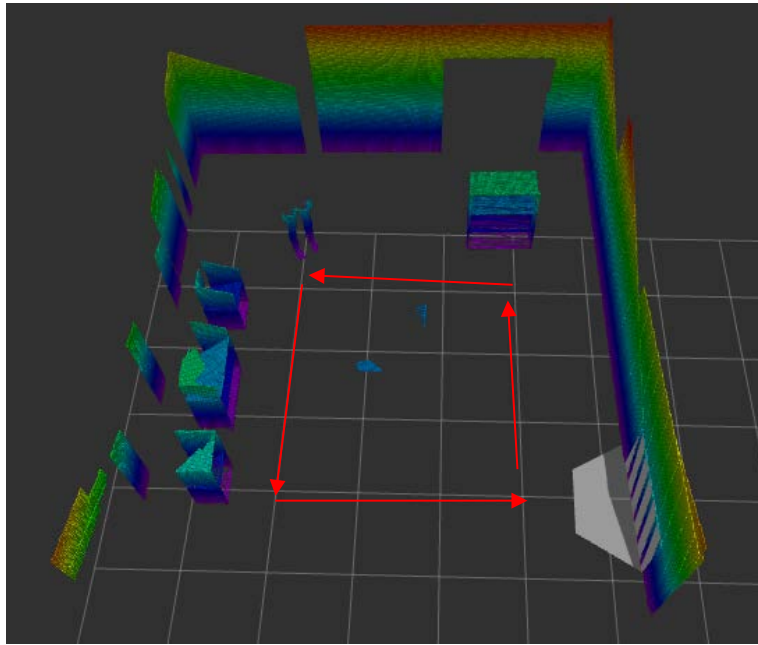


Figure 3.14 The final visualization for the complex room in Rviz.

Due to the height of the camera angle of view, it can be seen that the visualization of people is limited to the lower body. In order to verify the visualization of people, operate the robot to the position shown in Figure 3.15 below to make a targeted observation of people again in a different distance.

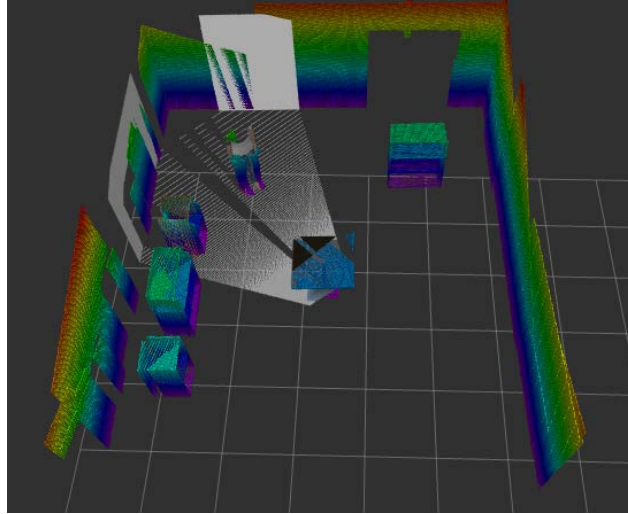
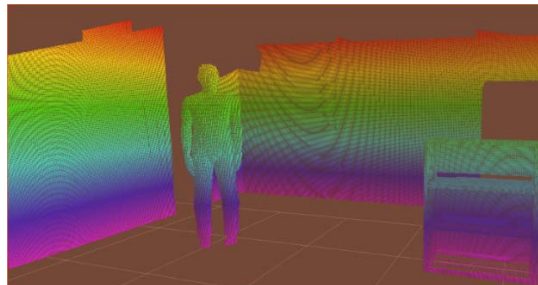


Figure 3.15 Targeted observation of people.

The comparison between the visualization results obtained from targeted observations of people and the model in Gazebo is shown in Figure 3.16.



(a)



(b)

Figure 3.16 The comparison between the visualization results obtained from targeted observations of people(a) and the model in Gazebo (b).

Combining two simulation results for different environments, the following conclusions are drawn. RGB-D camera SLAM is limited by the field of view of the camera, sometimes multiple observations of the same location at different distances are required. Nevertheless, with good map information, Octomap can well realize the visualization of different indoor environments, including objects with complex shapes such as people. The next chapter will use Octomap to visualize the experimental offline data.

4. VISUALIZATION EXPERIMENTS USING CAMERA INFORMATION

In this chapter, a visual experiment was performed using a camera to show the applicability of the method described in Chapter 2. After obtaining sufficient theoretical experience from the simulation experiment, the approach is applied to the real-world environment. Since the real environment is much more complicated than the simulated environment, the camera information is inevitably noisy, as a result, the algorithm discussed in Section 2.2 is used for filtering. Due to the camera's different accuracy, the information is downsampled and increased sampled for surface smoothness using the approach mentioned in Sections 2.3, and 2.4 before visualization.

According to the experiments conducted, the intermittent connection between the robot and the host computer often occurred. Point cloud preliminary detection results can sometimes show significant errors. These challenges are often caused by component failures or physical-level equipment limitations. In order to be able to deal with the various situations that may occur in the experiment more specifically, it is necessary to visualize offline data first.

In different environments, different methods are often used to process and optimize the data to get the best results. Even in indoor environments, the impact of different scenarios on data collection is often huge. As the team providing data in this project uses visual cameras for data collection, the reasons for the impact may come from indoor light, floor and wall materials, and environmental uniformity. Before discussing software-level issues in detail, the next section, the hardware equipment and experimental procedures used for the indoor experiment are presented.

4.1 Experimental Setup

The experimental team conducted several experiments to evaluate the performance of the ORB-SLAM algorithm using the platform discussed next. Most of the indoor environment is constituted by rooms and corridors. Therefore, in this project, two experiments were selected to test the applicability of the visualization approach.

4.1.1 Hardware Platform

The experiments use a mobile robot Jackal was used with different kinds of cameras shown in Figure 4.1. Jackal is a 4-wheel drive mobile robot with the ability to run through the rough surface. Each wheel cannot change directions. So, it turns in the differential mode drive is used which means it could turn in situ.



Figure 4.1 The Jackal robot shown with both a bumblebee camera and an RGB-D camera.

The robot motherboard is a 2.4GHz Celeron J1800 duo core processor and a 2GB memory card. Ethernet and WIFI are both available for communication. The robot can receive commands from both a PS4 controller by Bluetooth or a server computer by WIFI. Since the robot does not have a monitor, in order to check the robot system's working statement, an additional display screen with a VGA port is necessary. A keyboard also needed to connect on the robot motherboard for using the Linux system on the robot. When the robot successfully connects to the router, the robot can wirelessly connect to another computer via a secure shell (SSH).

No matter if data filtering or modeling in the visualization process is performed, there are quite high requirements on the hardware performance of the laptop, so it is very important to properly construct the computer system environment. The computer specifications are shown in Table 3.1.

It should be noted that although this computer system is capable of data processing, it only meets the most basic requirements of the visualization process. When the map is large or the point cloud density is high, the computer will take a long-time to complete calculations, making it almost impossible to build the map. Rviz for visualization is sometimes forced to stop the process when the point cloud density is high. This makes it necessary to trim the point cloud that will be used for visualization and select the part of the environment for visualization. The experimental software environment is described next.

4.1.2 Software Platform

At this stage, the software environment required for the experiment needs to be set up. Ubuntu 14.4 modified by the Clearpath Robotics, Inc is originally installed as the operating system on the Clearpath Jackal robot. What's more, Robot Operating System (ROS) is open-source software that works with the Ubuntu system and provides us access to manipulate the robot. Because the simulation was performed before the experiment, the operating system Ubuntu + ROS and its related toolkits have been installed. If it is found in the experiment that the component is missing and the process cannot be completed, you can re-enter the relevant installation instructions mentioned in Appendix A. These instructions will check for inoperable processes, patch missing parts and upgrade to the latest software.

In order to process offline point cloud files, the PCL library and related components are installed. Please refer to Appendix A.2 for the support environment that requires the installation of PCL related components. Next, the visualization results of offline map data points using ORB-SLAM are presented.

4.2 Visualization of the Right-angle Corridor Environment

The data source for this visualization experiment was an experiment conducted by the team on the second floor of the Potter building corridor as shown in Figure 4.2. A right-angle corridor similar to that used in the simulation was selected for the experimental site. Labs and offices lay on both sides and the notice boards are hang on the wall.



Figure 4.2 The right-angle corridor on the second floor of the Potter Building.

ORB-SLAM using an RGB-D camera to perform robot localization and map the environment.

4.2.1 Map Points Collection Using Camera

In order to implement the SLAM process, the experimental team used the Features from Accelerated Segment Test (FAST)^[56] algorithm, the Binary Robust Independent Elementary Features (BRIEF)^[57] algorithm, and the Oriented FAST and rotated BRIEF (ORB)^[58] algorithm to estimate environment map and camera positions. After that, the experimental team performed local mapping. Requiring, map point culling, new map points insertion, local bundle adjustment, and redundant keyframe culling. Finally, the loop closing is needed. The loop closing thread is a very important part of the SLAM system. Due to the cumulative error (drifting) of the visual odometry (VO)^[59] process, the main task of the loop closing thread is to detect the closed-loop using Bag of Words (BoW)^{[60][61]}. After detecting the closed-loop, similarity transformation Sim3^[62] is calculated to close the loop. At last, the global bundle adjustment^[62] is applied to optimize the keyframe poses and map points cumulative error to an acceptable range.

The camera used in this experiment is Kinect V1^[50], powered by Microsoft which can capture depth as well as images. The SLAM results are shown in Figure 4.3.

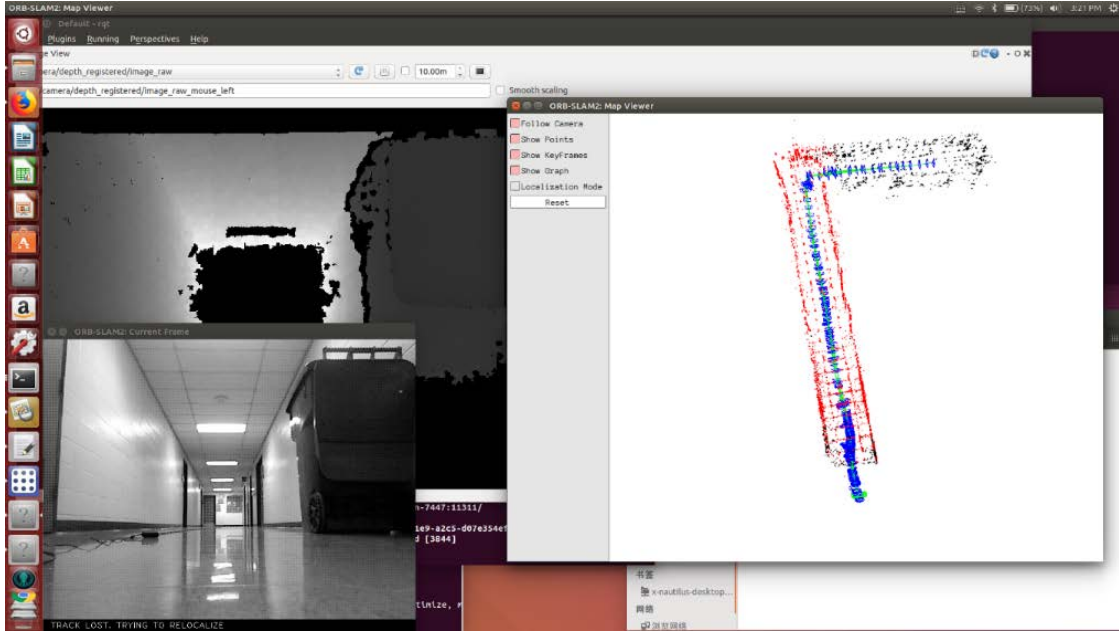


Figure 4.3 The experimental results from RGB-D cameras in the right-angle corridor.

The red points are the estimated map points that can be detected at the camera's current location while black points are optimized map points for the entire robot trajectory. Blue squares represent the locations and pose of the keyframes and record the camera trajectory. In this research, only optimized map points, that is, black point clouds, are used for visualization. Open the point cloud file separately for preview as shown in Figure 4.4.

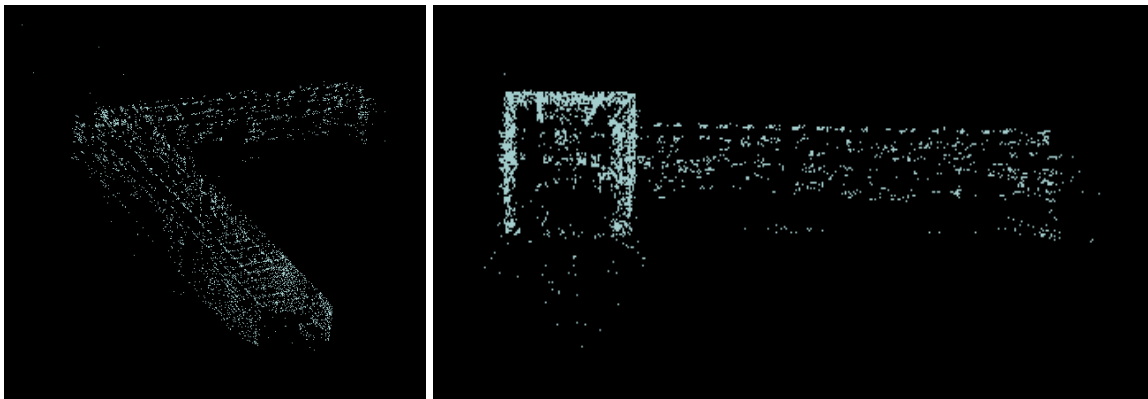


Figure 4.4 The left image is an overview of the map point cloud, and the right image is the front view of the point cloud.

If the point cloud shown in Figure 4.4 is used to generate an Octomap. The visualization result is shown in Figure 4.5.

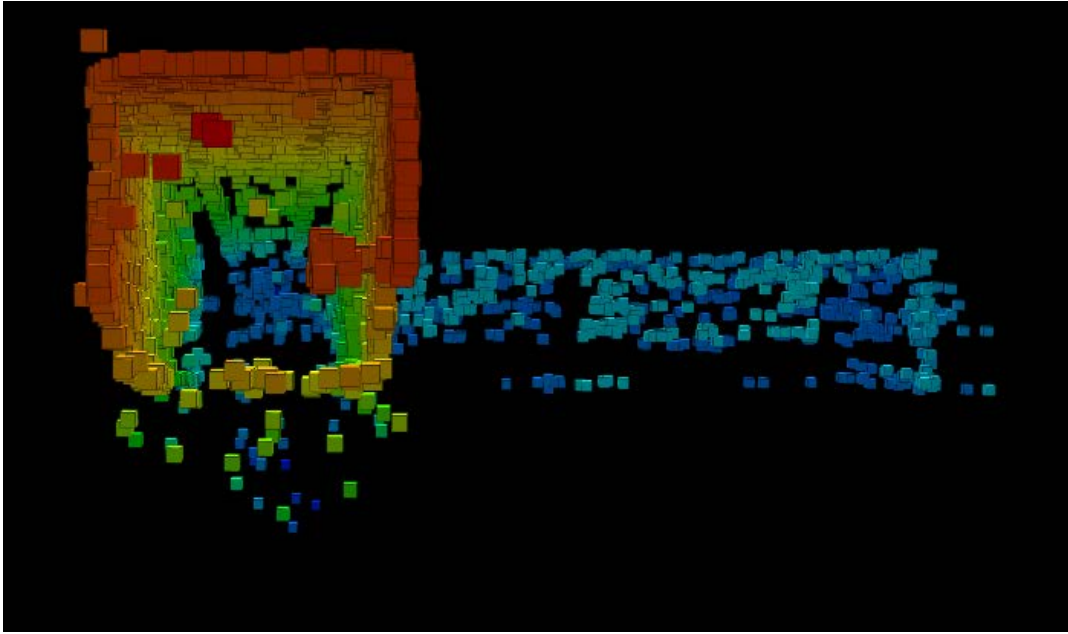


Figure 4.5 The Octomap visualized based on the map point.

It can be seen that although the SLAM process applied can fully record the shape of the corridor. But the visualization is not good. Our approach is to filter map points as discussed in Chapter 2 to improve visualization. In the following stages, several filters will be used to process and visualize the results obtained using the RGB-D camera. After visualizing a single right-angle corridor, the same method is used to deal with the point cloud of pedestrians in the square corridor. The purpose of this is to verify the robustness of the visualization method for a more complicated environment.

4.2.2 Map Points Data Processing

For preliminary processing, the pass-through filter and statistical removal filter are used. Fortunately, Point Cloud Library(PCL) provides these filter capabilities. For specific usage of this function, please refer to Appendix B.1.1. After preliminary processing with the pass-through and statistical removal filters, the new point cloud is shown in Figure 4.6.

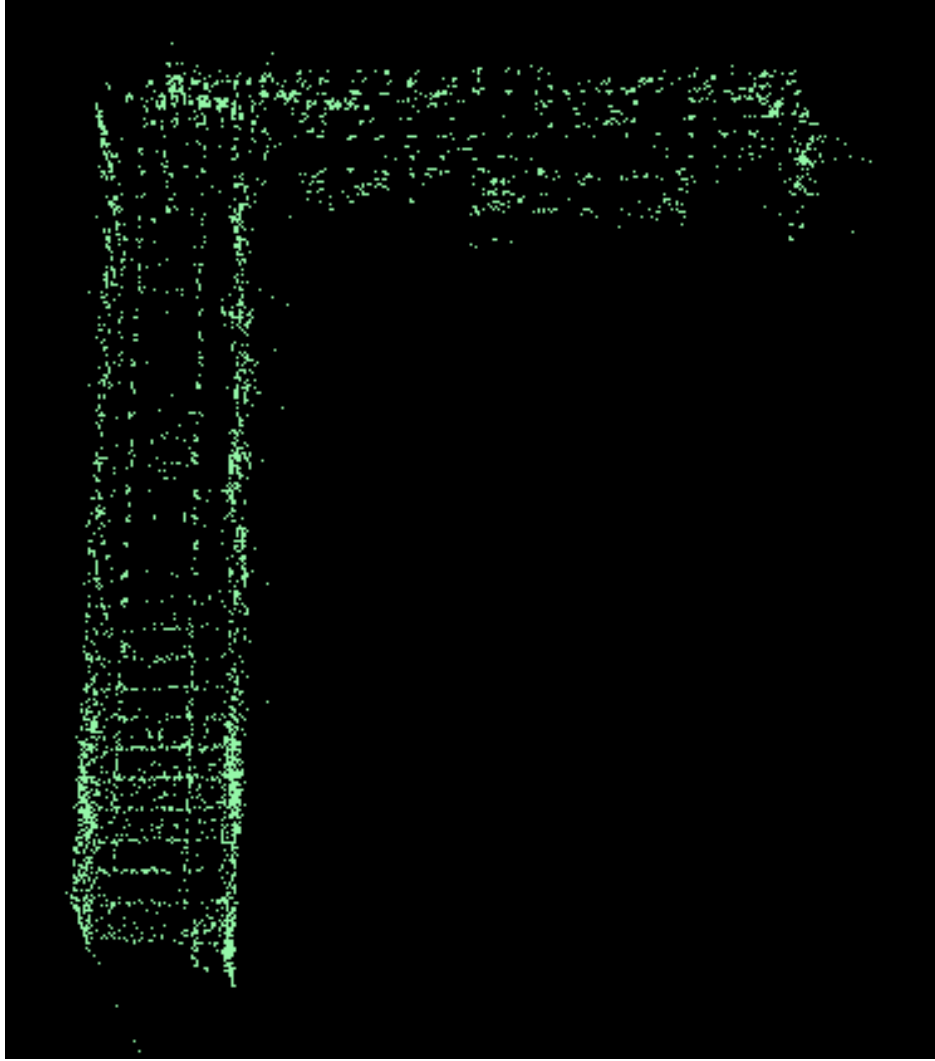


Figure 4.6 Map point cloud after preliminary processing.

Because of the large density of the points in the point cloud map, many of the points have the same position. The approach described in Section 2.6 is to use the voxel filter to reduce the density of the point cloud without losing key features of the map. Using the dichotomy method to set the leaf size multiple times in the interval from 0 to 1 to compare the voxelization effect, it is required that the density of the point cloud be as small as possible without losing too many map features. Finally, the leaf size is determined to be 0.03m. The leaf size requirements at this stage are not critical. Due to the powerful utility of the octree (discussed in detail in Section 2.8), minor errors will be repaired during the visualization stage. For detail implementation, please refer to Appendix B.1.2. The point cloud after downsampling is shown in Figure 4.7.



Figure 4.7 Point cloud after downsampling.

After the downsampling is completed, in order to repair the continuity of the point cloud and the occurrence of holes, increase sampling is required to smooth the point cloud. To achieve this, refer to Appendix B.1.3 for details. The result is shown in Figure 4.8.

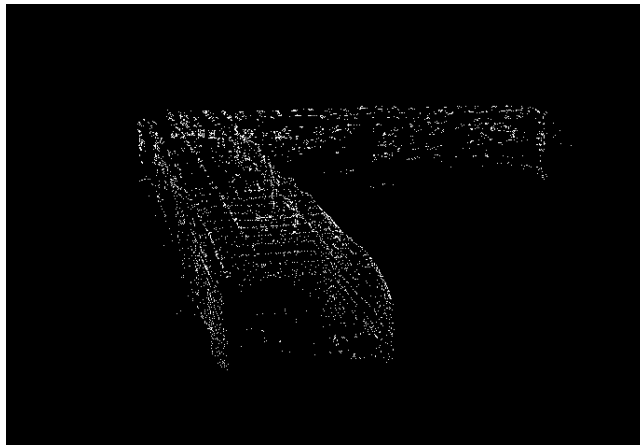
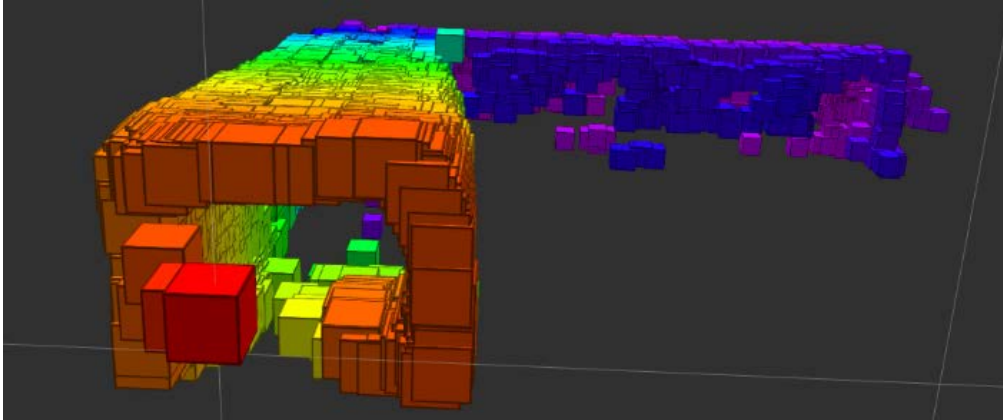


Figure 4.8 Point cloud after increase sampling processed.

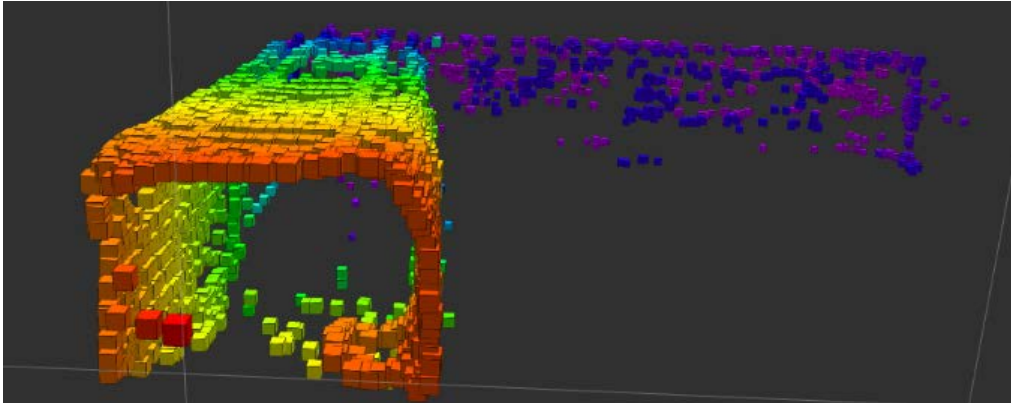
Before entering the visualization phase, the point cloud files are saved. The next challenge is to publish point cloud data to a topic through ROS and then start the Octomap to read the data in Rviz for visualization.

4.2.3 Publishing and Visualizing Point Cloud Data

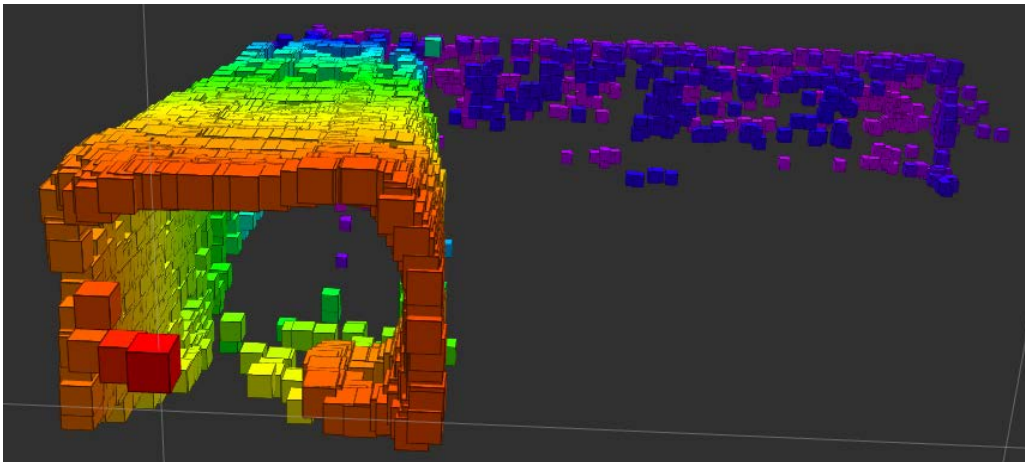
The map point cloud has been processed and filtered as described in previous Section. The specific steps for publishing and visualizing this point cloud are described in Appendix B.2. After these steps is completed the point cloud has been visualized as Octomap. Based on the excellent characteristics of Octomap, the resolution of the visualization can be arbitrarily customized, and the side length of the basic cube that forms the map can be set. Figure 4.7 (a) shows the result of a cube side length of 0.1m, Figure 4.7 (b) shows the result of a cube side length of 0.03m, and Figure 4.7 (c) shows the result of a cube side length of 0.05m.



(a)



(b)



(c)

Figure 4.9 Octomap. (a) shows the result of a cube side length of 0.1m, (b) shows the result of a cube side length of 0.03m, (c) shows the result of a cube side length of 0.05m.

In Figure 4.9, the color changes from red to blue to represent the depth change of the map. Compared with Figure 4.5, it can be seen that the results obtained after improved visualization by removing out of range points. At the same time, the walls appear smoother and resemble more closely the actual wall.

In order to better show the robustness of this approach, another visualization was performed for a more complicated environment. After completing the visualization of the simple right-angle corridor without pedestrians, the next step is to visualize the data obtained from the square corridor environment with pedestrians in the same way. Map points are processed using the same filters before visualization.

4.3 Visualization of the Square Corridor Environment

To test the robustness of this approach, a point cloud map obtained in another experiment is visualized. The environment is a typical square corridor. In this experiment, the RGB-D camera is used for information collection. The experimental team used the same approach as discussed before in Section 4.2.1. The SLAM results are shown in Figure 4.10.

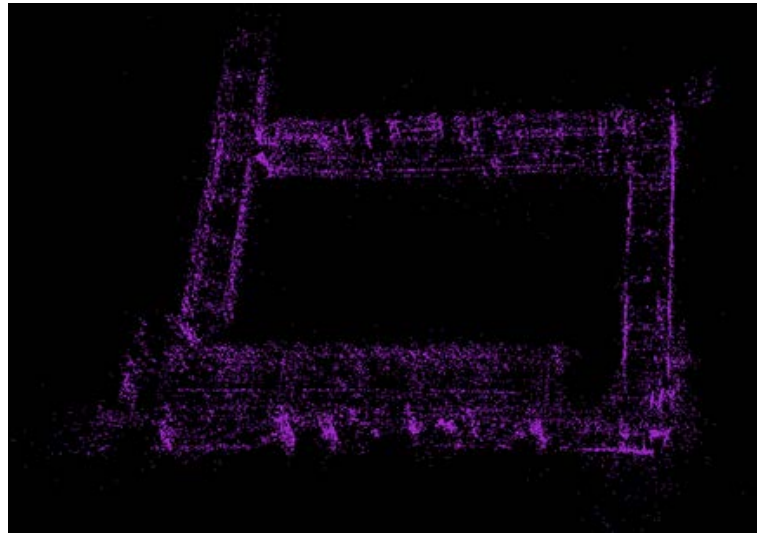
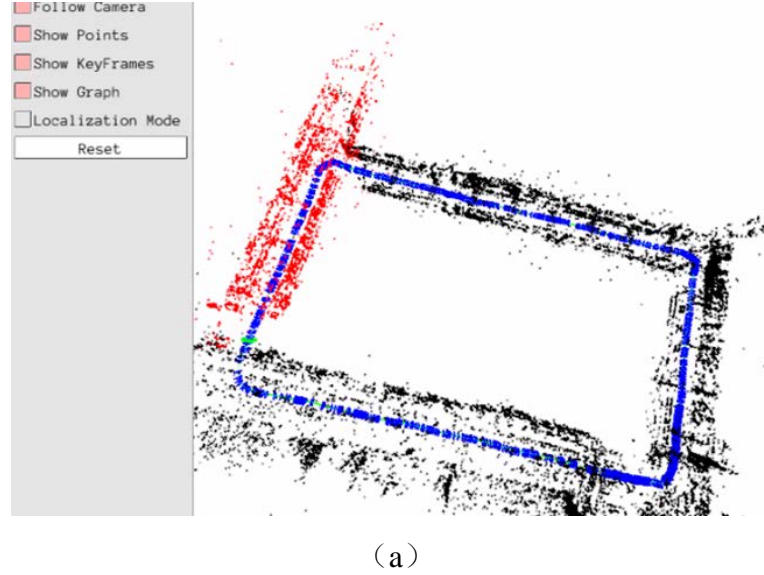


Figure 4.10 (a)The SLAM experimental results from RGB-D cameras in the square corridor
(b)The map points obtained for the square corridor environment.

In Figure 4.8(a), the red points are the estimated map points that are detected at the camera's current location while black points are optimized map points for the entire robot trajectory. Blue squares represent the locations and pose of the keyframes and record the camera trajectory. It can be seen that the data obtained from the square corridor environment is messy, and there are a lot of noise points distributed in the result. The optimization of the point cloud is performed through data processing. Including pass-through filter, statistical removal filter, downsampling using voxelization, smoothing by increasing sampling. The shape of the point cloud after processing is shown in Figure 4.11.

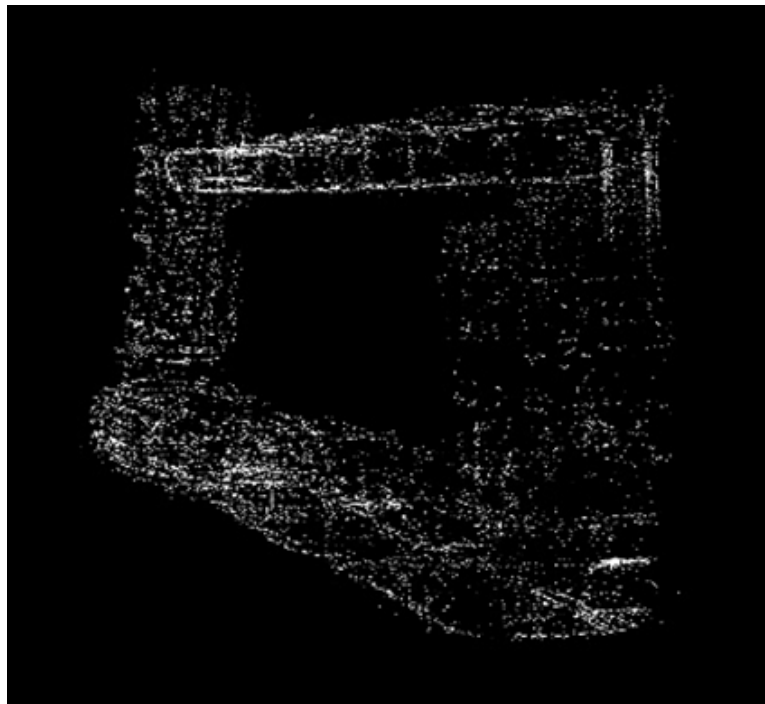


Figure 4.11 The point cloud from the square corridor environment after processing.

After the point cloud is processed, it is ready to be visualized using Octomap. At this stage, the side length of the basic cubes that make up the map is set to 0.1m. The visualization of the square corridor obtained in Rviz is shown in Figure 4.12.

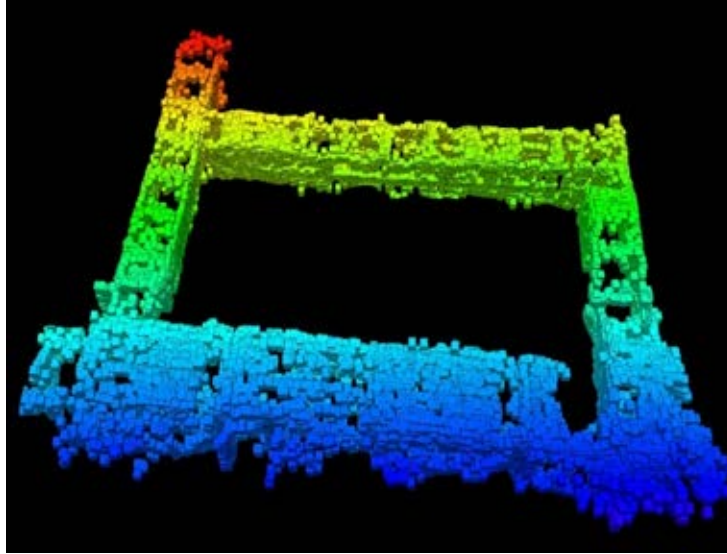


Figure 4.12 Visualization of the square corridor environment using Octomap.

As shown in Figure 4.12 the visualization result is not good, it can still roughly reflect the approximate outer contour of the measured corridor. However, since this point cloud collection experiment was performed during the daily working hours, many dynamic pedestrians were captured by the camera. The point cloud obtained from pedestrians is statistically similar to noise in free space, so this will cause these points to be removed by the data processing filters. After voxelization and surface smoothing, the position of pedestrians will no longer be significant. This shows that the map points are very important for the visualization. The quality of the initial data can greatly affect the visualization results. In the following section, high-density point clouds are used as input to evaluate the performance of the approach in the case of very dense accurate point cloud data.

4.4 Visualization of the Stairwell Environment

To verify the connection between the visualization results and the accuracy density of the point cloud, the same visualization methods can be used to analyze some high-density point cloud data. This can be seen as a test of the reliability and universality of the visualizations approach discussed in the previous section.

In this experiment, the point cloud data used for visualization comes from the main site of Technische Universität München^[51]. LiDAR and Kinect camera are used together in this project.

The dataset is licensed under the Creative Commons Attribution 3.0 Unported License, which allows the use of dataset both commercially and non-commercially. After data processing and octree-based visualization, the results are shown in Figure 4.13.

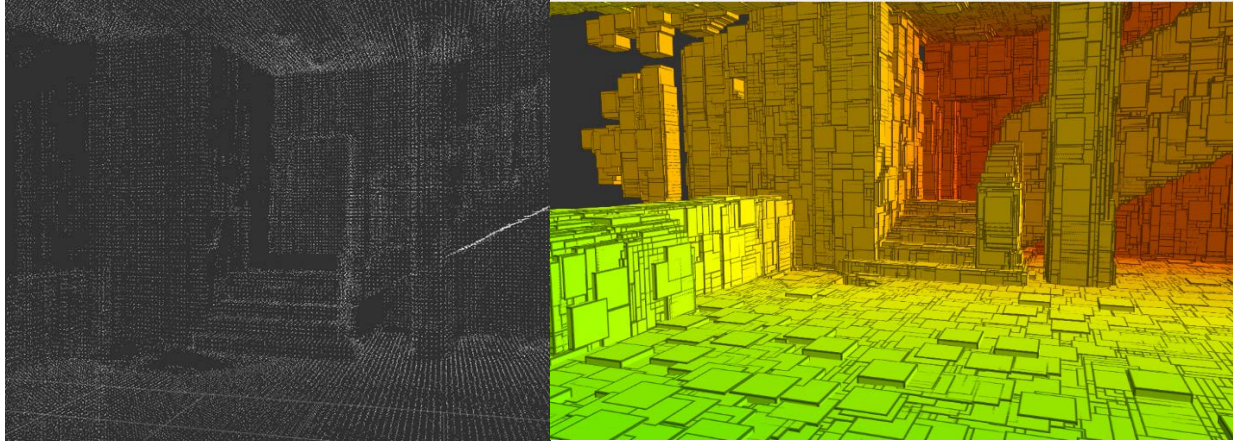


Figure 4.13 The left image is the point cloud view after downsampling. The right image is the result of the visualization using Octomap.

According to the environment shown in Figure 4.13, although the point cloud data has almost no noise, the accuracy is very high, but because the point cloud density exceeds the processing capacity of the laptop, downsampling was performed using voxelization. The geometrical features of point clouds are basically retained, while the number of computations is greatly reduced. The Octomap was constructed using the cube with a side length of 0.3m as the basic element. As shown in Figure 4.13, the approach used can visualize the environment well. In Figure 4.13, the key features common to indoors such as open spaces, stairs, and columns are clearly distinguished. In the visualization, the depth of the map is vividly displayed using color gradients. The map model has met the expected requirements and can be used for tasks such as navigation.

5. CONCLUSIONS AND FUTURE WORK

The main objective of this research is to visualize map points obtained from the SLAM algorithms. The suggested visualization approach included map points data processing using straight-through and statistical filters, a voxel filter for downsampling, and increasing the sampling amount to increase surface smoothness. The processed map points are visualized using Octomap. The applicability of the approach is verified by simulation and experimental results.

The Gazebo on the ROS operating system created a virtual environment that simulates the actual corridor environment. In this virtual environment, the Clearpath Jackal robot is imported. The robot is moved using Interactive Markers to detect the virtual environment. The Octomap is used to visualize the environment. The simulation results show that Octomap can be used to realize real-time 3D map visualization.

Through the study of offline experimental data, it is concluded that the data obtained from the ORB-SLAM algorithm using an RGB-D camera needs data processing before visualization. After visualization, the external form of the environment is vividly displayed. However, due to the limitation of the accuracy of the camera, the internal details of the environment often have similar statistical characteristics with noise, which brings some challenges to data processing and visualization. The best solution is to increase accuracy of data collection.

From the ORB-SLAM results using RGB-D cameras, on one hand, the experimental results shows that the proposed approach has a certain degree of robustness. The approach of this project can be used when the accuracy of the map points that needs to be visualized is low. Different filters are used to reduce noise and provide a respectable visualization of the environment. On the other hand, it is demonstrated that the process of collecting information is very important. When the SLAM results have high accuracy, the approach proposed provides a very good visualization. The approach reduces the complexity of the visualization results without destroying the geometry of the measured environment.

In future work, the filtering algorithm can be improved by using other complicated filters like Binning denoising filter, DBSCAN, and KD-Tree. Binning denoising filtering can be applied to the smoothing of skewed data. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a representative density-based clustering algorithm. DBSCAN is based on the principle of data clustering for denoising. The DBSCAN algorithm can find all dense areas of

sample points and treat these dense areas as clusters one by one. By filtering the size and density of the clusters, point clouds of specific features can be left. This method is relatively complex but can play an important role in large-scale map visualization. KD-Tree filtering is to construct a KD tree, randomly select points to find the average distance, and delete all points that are greater than twice the average distance. It is suitable for out-of-order point cloud denoising.

All three filters mentioned above play important roles in reducing noise. It should be noted that more targeted filters are far more complex than the commonly used filters mentioned in this research. This places higher demands on the computing power of the processor. It is suggested to upgrade the computer hardware by adding memory and upgrading the CPU to improve the computational power. In such a platform, complex dynamic environments with furniture, doors, windows, and pedestrians can be visualized using Octomap.

In addition, it is suggested to incorporate 2D maps obtained by radar or LIDAR to optimize the wall surface in 3D maps obtained by VSLAM. The high-resolution characteristics of radar or LIDAR can describe a wide range of simple environments such as long walls. However, the monotonous wall lacking characteristic points is precisely what VSLAM is not good at. Therefore, combination of the two can improve the visualization significantly.

APPENDIX A. SOFTWARE ENVIRONMENT

A.1 ROS and ROSCORE

ROS is the abbreviation of the Robot Operating System. ROS is a highly flexible software architecture for writing robotic software programs. It contains a large number of tools, libraries code, and protocols designed to simplify the process and complexity of creating, robust robot behavior across robotic platforms. The prototype of ROS comes from Stanford Artificial Intelligence Robot (STAIR) and Personal Robotics (PR) projects.^[5] ROS designers describe ROS as "ROS = Plumbing + Tools + Capabilities + Ecosystem"^[8], which means that ROS is a collection of communication mechanisms, tool packages, robotic high-level skills, and robotic ecosystems.

ROS is an open-source meta-OS for robots. It provides services including hardware abstraction, underlying device control, implementation of common functions, inter-process messaging, and package management. It also provides the tools and library functions needed to write, and run code across applications.

The main task of ROS is to provide support for robot research and development. ROS is a distributed process. The ROS packages are easy to share and publish. ROS also supports a federated system similar to a code repository, which also enables engineering collaboration and publishing. This design allows a project to be developed and implemented completely independent from the file system to the user interface (not subject to ROS restrictions). At the same time, all projects can be integrated with the basic tools of ROS. The development diagram of ROS is shown in detail in Figure A.1^[9] with different Versions of ROS indicated.

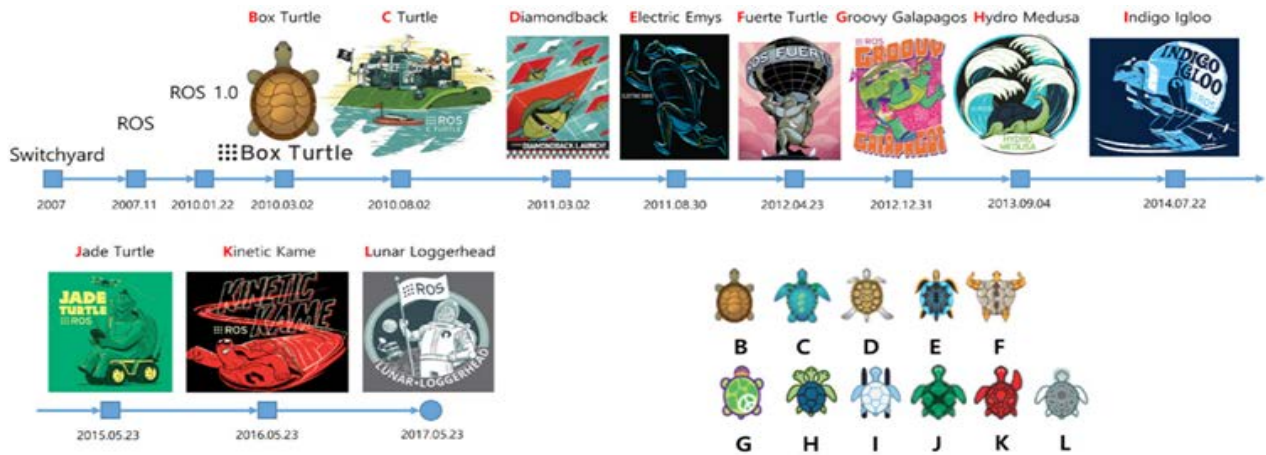


Figure A.1 ROS development stage.

In this project, the Kinetic version is used as the ROS operating system. The reason this version was chosen is that it not only supports the current relatively stable Ubuntu 16 but also runs successfully in a newer hardware environment. At the beginning of the experiment, the Indigo version of ROS with Ubuntu 14 was used. ThinkPad X230i (2012 version) is used at this stage. There are a large number of calculations and graphic construction, which have a considerable degree of requirements on the device graphics card and RAM capacity, and the older software limited the implementation capability. However, it should be noted that Ubuntu 14 lacks the corresponding environmental support for a new generation of graphics cards. As a new PC was purchased, the system was upgraded to Kinetic+Ubuntu 16.

ROSCORE is the first module needed to be run before using all ROS programs. After executing ROSCORE, the following message is shown on the screen as indicated in Figure A.2.

```

tempname@ECE-ROBOT-LAPTOP:~$ roscore
... logging to /home/tempname/.ros/log/f661a2c0-f9dc-11e9-9afd-f8a2d6475899/ros1
launch-ECE-ROBOT-LAPTOP-19758.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ECE-ROBOT-LAPTOP:45067/
ros_comm version 1.12.14

SUMMARY
=====
PARAMETERS
 * /rostdistro: kinetic
 * /rosversion: 1.12.14

NODES
auto-starting new master
process[master]: started with pid [19769]
ROS_MASTER_URI=http://ECE-ROBOT-LAPTOP:11311/

setting /run_id to f661a2c0-f9dc-11e9-9afd-f8a2d6475899
process[roscout-1]: started with pid [19782]
started core service [/roscout]

```

Figure A.2 ROSCORE has run successfully.

ROSCORE provides the environment for Rviz, the visualization software. In the next section, Rviz is explained as a visualization tool in ROS.

A.2 PCL

Point Cloud Library (PCL) is a large open-source programming library for processing 2D/3D images and point clouds. PCL provides general algorithms and efficient data structures, covering a variety of functions, such as point cloud acquisition, filter, segmentation, and visualization^[26]. PCL supports multiple operating system platforms, and PCL originated from a ROS-based study at the Technical University of Munich. Because the main process of visualization in this project is based on the ROS platform, it is very convenient and justified to use PCL on ROS.

From an application perspective, PCL can be used for point cloud segmentation, classification, calibration, and visualization. From a theoretical point of view, the PCL database contains many very mature algorithms that can better help people understand and create new point cloud algorithms. Whether it's for industrial applications or scientific research, PCL can provide practical help in the field of 3D data processing. Taking 3D city modeling as an example, although reconstructing 3D building models is still a challenging problem, point clouds have sufficient capacity for building reconstruction^[27].

PCL can be used on multiple platforms. PCL package can be installed on Windows systems, Linux systems, and Mac systems. In learning PCL, using the Linux system is more efficient. The various information on the point cloud in the PCL files is contained in a key data structure called PointCloud. When ROS calls various information in the point cloud, it is necessary to indicate the area where the data is saved. These locations are called public domains. Table A.1 indicates the essential public domains in the point cloud.

Table A.1 The essential public domain in the point cloud.

PUBLIC DOMAIN	EXPLANATION
Header	This field is the pcl::PCLHeader type and the acquisition time of the specified point cloud
Points	This field is a type of std::vector<PointT,...>, which is a container for point cloud storage. PointT is a class template parameter.
Width	This field specifies the width of the point cloud when organizing an image. Otherwise, there is only one.
Height	This field specifies the height of the point cloud. If there is no specification, there is only one.
Is_dense	This field specifies whether the point cloud contains invalid values (infinite or NaN)
Sensor_origin	This field is of type Eigen::Vector4f, which defines the sensor's acquisition pose in terms of a region's transition.
Sensor_orientation	This field is of type Eigen::Quaternion, which defines the sensor as a rotation angle.

PCL provides communication of point cloud data structures to the ROS interface based on a message. The communication system provided by ROS can import and export data ^[28]. Users can check PCL data in the Rviz software under the ROS platform. In addition, point cloud files can be published through themes, where they can be read and visualized using other service nodes. Octomap can read point cloud data and generate an image composed of several cubes. Rviz will provide the visualization tool for subsequent Octomap construction

The PCL library already exists in the public software source and is installed directly.

sudo apt-get install libpcl-dev

Now, the compiled point cloud library (PCL-1.7), and dependent libraries such as VTK-6.2.0 is installed.

The compilation process after installation may indicate the library file vtkproj4.so is missing To solve this problem by providing the library link as shown below:

sudo ln -s /usr/lib/libvtkproj4.so.5.10 /usr/lib/libvtkproj4.so

During the compilation, there will also be missing dependency libraries that make it impossible to use all the functions of PCL. You can use the following instructions to install all dependent libraries.


```

sudo apt-get update
sudo apt-get install git build-essential linux-libc-dev
sudo apt-get install cmake cmake-gui
sudo apt-get install libusb-1.0-0-dev libusb-dev libudev-dev
sudo apt-get install mpi-default-dev openmpi-bin openmpi-common
sudo apt-get install libflann1.8 libflann-dev
sudo apt-get install libeigen3-dev
sudo apt-get install libboost-all-dev
sudo apt-get install libvtk5.10-qt4 libvtk5.10 libvtk5-dev
sudo apt-get install libqhull* libgtest-dev
sudo apt-get install freeglut3-dev pkg-config
sudo apt-get install libxmu-dev libxi-dev
sudo apt-get install mono-complete
sudo apt-get install qt-sdk openjdk-8-jdk openjdk-8-jre

```

At this point, the PCL installation is complete. In order to conveniently preview the point cloud before or during processing, pcl-viewer is a commonly used PCL visualization tool. It can be downloaded by the following instruction.

```

sudo apt-get install pcl-tools

```

A.3 Gazebo

A.3.1 Reinstall Gazebo and Map Editor

If Gazebo cannot be opened, the following commands are executed:

```

$ sudo apt-get install ros-kinetic-gazebo-ros-pkgs ros-kinetic-gazebo-ros-control

```

After ensuring that the installation is complete, the following commands can be used in the terminal to start ROS and gazebo:

```

$ roscore
$ rosrun gazebo_ros gazebo

```

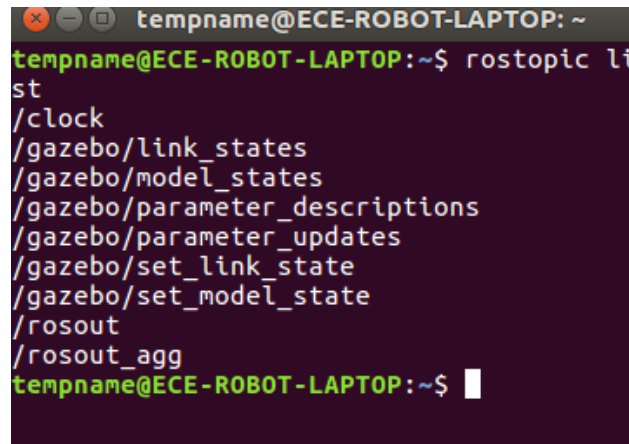
Check the ROS topic list to verify that Gazebo is successfully connected to the ROS system:

```

$ rostopic list

```

If the connection is successful, the list of topics that publishes/subscribe will be shown. Nodes can post messages to topics or subscribe to certain topics to receive messages on those topics. When the topic list is successfully called, the interface shown in Figure A.3 will be displayed.

A terminal window with a dark background and light-colored text. The window title is 'tempname@ECE-ROBOT-LAPTOP: ~'. The prompt is 'tempname@ECE-ROBOT-LAPTOP:~\$'. The command 'rostopic list' has been entered. The output is a list of ROS topics: '/clock', '/gazebo/link_states', '/gazebo/model_states', '/gazebo/parameter_descriptions', '/gazebo/parameter_updates', '/gazebo/set_link_state', '/gazebo/set_model_state', '/rosout', and '/rosout_agg'. The prompt 'tempname@ECE-ROBOT-LAPTOP:~\$' is followed by a cursor.

```
tempname@ECE-ROBOT-LAPTOP: ~
tempname@ECE-ROBOT-LAPTOP:~$ rostopic list
/clock
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/rosout
/rosout_agg
tempname@ECE-ROBOT-LAPTOP:~$
```

Figure A.3 List of topics in gazebo.

After the preparation is completed, the virtual environment can be officially started. First, draw the test area map manually using the Building Editor tool provided by Gazebo. Building Editor interface can be opened by selecting Edit --> Building Editor from the gazebo menu bar. As shown in Figure A.4, select the drawing option on the left and use the mouse to draw in the upper window. The simulated environment can be displayed in real-time in the lower window.

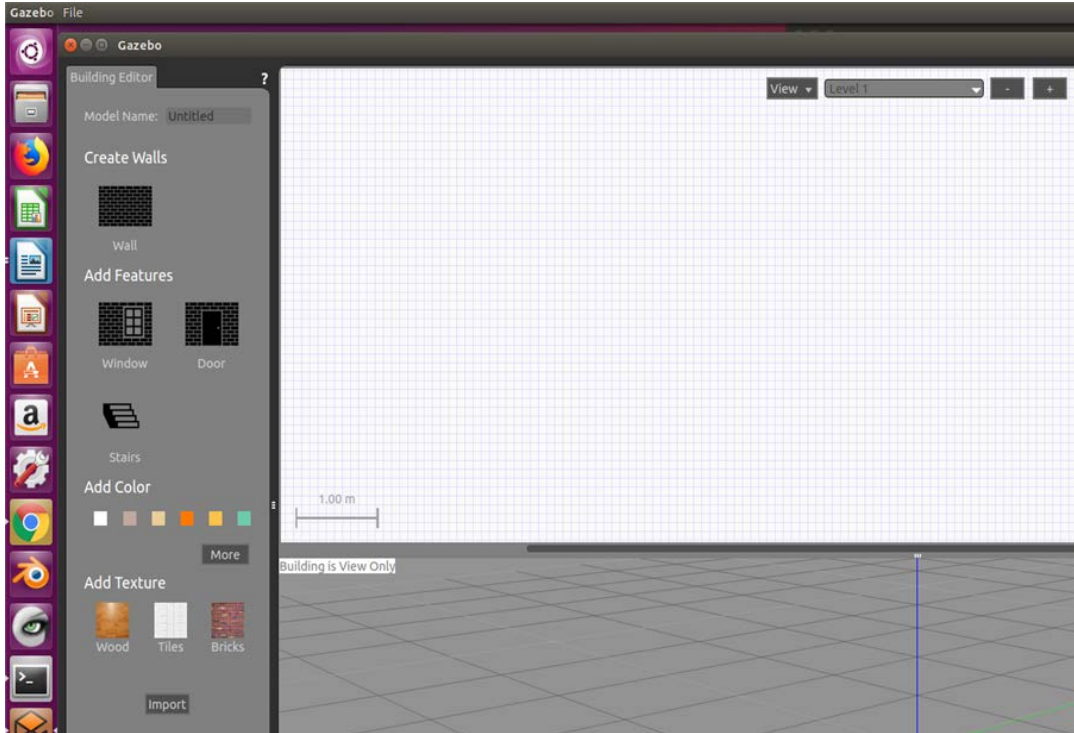


Figure A.4 Editor interface.

The robot used for data collection is Jackal from Clearpath Robotics. Fortunately, Clearpath Robotics provides guidance on how to simulate the robot ^[15]. The robot behavior can be simulated as described in the Jackal tutorials.

A.3.2 Simulate the Jakal Robot

To simulate the Jakal robot, install the *Jackal-specific metapackages for desktop*. It is crucial to enter the version of ROS used in this project. The install command is as follows.

```
sudo apt-get install ros-kinetic-jackal-simulator ros-kinetic-jackal-desktop
```

Gazebo is the most common simulation tool used in ROS. Jackal's model in Gazebo includes reasonable approximations of its dynamics, including wheel slippage, skidding, and inertia. The following command is used to import a 3D model of Clearpath Jackal.

```
roslaunch jackal_gazebo jackal_world.launch config:=front_laser
```

A.4 Rviz

In computational science and computer graphics, there is a strong desire to represent and visualize information in real-time, and many visual data structures and algorithms have been proposed to accomplish this. Unfortunately, the data flow model that is often chosen to solve in the visualization system is not flexible enough to visualize newly invented data structures and algorithms because of specific data structures.

To solve this problem, a new ROS-based visualization tool called Rviz was developed. It has many advantages. Mainly, the data structure is independent of the input information. Since no additional effort is required to manage the streaming network, and in Rviz, the interface to abstract information is simple ^[31].

There are a lot of control options in Rviz. Table A.2 briefly describes the primary function of Rviz^[11].

Table A.2 Control options in Rviz.

NAME	DESCRIPTION	MESSAGES USED
Axes	Displays a set of Axes.	
Effort	Shows the effort being put into each revolute joint of a robot.	sensor_msgs/JointStates
Camera	Creates a new rendering window from the perspective of a camera, and overlays the image on top of it.	sensor_msgs/Image, sensor_msgs/CameraInfo
Grid	Displays a 2D or 3D grid along a plane.	
Grid Cells	Draws cells from a grid, usual obstacles from a cost map from the navigation stack.	nav_msgs/GridCells
Image	Creates a new rendering window with an Image. Unlike	sensor_msgs/Image

	the Camera display, this display does not use a CameraInfo. Version: Diamondback+	
Interactive Marker	Displays 3D objects from one or multiple Interactive Marker servers and allows mouse interaction with them. Version: Electric+	visualization_msgs/InteractiveMarker
Laser Scan	Shows data from a laser scan, with different options.	sensor_msgs/LaserScan
Map	Displays a map on the ground plane.	nav_msgs/OccupancyGrid
Markers	It allows programmers to display arbitrary primitive shapes through a topic.	visualization_msgs/Marker, visualization_msgs/MarkerArray
Path	Shows a path from the navigation stack.	nav_msgs/Path
Point	Draws a point as a small sphere.	geometry_msgs/PointStamped
Pose	Draws a pose as either an arrow or axes.	geometry_msgs/PoseStamped
Point Cloud/ Point Cloud(2)	Shows data from a point cloud, with different options	sensor_msgs/PointCloud, sensor_msgs/PointCloud2

Three-dimensional(3D) data is an exciting category of information. From the ocean to the deep space, from the city to the countryside, 3D data are everywhere. But how can computers deal with 3D data?

The transmission of information requires an expression format and requires a data type. Therefore, people set various data types such as point clouds and depth maps^[10] to draw a 3D

world. Among them, point cloud data is a kind of very basic 3D data representation. In order to use point cloud data more efficiently for visualization on the ROS platform, you can use the large cross-platform open-source C++ programming library Point Cloud Library (PCL).

A.5 The RGB-D Camera Module SLAM Package

The RGB-D camera module is installed as follows:

First, it is needed to clone the package to the workspace by.

```
$ cd ~/catkin_ws/src
```

```
$ git clone https://github.com/DroidAITech/ROS-Academy-for-Beginners.git
```

Afterward, install the dependencies.

```
$ cd ~/catkin_ws
```

```
$ rosdep install --from-paths src --ignore-src --rosdistro=kinetic -y
```

After the dependencies are installed, the new environment is refreshed by

```
$ catkin_make
```

```
$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

```
$ source ~/.bashrc
```

Then, a series of function packages are installed on the personal computer as the master computer. The ORB-SLAM2^[49] function package based on the environment data packet and RGB-D camera in the Gazebo simulation environment is needed. ORB-SLAM2 is a real-time SLAM library for Monocular, Stereo and RGB-D cameras that computes the camera trajectory and a sparse 3D reconstruction. The supplier of this simulation environment is the Institute of Software of the Chinese Academy of Sciences. Before compiling, install the libraries needed for lightweight OpenGL input/output and video display that encapsulates OpenGL. And the feature pack requires the OpenCV Vision Library to process images and features, but OpenCV does not need to be installed before compiling ORB_SLAM2. But these tool libraries don't have to be in the workspace. To install the link library, use the following link

```
$ sudo apt-get install libboost-all-dev libblas-dev liblapack-dev
```

After this, the ORB_SLAM2 source package is available at

```
$ cd ~/catkin_ws/src
```

```
$ git clone https://github.com/raulmur/ORB_SLAM2.git
```

After the ORB_SLAM2 package is available, you can complete the compilation and installation of Eigen, Pangolin, and OpenCV as supporting elements in environmental data packages. Then compile the ORB_SLAM2 feature pack.

```
$ export
ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:/catkin_ws/src/ORB_SLAM2/Examples/
ROS
$ cd ORB_SLAM2
$ chmod +x build.sh
$ ./build.sh
$ chmod +x build_ros.sh
$ ./build_ros.sh

$ cd
$ gedit .bashrc
#If you add the following code to the end of the file, it will always take effect.
$ export
ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:/catkin_ws/src/ORB_SLAM2/Examples/
ROS
```

Finally, use the command below to install the last dependency.

```
$ sudo apt-get -y install libopencv-dev build-essential cmake git libgtk2.0-dev pkg-config p
```

Start Roscore, then modify the ORB_SLAM2/Examples/ROS/ORB_SLAM2/src/ros_rgbd.cc source file before starting the ORB_SLAM2 function, because the topic name published by the robot in the simulation environment is different from the topic name of the algorithm subscription, os_rgbd. The topic name subscribed to in the .cc source file needs to be modified to the name published by the robot. The RGB-D camera will simultaneously publish the two topics of the color image and depth image and then recompile the ORB_SLAM2 function package.

Firstly, modify the ORB_SLAM2/Examples/ROS/ORB_SLAM2/src/ros_rgbd.cc source file

```
Message_filters::Subscriber<sensor_msgs::Image>depth_sub(nh,  
"camera/depth_registered/image_raw", 1);  
#depth_registered Change to depth  
Message_filters::Subscriber<sensor_msgs::Image> depth_sub(nh, "camera/depth/image_raw",  
1);
```

Next open a new terminal and start the ORB_SLAM function node

```
$ roslaunch orbslam2_demo ros_orbslam2.launch
```

Open a new terminal and start the gazebo simulation environment.

```
$ roslaunch robot_sim_demo robot_spawn.launch
```

At this time, Gazebo is ready for simulation. It is necessary to prepare Rviz for visualization. Whether it is processing offline data experiments or simulation data in a virtual environment, all the visualization in this research is done in Rviz. After the layout is completed, the sensor data can be read into the Octomap. After that, Rviz can visualize maps that are enough for navigation, terrain characterization, and more.

In order to display Octomap in real-time, it needs to be implemented using Rviz. The idea is to publish the point cloud data to a topic via ROS, such as "/outputCloud", and then start the Octomap node to read the data into the topic and publish it to another new topic. Finally, receive this new topic in Rviz for real-time display.

ROS is needed to support both Gazebo and Rviz, so we open a terminal (ctrl+alt+T) and enter the following command to install the Octomap-related Rviz component.

```
sudo apt-get install ros-kinetic-octomap-ros  
sudo apt-get install ros-kinetic-octomap-msgs  
sudo apt-get install ros-kinetic-octomap-server  
sudo apt-get install ros-kinetic-octomap-rviz-plugins
```

If the Octomap plugin is installed and Rviz is launched, there will be a folder called 'Octomap_rviz_plugins' as shown in Figure A.5.

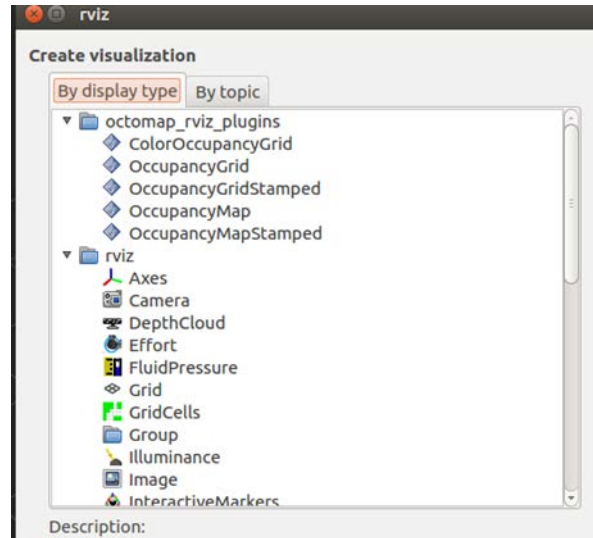


Figure A.5 Rviz adding menu with Octomap tools.

In order to load the point cloud file and then publish it through a topic, A cpp file is created as follows.

`<launch>`

```
<node pkg="octomap_server" type="octomap_server_node" name="octomap_server">
```

```
<!-- resolution in meters per pixel -->
```

```
<param name="resolution" value="0.05" /> //keep same value when doing real experiment
```

```
<!-- name of the fixed frame, needs to be "/map" for SLAM -->
```

```
<param name="frame_id" type="string" value="/camera" />
```

```
<!-- max range / depth resolution of the kinect in meter -->
```

```
<param name="sensor_model/max_range" value="100.0" /> //default values
```

```
<param name="latch" value="true" />
```

```
<!-- max/min height for occupancy map, should be in meters -->
```

```
<param name="pointcloud_max_z" value="1000" /> //default values
```

```
<param name="pointcloud_min_z" value="0" />
```

```
<!-- topic from where pointcloud2 messages are subscribed -->  
<remap from="/cloud_in" to="/pointcloud/output" />
```

```
</node>
```

```
</launch>
```

After building the launch file, start the point cloud publishing node.

```
Rosrun publish_pointcloud publish_pointcloud
```

After starting this node, the Octomap service node can be started.

```
roslaunch publish_pointcloud octomaptransform.launch
```

The following launch invocation to start Rviz with standard Jackal.

```
roslaunch jackal_viz view_robot.launch
```

APPENDIX B. OFFLINE DATA VISUALIZATION

B.1 Map Points Data Processing

B.1.1 Preliminary Processing

The following steps are required. First, the point cloud data obtained by the RGB-D SLAM is imported into the PCL system, which is implemented by :

```
cloud->points.clear();
std::string pcd_file_name;
std::string pcd_file_name_filtered;
std::cout<<"please input pcd file name:"<<endl;
std::cin>>pcd_file_name;
std::cout<<"the filtered pcd named *_filtered.pcd "<<endl;
pcl::io::loadPCDFile(pcd_file_name, *cloud);
```

After the map data is successfully imported, a straight-through filter is used to cut out the obviously wrong point in the Z-axis direction, which is implemented by :

```
pcl::PassThrough<pcl::PointXYZRGBA> pass;
pass.setInputCloud (cloud);
pass.setFilterFieldName ("z")
pass.setFilterLimits (0, 5000); //height limit is 5m
pass.filter (*cloud_medium);
```

After clearing out the obvious error points, the statistical filter is used to delete the outliers. After several trials, the number of neighboring points near the query point was selected to be with a threshold of 2.

```
pcl::StatisticalOutlierRemoval<pcl::PointXYZRGBA> Static;
Static.setInputCloud (cloud_medium);
Static.setMeanK (20);
Static.setStddevMulThresh (2); // scaling factor  $\alpha$ 
Static.filter (*cloud_filtered);
```

B.1.2 Downsampling

The specific implementation for downsampling is as follows:

```
// Create the filtering object  
pcl::VoxelGrid<pcl::PointXYZRGBA> sor;  
sor.setInputCloud(cloud);  
sor.setLeafSize(0.03f, 0.03f, 0.03f);  
sor.filter(*cloud);
```

B.1.3 Increase Sampling Step

The following steps are taken to increase sampling is as follows.

```
pcl::MovingLeastSquares<pcl::PointXYZ, pcl::PointXYZ> filter;  
filter.setInputCloud(cloud);  
pcl::search::KdTree<pcl::PointXYZ>::Ptr kdtree;  
filter.setSearchMethod(kdtree);  
filter.setSearchRadius(0.3); // Set the search neighborhood radius to 0.3m  
filter.setUpsamplingMethod(pcl::MovingLeastSquares<pcl::PointXYZ,  
pcl::PointXYZ>::SAMPLE_LOCAL_PLANE);  
filter.setUpsamplingRadius(0.02); // the radius of the sample determines the thickness of the wall  
filter.setUpsamplingStepSize(0.005); // the number of sampling steps determines the number of  
points after processing  
filter.process(*filteredCloud);
```

Kd-Tree is a data structure, which is a special case of a spatial binary tree, which can be conveniently used for range search. KD-Tree is used here to facilitate management and search for point clouds. This structure can easily find the nearest neighbors. *filter.setSearchRadius (0.3)* means to search all points in the space with the current point with a radius of 0.3m. The larger the search radius, the greater the smoothness. Then, fit these points with a 2nd order polynomial. Both, the radius of the sampling and the number of sampling steps use the default values of the MLS feature pack.

B.2 Publishing and Visualizing Point Cloud Data

Load this point cloud and publish through a topic by using *octomaptransform.launch*. The *octomaptransform.launch* file is shown below.

```
#include<iostream>
#include<string>
#include <stdlib.h>
#include <stdio.h>
#include <sstream>
#include <vector>

#include<ros/ros.h>
#include<pcl/point_cloud.h>
#include<pcl_conversions/pcl_conversions.h>
#include<sensor_msgs/PointCloud2.h>
#include<pcl/io/pcd_io.h>

using namespace std;

int main (int argc, char **argv)
{
    std::string topic,path,frame_id;
    int hz=5;

    ros::init (argc, argv, "publish_pointcloud");
    ros::NodeHandle nh;

    nh.param<std::string>("path", path, "/home/tempname/catkin_ws/test.pcd");
    nh.param<std::string>("frame_id", frame_id, "camera");
    nh.param<std::string>("topic", topic, "/pointcloud/output");
    nh.param<int>("hz", hz, 5);
```

```
ros::Publisher pcl_pub = nh.advertise<sensor_msgs::PointCloud2> (topic, 10);
```

```
pcl::PointCloud<pcl::PointXYZ> cloud;
```

```
sensor_msgs::PointCloud2 output;
```

```
pcl::io::loadPCDFile (path, cloud);
```

```
pcl::toROSMsg(cloud,output);// Converted into data types under ROS and finally  
published through topics
```

```
output.header.stamp=ros::Time::now();
```

```
output.header.frame_id =frame_id;
```

```
cout<<"path = "<<path<<endl;
```

```
cout<<"frame_id = "<<frame_id<<endl;
```

```
cout<<"topic = "<<topic<<endl;
```

```
cout<<"hz = "<<hz<<endl;
```

```
ros::Rate loop_rate(hz);
```

```
while (ros::ok())
```

```
{
```

```
    pcl_pub.publish(output);
```

```
    ros::spinOnce();
```

```
    loop_rate.sleep();
```

```
}
```

```
return 0;
```

```
}
```

The coordinate system need to be set as "camera" in Rviz, which is the same as the *frame_id* parameter in the *octomaptransform.launch* file. Otherwise, Octomap may not release data. At this point, go to the folder where *octomaptransform.launch* exists, call up a terminal (Ctrl + alt + T), and start the point cloud publishing node separately as shown below:

```
roslaunch publish_pointcloud publish_pointcloud
```

Open a new terminal and run RVIZ:

```
roslaunch rviz rviz
```

Click the Add button to add the "PointCloud2 Module". Set the theme to "/ pointcloud / outputOcoto" and FixedFram to "camera"

REFERENCES

- [1] Goldsman, D., Nance, R. E., & Wilson, J. R. (2010). A brief history of simulation revisited. In Proceedings of the 2010 Winter Simulation Conference. pp. 567-574.
- [2] Heermann, D. W. (1990). Computer-simulation methods. In Computer Simulation Methods in Theoretical Physics. pp. 8-12.
- [3] Durrant-Whyte, H.; Bailey, T. (2006). "Simultaneous localization and mapping: part I". IEEE Robotics & Automation Magazine. 13 (2). pp. 99–110.
- [4] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., & Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees. Autonomous robots, 34(3). pp.189-206.
- [5] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009). ROS: an open-source Robot Operating System. In ICRA workshop on open source software Vol. 3, No. 3.2. pp. 5.
- [6] Kam, H. R., Lee, S. H., Park, T., & Kim, C. H. (2015). Rviz: a toolkit for real domain data visualization. Telecommunication Systems, 60(2). pp. 337-345.
- [7] Zou, D., & Tan, P. (2012). Coslam: Collaborative visual slam in dynamic environments. IEEE transactions on pattern analysis and machine intelligence, 35(2). pp. 354-366.
- [8] About ROS. (2020). Retrieved from <https://www.ros.org/about-ros/>
- [9] History. (2020). Retrieved from <https://www.ros.org/history/>
- [10] Muller, K., Merkle, P., & Wiegand, T. (2010). 3-D video representation using depth maps. Proceedings of the IEEE, 99(4). pp. 643-656.
- [11] Primary function of ROS. Retrieved from <http://wiki.ros.org/rviz/UserGuide#Startup>
- [12] Filtering in PCL Targeted. Retrieved from <https://zhuanlan.zhihu.com/p/54345510>
- [13] Brief introduction of Octomap. Retrieved from <https://www.cnblogs.com/gaoxiang12/p/5041142.html>
- [14] Wikipedia Octree. Retrieved from <https://de.wikipedia.org/wiki/Octree>
- [15] SIMULATING JACKAL. Retrieved from <http://www.clearpathrobotics.com/assets/guides/jackal/simulation.html>

- [16] Davison, A. J., Cid, Y. G., & Kita, N. (2004). Real-time 3D SLAM with wide-angle vision. *IFAC Proceedings Volumes*, 37(8). pp. 868-873.
- [17] Chang, H. J., Lee, C. G., Hu, Y. C., & Lu, Y. H. (2007). Multi-robot SLAM with topological/metric maps. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 1467-1472.
- [18] Magnabosco, M.; Breckon, T.P. (2013). "Cross-Spectral Visual Simultaneous Localization And Mapping (SLAM) with Sensor Handover" (PDF). *Robotics and Autonomous Systems*. 63 (2). pp. 195–208.
- [19] Strasdat, H., Montiel, J. M., & Davison, A. J. (2012). Visual SLAM: why filter?. *Image and Vision Computing*, 30(2). pp.65-77.
- [20] Gossow, D., Leeper, A., Hershberger, D., & Ciocarlie, M. (2011). Interactive markers: 3-d user interfaces for ROS applications [ros topics]. *IEEE Robotics & Automation Magazine*, 18(4). pp.14-15.
- [21] Angeli, A., Doncieux, S., Meyer, J. A., & Filliat, D. (2008). Incremental vision-based topological SLAM. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 1031-1036.
- [22] Endres, F., Hess, J., Engelhard, N., Sturm, J., Cremers, D., & Burgard, W. (2012). An evaluation of the RGB-D SLAM system. In *2012 IEEE International Conference on Robotics and Automation*. pp. 1691-1696.
- [23] Zhang, Peter & Millos, Evangelous & Gu, Jason. (2009). General Concept of 3D SLAM. 10.5772/6993.
- [24] Chen, C. W., Chen, W. Z., Peng, J. W., Cheng, B. X., Pan, T. Y., & Kuo, H. C. (2017). A Real-Time Markerless Augmented Reality Framework Based on SLAM Technique. In *2017 14th International Symposium on Pervasive Systems, Algorithms and Networks*. pp. 127-132.
- [25] Meng, W., Hu, Y., Lin, J., Lin, F., & Teo, R. (2015). ROS+ unity: An efficient high-fidelity 3D multi-UAV navigation and control simulator in GPS-denied environments. In *IECON 2015-41st Annual Conference of the IEEE Industrial Electronics Society*. pp. 2562-2567.
- [26] Rusu, R. B., & Cousins, S. (2011). 3d is here: Point cloud library (PCL). In *2011 IEEE international conference on robotics and automation* . pp. 1-4.

- [27] Malihi, S., Zoej, M. V., Hahn, M., Mokhtarzade, M., & Arefi, H. (2016). 3D building reconstruction using dense photogrammetric point cloud. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 3, pp. 71.
- [28] Będkowski, J., Pełka, M., Majek, K., Fitri, T., & Naruniec, J. (2015). Open source robotic 3D mapping framework with ROS—Robot Operating System, PCL—Point Cloud Library and Cloud Compare. In *2015 International Conference on Electrical Engineering and Informatics (ICEEI)*. pp. 644-649.
- [29] Papon, J., Abramov, A., Schoeler, M., & Worgotter, F. (2013). Voxel cloud connectivity segmentation-supervoxels for point clouds. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 2027-2034.
- [30] Miknis, M., Davies, R., Plassmann, P., & Ware, A. (2015). Near real-time point cloud processing using the PCL. In *2015 International Conference on Systems, Signals and Image Processing (IWSSIP)* . pp. 153-156.
- [31] Hershberger, D., Gossow, D., & Faust, J.(2016). RViz, 3D visualization tool for ROS. Retrieved from <http://wiki.ros.org/rviz>.
- [32] Takaya, K., Asai, T., Kroumov, V., & Smarandache, F. (2016). Simulation environment for mobile robots testing using ROS and Gazebo. In *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)* . pp. 96-101.
- [33] Afanasyev, I., Sagitov, A., & Magid, E. (2015). ROS-based SLAM for a Gazebo-simulated mobile robot in image-based 3D model of indoor environment. *International Conference on Advanced Concepts for Intelligent Vision Systems*. pp. 273-283.
- [34] Koenig, N., & Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*(IEEE Cat. No. 04CH37566). pp. 2149-2154.
- [35] Odometry and Encoders. Retrieved from <http://ttuadvancedrobotics.wikidot.com/odometry>
- [36] Toledo, J., Piñeiro, J. D., Arnay, R., Acosta, D., & Acosta, L. (2018). Improving odometric accuracy for an autonomous electric cart. *Sensors*, 18(1). pp. 200.
- [37] NOAA.(2020)What is LIDAR? National Ocean Service website, Retrieved from <https://oceanservice.noaa.gov/facts/lidar.html>
- [38] Seurat, G. (2020). A Sunday on La Grande Jatte - 1884. Retrieved from <https://www.artic.edu/artworks/27992/a-sunday-on-la-grande-jatte-1884>

- [39] Wolf, J., Burgard, W., & Burkhardt, H. (2002). Robust vision-based localization for mobile robots using an image retrieval system based on invariant features. In Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292) .pp. 359-365.
- [40] Taketomi, T., Uchiyama, H., & Ikeda, S. (2017). Visual SLAM algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1), pp. 1.
- [41] Augenstein, S., & Rock, S. M. (2011). Improved frame-to-frame pose tracking during vision-only SLAM/SFM with a tumbling target. In 2011 IEEE International Conference on Robotics and Automation. pp. 3131-3138.
- [42] Klein, G., & Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. In 2007 6th IEEE and ACM international symposium on mixed and augmented reality. pp. 225-234.
- [43] Tian, Y., Meng, X., Tao, D., Liu, D., & Feng, C. (2015). Upper limb motion tracking with the integration of IMU and Kinect. *Neurocomputing*, 159. pp. 207-218.
- [44] Engel, J., Schöps, T., & Cremers, D. (2014). LSD-SLAM: Large-scale direct monocular SLAM. In European conference on computer vision. pp. 834-849.
- [45] Weingarten, J., & Siegwart, R. (2005). EKF-based 3D SLAM for structured environment reconstruction. In 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 3834-3839.
- [46] Ahmad, N., Ghazilla, R. A. R., Khairi, N. M., & Kasi, V. (2013). Reviews on various inertial measurement unit (IMU) sensor applications. *International Journal of Signal Processing Systems*, 1(2). pp. 256-262.
- [47] Minecraft. (2020). Retrieved from <https://en.wikipedia.org/wiki/Minecraft>
- [48] Xiaoqing, W., & Yuqing, H. (2017). Smoothing and resampling of point cloud based on moving least squares. *Microcomputer & Its Applications*, (11). pp.14.
- [49] Mur-Artal, R., & Tardós, J. D. (2017). ORB-SLAM2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5). pp.1255-1262.
- [50] Wasenmüller, O., & Stricker, D. (2016, November). Comparison of kinect v1 and v2 depth images in terms of accuracy and precision. In Asian Conference on Computer Vision. pp. 34-45.

- [51] Technische Universität München (2020). Datasets.
Retrieved from <https://vision.in.tum.de/data/datasets>
- [52] Intel® Xeon® Processor E5-2630 v3
Retrieved from <https://ark.intel.com/content/www/us/en/ark/products/83356/intel-xeon-processor-e5-2630-v3-20m-cache-2-40-ghz.html>
- [53] Kazhdan, M., Bolitho, M., & Hoppe, H. (2006). Poisson surface reconstruction. In Proceedings of the fourth Eurographics symposium on Geometry processing (Vol. 7).
- [54] Maurya, S. R., & Magar, G. M. (2018). Performance of Greedy Triangulation Algorithm on Reconstruction of Coastal Dune Surface. In 2018 3rd International Conference for Convergence in Technology (I2CT) .pp. 1-6
- [55] Fischler, M. A., & Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24(6), 381-395.
- [56] Edward Rosten, Reid Porter and Tom Drummond, "FASTER and better: A machine learning approach to corner detection". IEEE Transactions on Pattern Analysis and Machine Intelligence, 2010, pp 105–119.
- [57] Michael Calonder, Vincent Lepetit, Christoph Strecha and Pascal Fua, “BRIEF: Binary Robust Independent Elementary Features.” European Conference on Computer Vision (ECCV) 2010, pp 778-792.
- [58] Raúl Mur-Artal, J. M. M. Montiel and Juan D. Tardós, “ORB-SLAM: A Versatile and Accurate Monocular SLAM System.” IEEE Transactions on Robotics, October 2015, Vol. 31, No. 5, pp. 1147-1163.
- [59] Forster, C., Pizzoli, M. and Scaramuzza, D. (2014). SVO: Fast semi-direct monocular visual odometry. IEEE International Conference on Robotics and Automation (ICRA), 2014
- [60] Galvez-Lopez, Dorian, and Juan D. Tardos. “Real-Time Loop Detection with Bags of Binary Words.” 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011.
- [61] Jutta Willamowski, Damian Arregui, Gabriella Csurka, Christopher R. Dance, Lixin Fan, "Categorizing nine visual classes using local appearance descriptors." Icpr Workshop on Learning for Adaptable Visual Systems (2004).
- [62] Horn, Berthold K. P, “Closed-Form Solution of Absolute Orientation Using Unit Quaternions.” Journal of the Optical Society of America A, Jan. 1987, Vol. 4, No. 4, p. 629.