HIGH-PERFORMANCE 3D MAPPING OF UNKNOWN ENVIRONMENTS USING PARALLEL COMPUTING FOR MOBILE ROBOTS

K. T. D. S. De Silva

198011G

Degree of Master of Science/ Master of Engineering

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

August 2021

HIGH-PERFORMANCE 3D MAPPING OF UNKNOWN ENVIRONMENTS USING PARALLEL COMPUTING FOR MOBILE ROBOTS

K. T. D. S. De Silva

198011G

Thesis submitted in partial fulfillment of the requirements for the Degree of Master of Science (Research) in Computer Science and Engineering

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

August 2021

DECLARATION

I declare that this is my own work, and this dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name of the candidate: K. T. D. S. De Silva

Signature:

Date: 13/12/2021

The above candidate has carried out research for the Master's thesis/dissertation under my supervision.

Name of the supervisor: Dr. C. D. Gamage
Signature of the supervisor:

Date: 13.12.2021

Name of the supervisor: Dr. S. J. Sooriyaarachchi
Signature of the supervisor:

Date: 14/12/2021

ABSTRACT

Autonomous multi-robot systems are a popular research field in the 3D mapping of unknown environments. High fault tolerance, increased accuracy, and low latency in coverage are the main reasons why a multi-robot system is preferred over a single robot in an unpredictable field. Compared with 3D scene reconstruction which is a conceptually similar but resource-wise different technique, autonomous mobile robot 3D mapping techniques are missing a crucial element. Since most mobile robots run on low computationally powered processing units, the real-time registration of point clouds into high-resolution 3D occupancy grid maps is a challenge.

Until recently, it was nearly impossible to perform parallel point cloud registration in mobile platforms. Serial processing of a large amount of high-frequency input data leads to buffer overflows and failure to include all information into the 3D map. With the introduction of Graphical Processing Units (GPUs) into commodity hardware, mobile robot 3D mapping now can achieve faster time performance, using the same algorithmic techniques as 3D scene reconstruction. However, parallelization of mobile robot 3D occupancy grid mapping process is a less frequently discussed topic.

As a Central Processing Unit (CPU) is necessary to run conventional middleware, operating system, and hardware drivers, the system is developed as a CPU-GPU mixed pipeline. The precomputed *free scan mask* is used to accelerate the process of identifying free voxels in space. Point positional information is transformed into unsinged integer coordinates to cope with Morton codes, which is a linear representation of octree nodes instead of traditional spatial octrees. 64-bit M-codes and 32-bit RGBO-codes are stored in a hash table to reduce access time compared to a hierarchical octree.

Point cloud transformation, ray tracing, mapping point coordinated into integer scale, Morton-coded voxel generation, RGBO-code generation are the processes that are performed inside the GPU. Retrieving point cloud information, map update using bitwise operations and map publish are executed within the CPU.

Additionally, a multi-robot system is prototyped as a team of wheeled robots autonomously exploring an unknown, even-surfaced environment, while building and merging fast 3D occupancy grid maps and communicating using a multi-master communication protocol.

Keywords – Multi-robot system, GPU acceleration, 3D mapping, occupancy grids, point cloud registration, ray tracing, free scan mask, Morton order, linear octree

ACKNOWLEDGEMENT

This thesis is a result of the guidance and encouragement of various personnel. Without them, I wouldn't be able to make it a success.

First and foremost, I would like to thank my research supervisors, Dr. Chandana Gamage and Dr. Sulochana Sooriyaarachchi. They constantly supported me with technical guidance throughout the research and steered my research in the right direction. They also provided me necessary motivation during the Covid-19 lockdown period.

Next, I thank my progress review panel, Dr. Chathura De Silva and Dr. Thilina Lalitharatne, for dedicating their time and providing me valuable feedback for improvement.

This research was funded by Xavier AI Pvt. Ltd., Previously known as Rebirth Technologies Pvt. Ltd., Sri Lanka. A special thanks go to them for their interest in financial investment, and feedback on taking this project into a commercial system.

I would like to thank my friends at the final year project group for laying the groundwork for this research and the initial prototype development. Also, the support and company of the staff of IntelliSense research lab, University of Moratuwa must be appreciated.

Finally, I would like to thank my husband and my family for the encouragement and the immense support they gave me in their best.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AP	Access Point
API	Application Programming Interface
CPU	Central Processing Unit
CUDA	Computed Unified Device Architecture
DSM	Digital Surface Model
Gb	Giga bytes
GPS	Global positioning system
GPU	Graphics Processing Unit
ICP	Iterative closest Point
IMU	Inertial Measurement Unit
IP	Internet Protocol
LUT	Look Up Table
MANET	Mobile Adhoc Network
MAV	Micro Aerial Vehicle
M-code	Morton Code
MCL	Monte Carlo Localization
MVOG	Multi Volume Occupancy Grid
NBV	Next Best View
NDT	Normal Distribution Transform
OS	Operating System
ROS	Robot Operating System
RGBO-code	Red, Green, Blue, Occupancy Code
RRT	Rapidly Exploring Random Trees
SAR	Search And Rescue
SDF	Signed Distance Function
TSDF	Truncated Signed Distance Function
SIMD	Single Instruction Multiple Data
URI	Uniform Resource Identifier

1 INTRODUCTION

1.1 Background

Replacing humans with machines in unpredictable environments has been a popular research area for several decades. Search-and-rescue and damage estimation in disaster sites, cave and underwater mapping, autonomous driving vehicles, and extraplanetary surface exploration are some of the applications that benefit from mobile robotics. The objective of this work under autonomous mobile robotic systems in the above scenarios is, obtaining the maximum coverage of the environment while building a 3D digital representation of the environment. The outcome is a 3D representation of the environment to be used by humans, and the robotic system itself.

In most applications, the initial road map of the environment is not previously available to the robots. Even if it is available, the robots might have to face the situations where the initial map has been changed (i.e.- previously navigable paths blocked due to fallen debris). Therefore, the best approach is to create the maps while exploring and using the same map to find new frontiers.

Within the large research scope of multi-robot 3D mapping and autonomous exploration, this thesis is focused on the 3D map generation process which is the central mechanism of the system. The rest of the components are extracted from a base prototype the author has developed in a previous project for *multi-robot 3D exploration of uneven-surfaced environment* [1], and it is modified according to the research objectives.

Figure 1.1: Two example applications of autonomous mobile robot mapping. (a) Mars Curiosity rover; Source: [2] (b) Underwater operated robots in Fukushima nuclear power plant accident; Source: [3]

1.2 Multi-robot systems

Using a multi-robot system is advantageous for the process of 3D mapping and exploration in unknown environments in several ways. Multiple robots can reduce the time taken by the map building process when the robots are distributed throughout the field. They can increase the accuracy of the map by updating the same map segment multiple times. The system achieves availability and failure proneness with multiple robots operating in the field.

The following features are expected from a multi-robot system operating in the above-discussed scenario.

- Autonomous exploration with the aid of a map which is mostly self-created
- Environmental awareness to respond to sudden changes not indicated in the map
- Ability to perceive the surrounding and create individual 3D maps
- Creating a unified view of the environment with the aid of other robots in the network

Thus, a multi-robot system can provide required coverage speed and accuracy to a certain degree. But if the system needs to further enhance the performance, the 3D map generation process must be improved at a computer engineering design level. High-performance computing hardware devices such as GPUs are widely available in commodity hardware nowadays. Therefore, necessary high-performance software techniques can be implemented to reduce the latency and increase the accuracy of the 3D map generation system.

1.3 Performance measures of 3D mapping

Map is the most essential component of an autonomously navigating mobile robot system. The accuracy and map consistency requirements depend on the mobile robotic application. Agricultural field robots performing tasks like ploughing, fertilizing, and harvesting in large terrains can tolerate a mapping error of some degree, a few meters in this example. In the meantime, more task-oriented applications like warehouse robots and autonomous vehicles require the highest mapping accuracy a modern

system can provide (Figure 1.2). An error of a few centimeters can result in a warehouse robot selecting a wrong item from a shelf or an autonomous driving car risking the safety of other vehicles on a busy road.

In non-time-critical applications, the robots are allowed to allocate a sufficient amount of time for map registration. As an example, an autonomous rover mapping the surface of planet Mars can stay stationary until the local map is complete and slowly continue its exploration. But time-critical applications such as search and rescue missions, and autonomous cars cannot afford to delay their parallel operations waiting for the map registration process to complete. Figure 3 shows the uncertainty of an urban environment where an autonomous driving car is operated. The self-driving car should update its dynamic map, then use it to avoid collision with the pedestrian, cyclist and the other car while maintaining its mobility on the road. A medical robot launched into a large mine is another example of a time-critical 3D mapping application. The robot must create a 3D map with minimum latency and use it for the fidelity of navigation to discover victims.

Figure 1.2: Example applications requiring high accuracy in mapping. (a) a warehouse robot picking up a small package; Source: [4] (b) autonomous vehicles driving in a busy road; Source: [5]

1.4 High-performance computing for 3D map generation

With map building latency being a critical concern in most robotic exploration systems, it also has a vast impact on map quality. If the map registration process cannot cope with the input frame rate, the robot might have to discard a large number of unprocessed input frames. This leads to incomplete maps and robots wasting time rediscovering empty patches in the maps.

In terms of latency, modern scene reconstruction approaches have achieved better results than state-of-the-art mobile robot mapping systems. The reason for their significant time performance is the use of GPU hardware. GPUs extend Single Instruction Multiple Data (SIMD) architecture, which allows thousands of parallel software threads running in the device to achieve a central goal. However, scene reconstruction approaches are used for object modeling, graphic processing in video games, and virtual reality applications. They are not suitable for autonomous mobile robotic systems due to the high memory footprint, and the requirement for advanced and expensive graphical hardware. The scene reconstruction approaches often run entirely on GPUs and map only the occupied space.

As of today, relatively inexpensive GPU cards are being more and more available with commodity hardware such as NVIDIA TX1, TX2 and Xavier. Hence several high-performance computing techniques can be adopted into autonomous mobile robot mapping systems. Since these mapping sub-systems collaborate with other crucial processes running within the robot, it is still impossible to process them entirely on the GPU. Therefore, it is important to identify parallelizable tasks in 3D map generation and migrate them to graphics hardware without affecting other components in the system.

Figure 1.3: Environment uncertainty of an autonomous vehicle; Source: [6]

1.5 The base prototype system

The hardware system prototype of this research is based on the previous work [1]

of the author. The existing multi-robot system prototype has the following features that are useful for this work.

- 3D mapping with OctoMap [7]
- Inter-robot communication with master-slave protocol
- Manual goal based autonomous navigation.

The previous work has been used as the base prototype of this system. The following practical drawbacks of the base prototype were observed.

- OctoMap's map building latency does not meet real-time constraints and mapping accuracy mentioned in section 1.3.
- The master-slave communication method is prone to single point failure and it prefers continuous connectivity of the network.
- Setting manual goals for autonomous navigation does not fulfill autonomous exploration.

Research objectives of this work are built on top of addressing the above-mentioned drawbacks of the base prototype. The expected enhancements are listed as follows.

- Developing a 3D mapping technique that builds maps faster and more accurately than OctoMap [7]
- Integrating a communication protocol that is robust to robot failures and disconnection.
- Integrating an autonomous exploration method to self-assign navigation goals.

1.6 Problem statement

This thesis addresses the problem of,

Developing an autonomous multi-robot system for building an accurate 3D representation of an unknown environment, in real time.

1.7 Objectives

The thesis objectives are, *Implementing a multi-robot 3D mapping system,*

- *To reduce mapping errors and inconsistencies*

- *To build and update maps with the minimum latency*
- *To Enhance the existing multi-robot system prototype by integrating autonomous exploration, and robust inter-robot communication.*

1.8 Research contribution

The thesis contribution can be identified as two outcomes.

(1) G3DOMap - A GPU accelerated mobile robot 3D map building module. It can be used with any robot platform with three requirements: an RGBD camera, an Inertial Measurement Unit (IMU), and a GPU. G3DOMap builds 3D occupancy grid maps with high accuracy in real-time. The maps are stored in hash tables which allow fast access to octree nodes. The octree nodes are represented as Morton codes. Voxel occupancy is updated linearly with upper and lower bounds, leading to fast identification of dynamic obstacles in the map.

(2) An autonomous multi-robot 3D mapping system prototype– A mobile robot system runs on Robot Operating System (ROS) with the following features; (i) Efficient and accurate building of 3D occupancy grid maps of an unknown and dynamic environment, using G3DOMap (ii) Sharing the maps among robots in a distributed communication network, (iii) Merging of local maps into a global map with known robot poses, and (iv) Autonomous exploration of frontiers in even-surfaced environments.

1.9 Structure of thesis

This thesis consists of nine chapters including information about overall system design and sub-module development.

Chapter 2 discusses related work in the areas of 3D mapping, multi-robot coordination, inter-robot communication, and autonomous exploration. The base prototype is evaluated against findings from the literature review.

Chapter 3 proposes the research methodology after comparing different techniques discovered in the literature survey for 3D mapping, multi-robot systems, and autonomous exploration. The section also rejects or accepts the techniques used in the

base prototype.

Chapter 4 discusses the abstract system design including overall system architecture and system data flow. It also introduces hardware and software parameters used in development and testing.

Chapter 5 presents G3DOMap, the novel GPU-based 3D occupancy grid mapping system. Its system architecture and the implementation details are discussed comprehensively. The GPU pipeline consists of ray tracing, point transformation, M-code generation, RGBO-code generation, and voxel creation. Point cloud data preparation and map update belong to the CPU pipeline. Morton coded octree is compared against the traditional octree's time performance, and the overall map registration process is compared with OctoMap. Quantitative and qualitative results of G3DOMap are presented side by side with the performance of OctoMap.

Chapter 6 addresses the 3D map merging process of the system. The odometry and initial pose-dependent, direct map merging method used in the system is discussed. Then it addresses the errors and inconsistencies that occur in the maps due to the procedure.

Chapter 7 discusses how multimaster-fkie ROS package is integrated into the system.

Chapter 8 discusses how explore-lite ROS package is integrated into the system.

Chapter 9 concludes the thesis and discusses future work.

2 LITERATURE REVIEW

2.1 Introduction

This chapter intends to identify prominent work related to this research and provide a comprehensive review of different techniques.

The related work is investigated as four categories.

(1) 3D environmental mapping approaches for mobile robots

(2) 3D scene reconstruction approaches using high-performance computing techniques

(3) Inter-robot system coordination and communication

(4) Autonomous exploration in unknown environments

Map building techniques in the literature are found in two main application domains: environmental representations for autonomous robots and scene reconstruction for human viewers. Autonomous robots must utilize embedded mobile processors as their computational units. Therefore, their mapping schemes focus on efficient use of resources such as memory, access time, and computational power. On the other hand, scene reconstruction approaches usually have high memory and computational demand, but they have access to high-performance devices. Even though scene reconstruction approaches seem irrelevant to this work, they are studied in the scope of high-performance computing techniques.

2.2 3D environmental mapping approaches for mobile robots

Mobile robot 3D mapping techniques are found in three categories.

(1) Grid based mapping methods

(2) Point cloud mapping methods

(3) Polygonal mapping methods

2.2.1 Grid based mapping methods

Mathematical preliminaries for probabilistic occupancy grids were introduced by Elfes [8] in 1989. The approach is based on Bayesian estimation for incremental update

of tessellated environment, taken from multiple robotic sensors over multiple points of view. This approach is designed to support development of agile and robust sensor-based mapping mechanisms, incremental discovery procedures, explicit handling of the uncertainty of environment models, and multi-sensor information integration. This work has been widely used by researchers throughout history, applying it for various occupancy grid map techniques.

High memory consumption of grid structures is the main challenge in 3D occupancy mapping. Even though the memory footprint of a 3D grid map is less than that of a point cloud map, it can introduce a memory overhead to a mobile robot system if the map is not stored efficiently. Ryde et al. [9] propose *multi-resolution occupied voxel lists* as a solution to the above challenge. The hash table data structure only holds the occupied voxels, and the map is represented as a 3D probability density function. Each entry in the voxel list contains the integer position of a voxel and an index to the number of times (integer) the given voxel has been observed occupied. If the occupancy falls below zero, the voxel is removed from the list. The voxel update is done in an additive manner using log odds.

Dryanovski et al. [10] presents *multi-volume 3D occupancy grid maps* (MVOG maps), which stores both occupied and free space. The map is a 2D grid lying in the X plane where each cell contains two lists of volumes: positive and negative. Negative (free) and positive (occupied) observations are grouped into vertical volumes to save space. Updating the map is a three-step process: rasterizing scan readings to obtain the cells they cross, creating and inserting new volumes into volume lists, and verifying their mathematical constraints. The probabilistic value of a cell can be obtained based on its volume and density. The authors have overcome *map overconfidence*[1] issue by assigning a higher weight to newer readings.

When large-scale 3D occupancy grid maps are used in conventional SLAM systems, path planning and loop closure can be computationally inefficient. Schmucket al. [11] propose a solution to the above problems by *using hybrid metric-topological maps*.

[1] a condition occurs in probabilistic cell update methods, in which the same number of negative observations are needed as the number of positive observations to completely remove a dynamic object from a cell and vice versa.

Their key idea is splitting the global map into multiple occupancy grid submaps located within a global coordinate frame. In this system, the corrected keyframe poses only affect the relevant submap. Hence the computational complexity and the error propagation is minimal.

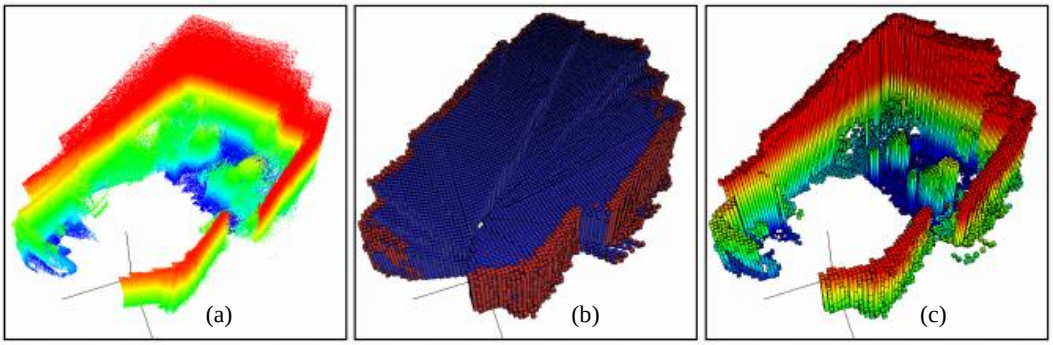


Figure 2.1: Three Stages of 3D MVOG map generation. (a) Raw point cloud, (b) Positive volumes in red and negative volumes in blue, (c) The map showing obstacle volumes; Source: [10]

Developing computational and memory-efficient tree algorithms for 3D occupancy grids is a highly researched area in the literature. *Octree* is a very suitable tree structure for 3D space which recursively divides 3D cartesian space into eight children. The efficiency of the octree depends on its design for storage, node access, update, and compact methods. Therefore, octree implementation research is more platform-independent than being application-specific.

A computation and memory efficient octree data structure, FastOctTree was proposed by Poppinga et al. [12]. The main design feature of FastOctTree is the distinction between its intermediate nodes and leaf nodes. A node contains only one word member, a union which used as a pointer to point to a struct containing all its children. If the node is a leaf node, the union is used as a bitmap to show the occupancy value. Based on the sibling nodes' occupancy, the nodes are collapsed for space efficiency. As map-related algorithms like SLAM and map merging require the calculation of nearest neighbors, the authors introduce a fast nearest neighbor search algorithm by exploiting the hierarchical structure of octrees.

OctoMap probabilistic 3D occupancy grid map is developed by Hornung et al. [7],

taking the underlying uncertainty of scans due to dynamic objects and reflected scans into account. OctoMap represents all three states (occupied, free and unknown) of the environment as one of its unique features. The map building process contains all the traditional methods such as ray tracing, probabilistic voxel update using the Elfes equation, and tree pruning. In addition to above operations, OctoMap explicitly model unknown space using uninitialized tree nodes. The octree is limited to a fixed maximum depth of 16, which makes the random node lookup complexity $O(16)$. While the maximum depth is user defined, a 16-level deep tree can cover a (655.36 m)^3 cubic space at 1 cm resolution. Instead of using Moravec and Elfes [13] probabilistic equation in pure form, the authors use the log odds form to reduce computational cost. Setting upper and lower bounds for log odds value preserves the map confidence. These clamping threshold [14] values are used to represent that a node is stabilized over time, and only then the node is pruned for lossless compression. The authors have made the system memory efficient by storing one pointer per node instead of eight to represent child nodes. The single pointer then points to an array of eight pointers. Leaf nodes, which are 80-85% of all nodes do not carry further pointers saving 60-65% memory from traditional methods. The framework publishes binary octrees, color octrees and 2D costmaps for navigation.

Quadtree is the spatial equivalence to octrees in 2D space. Li and Ruichek [15] present the *variable resolution occupancy grid mapping* technique for 2D space. They use Morton codes to solve the problem of *efficient addressing of tree nodes*. Morton

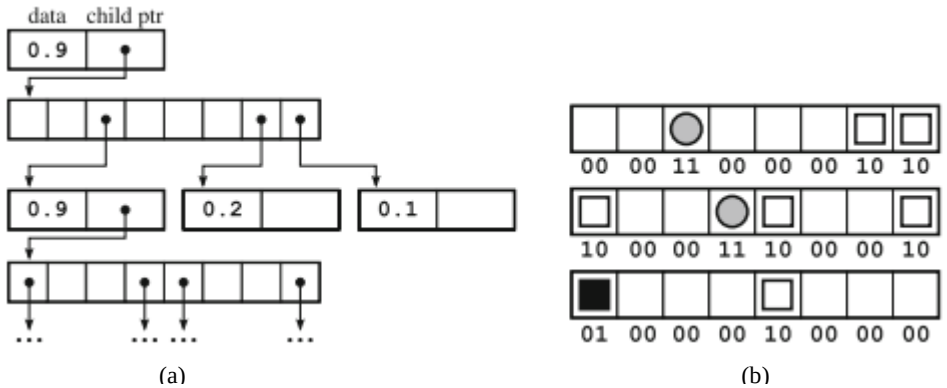


Figure 2.2: Octree memory structure (a) Octree node memory connected by pointers. Actual data is stored as one float to denote occupancy value (b) Tree stored as a serialized bit-stream; Source: [7]

codes map multi-dimensional data into one dimension while preserving locality between nodes. The system adjusts to variable resolution by splitting and merging nodes at different depth levels.

Saarinen et al. [16] present a hybrid 3D mapping method by combining Normal Distribution Transform (NDT) maps and occupancy grid maps. The hybrid scheme is used to achieve compactness, accuracy, robustness against dynamic changes, real-time performance, and long-time operations. They utilize the compactness of NDT maps and robustness of occupancy maps. The authors have first divided the 3D space into grids, then applied 3D NDT for each cell. Since NDT uses nine parameters to describe space, it can provide more accurate information than only using the occupancy value.

Occupancy grids always divide the space into perfect rectangular/cubical tessellations. This leads to surface deformation to some degree and affects vital operations like obstacle avoidance. AtomMap [17] is a 3D probabilistic occupancy map representation which indicates voxels as an unordered collection of non-overlapping, equally sized spheres. Its underlying data structure is arbitrary depth kd-trees, and the map supports more accurate surface reconstructions and path planning algorithms like A*.

2.2.2 Point cloud map methods

A point cloud is a set of data points in space [18]. A point cloud is commonly represented as an array for the convenience of transmitting between machines and serial data processing.

Point cloud maps are environmental models represented in their original form. There, incremental point cloud scans are registered into one coordinate frame and stored as a single map. Since original point clouds are dense, most approaches downsample the point cloud to a fixed resolution before generating maps. Several operations can be performed on point cloud maps, such as point cloud merging, loop closure in SLAM, plane and edge detection, segmentation, and model reconstruction. Point cloud maps can be easily converted into other map types.

Nucher et al. [19] present a 6D SLAM approach with point cloud based volumetric 3D mapping. Their target is to map abandoned underground mines to identify unstable

segments. The registration scheme matches the entire point set rather than a single 2D plane. This results in high precision and reduces local minima in the Iterative closest Point (ICP) algorithm. The ICP-based registration algorithm is called *simultaneous matching.* The authors claim two scans are overlapping if more than 250 correspondence point pairs exist. As ICP is computationally expensive, the system downsamples the input scans to 10 cm resolution using *fast filtering*. Kd-trees with terminal buckets are proposed as the map data structure to reduce access time.

Rovira-Mas [20] proposes a system for developing *3D point cloud terrain maps* for agricultural fields. They create terrain maps by combining information gathered from a stereo camera, Global Positioning System (GPS), and Inertial Measurement Unit (IMU). The stereo camera is used to create point clouds with six different parameters per point: east coordinate, north coordinate, altitude, red component, green component and, blue component. The stereo images are registered into a global map with the aid of IMU and GPS readings. The system handles two local mapping issues: scarce matching and unfiltered mismatching. Map uncertainty is resolved by using the *validity box* concept. The system seems to be more suitable for a static environment. The authors use *3D density grids* for navigation purposes. The density grids store the number of stereo scans per unit volume in a grid map.

Valencia et al. [21] present a 3D mapping system for SLAM in urban areas. Their mapping environment is over 15,000 square meters and includes challenging features such as uneven-surfaced environment, underpasses, poor GPS coverage, moderate vegetation, points with aliasing, and shadows. Since the original ICP [22] is sensitive to noise levels, occlusion, and complexity of range data, the authors have used a modified ICP algorithm for point set registration in map building. They use a hierarchical correspondence search using the point-to-plane strategy at coarser levels and the point-to-point strategy at finer levels. The system registers point clouds with ± 5cm noise level for each laser beam. While the urban map is presented as a point cloud map, the navigation is supported by a 2D grid map derived from the original map.

2.2.3 Polygonal map methods

Geometrical mapping techniques preserve the mapping objects' actual shapes as much as possible, rather than using a tessellated representation.

Huber et al. [23] introduce a high-resolution *triangular mesh* technique to model large terrain information gathered from range sensors. The initial triangular meshes are formed by connecting nearest neighbor points. Their goal is to solve the point cloud registration problem without knowing the transformation between point sets and without using surface segmentation or feature extraction methods. It has been achieved by using *spin-image,* a 2D signature for describing the local shape of a free-form surface at a point. If two points have similar spin-images, they have similar local shapes. As the number of input point sets is large, the authors have used sparse octrees with fast hashing to access them.

Thrun et al. [24] address the problem of creating compact 3D maps of large cyclic environments using 2D laser range finders. A mobile robot with two laser scanners is used to build 3D models. The robotic system connects nearby laser measurements into smaller polygons to generate *multi-polygon surface models*. When the number of measurements is large, the maps become overly complex. This problem is solved by fusing similar polygons. The noise reduction is achieved by filtering the outliers based on scan distances.

Polygonal city mapping by combining façade models and airborne models is proposed by Furh and Zakhor [25]. They use DSM (Digital Surface model) for the initial airborne map and MCL (Monte Carlo Localization) to localize the façade maps into the ground map. The efficiency of DSM's input scan processing is increased by resampling the points into a grid-based row-column structure and using the average z coordinate of the cell as the altitude. After the pre-processing, the DSMs are converted into triangular meshes. To create the digital terrain model, the authors identify the façade locations from the DSM where height discontinuities are detected. MCL is used for localization of the ground cameras and fusing the two models. Howard et al. [26] discuss a similar system for aerial map-based localization.

Wolf et al. [27] present a very compact, memory-efficient urban mapping system

by extracting planer details from point cloud maps. An urban area often consists of buildings with flat walls. Hence, they can be modeled as planar surfaces by providing

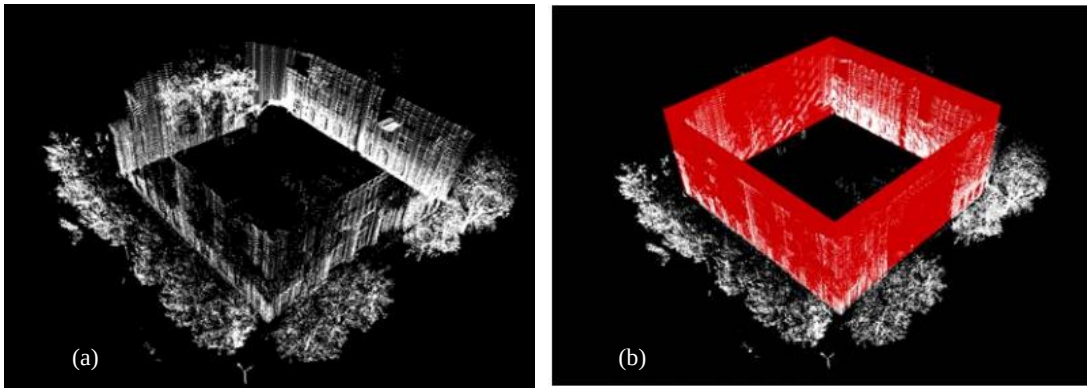


Figure 2.3: Point cloud (a) to planer representation(b); Source: [27]

enough details for path planning and visualization. Their algorithm has three steps: (1) generating point cloud maps from odometry, IMU, GPS, and MCL (2) extracting planes from point cloud map using 3D Hough transformation (3) associating planes for geometrically representing the buildings. The algorithm can predict the full model of a building based on the partial information available in the point cloud. Non-planar objects are represented as point clouds.

Xiao et al. [28] presents another *planar based* 3D mapping method. They first identify planer segments in the point cloud using a Cached Octree region Growing algorithm with *K nearest neighbor search*. Then the identified planer segments are divided into small geometrical faces like triangles and quadrilaterals, and the total area of the original planar segment is calculated. Then a heuristic transformation algorithm is used to find the transformation matrix between point clouds. Two and three non-parallel plane pairs are correspondingly enough to calculate the rotation matrix and the translation vector. The authors state that over 90% compression ratio can be achieved by representing point clouds as small planar segments.

2.2.4 Summary

The Table 2.1 summarizes the related work discussed in this section.

The following summary is based on the comprehensive literature survey of the related work.

- Point cloud maps are straightforward and suitable for full environmental modeling. However, its memory consumption is high due to a large number of points on a map. Dense point cloud maps are more suitable for surface reconstruction applications than for mobile robotics. If point cloud maps are used for robot navigation, they should be sampled down to reduce computation complexity [25], [19]. Since this is the most primitive form of a map, point cloud maps can be converted into other map types.

- Polygonal maps are a more accurate representation of the world because they generate mathematical models of surface information. It is the least memory-consuming map type compared to the other two map types. As an example, for representing a large vertical wall, a point cloud map may take thousands of points, an occupancy grid map may consume hundreds of voxels, and a polygonal surface map may describe the vertical plane in a single equation [27].

- The mathematical complexity of occupancy grid maps lies between point cloud maps and polygonal maps, where the point could map being the least complex. Discrete level environment representation makes it the most machine-friendly technique. Therefore, occupancy grid maps are the most popular 3D mapping method in mobile robotics. Due to the voxel effect, occupancy grid maps can be the least accurate surface representation. But in terms of path planning, all calculations can be directly done on map cells. Occupancy grid maps can reduce their resolutions and reduce computation complexity.

- Representing dynamic environments is more complex than representing static environments. Probabilistic sensor data fusion is the technique to resolve the dynamic environment problem. This technique is mostly used in occupancy grid maps due to their atomicity. The probabilistic methods help to fuse multiple viewpoints and to increase the robustness.

- In the past, 3D occupancy grids were considered highly space-consuming because of their multi-dimensional array structure. Unless the underlying data structure is designed to address space and time complexity concerns,

occupancy grids can end up being the bottleneck of a mobile robotic system. Constant depth kd-tree approaches have resolved the time and space problem. Today, octree data structures are used in traditional CPU-based occupancy grid mapping systems with the ability to compress, adjust resolution, fast update, and spatial locality.

- A few works pointed out that occupancy grid maps are not a proper approach for large-scale, long-running systems [24], [16], [11]. In such cases, those systems have been benefitted from hybrid approaches. However, they have not abandoned the advantages gained by space discretization by occupancy grid maps.

2.3 3D scene reconstruction approaches using high-performance computing techniques

The above section explored the prominent literature in 3D map generation using mobile robots. This section is dedicated to the 3D mapping work done using modern high-performance computing techniques such as parallel threads and GPU methods.

2.3.1 Prominent work

Zhou et al. [29] present the first data-parallel surface reconstruction algorithm that runs entirely on GPU. They put incoming point cloud data into an octree using level order traversal. All octree nodes at the same tree level are processed in parallel, spawning a new thread for every node at the same level. For searching data pointers of relative nodes, precomputed Look Up Tables (LUTs) are used. A maximum tree depth of 10 is used while an octree node holds 62 pointers to entities: parent node, 8 child nodes, 27 neighboring nodes, 8 vertices, 12 edges, and 6 faces. Given a set of 500K points, the overall GPU algorithm has achieved a frame rate of 5.

Pirker et al. [30] propose a GPU-based 3D occupancy grid map building system for large indoor environments using a depth image pyramid method. Since voxels are independent random variables, the authors use an inverse modeling algorithm to process them parallelly. The mentioned inverse sensor model depends on distance measurements instead of depth image readings. Therefore, image pyramids are generated using the Euclidean distance between the voxel centroid and the camera

center. A weighted interpolation scheme is proposed to calculate voxel occupancy. The authors have achieved a framerate of 14 at 8 cm resolution.

Table 2.1: Mobile robot 3D mapping techniques related work summary

System	Application	Map type	Map data structure	Special features
Elfes[2] [8]	Navigation and mapping	Probabilistic 2D occupancy grids	Grid arrays	Introduction of probabilistic mapping
Huber et al. [23]	High resolution large terrain mapping	3D triangular mesh	Octrees with fast hashing	2D Spin image point descriptor
Fruh et al. [25]	Urban modeling	Polygonal map		DSM of aerial view combined with façade maps
Nycher et al. [19]	Mapping abandoned mines	3D point cloud	kd-tree with terminal buckets	Down sample the point cloud using fast filtering
Wolf et al. [27]	Urban building mapping	Geometric planer maps		Predicts partially observed model information
Mas et al. [20]	Terrain mapping for Precision agriculture	Point cloud terrain maps		Use 3D density grids for navigation
Valencia et al. [21]	SLAM for urban service robots	3D point cloud maps		point-to-point ICP and point-to-plane ICP for registration
Xiao at al. [28]	Outdoor mapping	3D Planar map	Octree	Planar surface area is used to find correspondences
Ryde et al. [9]	Navigation and mapping	Multi resolution voxel lists	Voxel lists in hash tables	Occupied voxels are stored in a compact form
Li et al. [15]	Intelligent vehicles	Probabilistic 2D occupancy grids	Quadtrees as Morton code	Multi-resolution maps, M-codes
Dryanovsky et al. [10]	Indoor Micro aerial vehicles (MAVs)	Multi volume occupancy grid maps	Voxel lists	Vertical volumes are used to indicate the occupancy of a 2D plane.
Poppinga et al. [12]	Octree optimization	3D occupancy grid	Octree	Nearest neighbor search, pointer based inner nodes
Hornung et al. [7]	Probabilistic octrees	3D occupancy grid	Octree	Lossless compression, represents unexplored area

[2] This work is not a 3D mapping technique. But it introduces the concept of probabilistic occupancy grid mapping

Saarinen et al. [16]	Large scale, long-term industrial mapping	3D NDT-Occupancy maps	Grids	Combination of NDT and occupancy grid maps
Schmuck et al. [11]	Large-scale mapping	Hybrid metric-topological 3D occupancy	Octrees	Map consists of small submaps
Fridovich-Keil et al. [17]	Mobile robot navigation using maps	AtomMap	Arbitrary depth kd-trees	Voxels are represented with spheres/atoms

GigaVoxels [31] is a voxel-based video rendering pipeline for generating . It uses several GPU-based methods: sparse hierarchical octrees with bricks for compact storage, efficient access and update of voxels, rendering algorithm based on ray casting and voxel cone tracing, and virtual memory scheme for caching and on-demand streaming. Each non-empty node in the octree is attached with a 3D brick, a low-resolution gird, which approximates the original volume of the octree node. Bricks improve hardware interpolation and cache coherence. The nodes and bricks are stored in separate memory spaces in GPU named node pool and brick pool.

Zeng et al. [32] propose a sparse octree-based surface reconstruction method to enhance the space consumption of the KinectFusion [33] framework. While KinectFusion stores its Signed Distance Function (SDF)-based reconstructed scene is a uniform 3D grid of voxels, this work has got rid of all empty voxels. The octree is divided into four layers: top, branch, middle, and Data layer. There are pre-allocated node buffers in GPU memory for each layer. SDF data is stored in data layer nodes. All system data including the octree, captured depth maps, predicted surfaces, and auxiliary arrays are stored in the global memory. The experiments show that the system consumes 10% less memory compared to KinectFusion.

Nießner et al. [34] reconstruct surface data in real-time and at scale using *voxel hashing*. Their work is based on the earlier Truncated-SDF (TSDF) and grid-based method introduced by Curless et al. [35], where most of the voxels in the grid are free or unobserved. Voxel hashing is a non-hierarchical data scheme that efficiently handles the sparsity of TSDF representation. The system fuses new TSDF samples into the hash table with minimum collisions using buckets and appended linked lists. Race conditions are resolved using locks on hash buckets, unwanted voxels are garbage collected, and the voxel blocks are bidirectionally streamed between CPU and GPU. *Voxel blocks* are not stored spatially, but the hashing function allows efficient

lookup for neighbors using *integer rounded* world coordinates.

Kleinschmidth et al. [36] present a 3D multi-channel point cloud approach to aid search and rescue operations via a virtual reality interface. The interface merges 2D data such as RBG, thermal, active infrared, and 3D data of time-of-flight depth sensors. 2D to 3D projection is the mathematical basis of their work. The number of memory exchanges between device and host is minimized by transferring calibration details to GPU memory and allocating space for static information before runtime. The system has 80% performance acceleration compared to CPU-based implementation and has achieved a framerate of 35.6.

BundleFusion [37] is a real-time end-to-end 3D reconstruction framework that makes 3D scanning easily accessible to untrained users. The main goal is to develop an efficient global pose optimization algorithm. The researchers have used two GPUs together: Nvidia GeForce GTX Titan X and GTX Titan Black. They have been used for volumetric reconstruction and correspondence search, and global pose optimization respectively. The parallelized processes handle loop closure, tracking failure recovery, and reduction of local geometric drift. The approach runs 20 times faster than similar CPU based method Ceres. [38]

An incremental triangle mesh map generation technique using *spatial hashing* is proposed by Doi et al. [39]. They have developed a parallel mesh generation pipeline for online mesh extraction, update, garbage collection and volumetric data fusion. The system input is a stream of depth images from a handheld RBGD sensor. The IMU data are assumed to be accurate. The data is stored as a cascading hash table where space information is divided into large blocks and cubes. The authors use a Computed Unified Device Architecture (CUDA) implementation. In there, blocks and cubes are handled by stream processors and threads respectively. One corner and three edges per cube are registered by pointers since the other corners and edges can be accessed through adjacent cubes. The corner represents the TSDF value and vertices represent the triangles in the surface mesh. They are both stored in pointer arrays and data is managed in the memory heap. Vertices with no references are garbage collected. The authors have also addressed memory leak conditions.

Pose drift and non-real-time registration are two of the shortcomings of large-scale 3D reconstruction pipelines. Taking those into account, Dong et al. [40] propose a robust and offline 3D reconstruction method using GPU acceleration for RGBD odometry, geometric feature extraction and matching, point cloud registration,

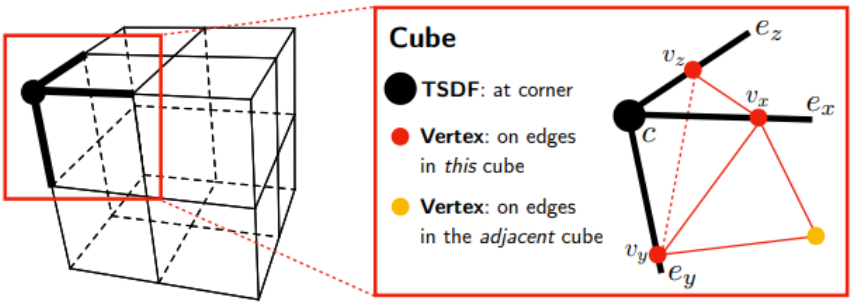


Figure 2.4: Cube level data structure. Only one corner and three edges maintained per cube; Source: [39]

volumetric integration, and mesh extraction. The proposed method contains four main stages. *Submap creation* step estimates frame-to-frame camera pose for even subsets of the input sequence. *The submap registration* step uses pairwise submap registration. Colored ICP is used for adjacent submap pairs while geometric feature-based FGR is used for distant submaps for loop closure detection. *Refine registration* step is done again using colored ICP. *The scene integration* step fuses the entire RGBD sequence using TSDF integration. As GPU-based data structures, a hash table and linked lists are used, where each hashed key corresponds to a fixed size bucket array, and linked lists are used for avoiding collisions. The marching cube method is used to extract surface data from final TSDF volume maps. This process is similar to the first author's previous work [40]. The authors have tested the system on an Nvidia 1070 graphics card as well as on NVIDIA Jetson TX2 mobile hardware.

The next section summarizes the findings of prominent work in GPU based scene reconstruction techniques.

2.3.2 Summary

A summary of major related work in GPU accelerated scene reconstruction is presented in Table 2.2.

Almost all GPU accelerated 3D mapping systems are scene or surface reconstruction approaches. Therefore, they ignore the free and unknown environment, which makes it easier to build sparse data structures with fast access and update rates. Also, most of the above systems are based on the handheld mapping. We can see that a hash table has been a better choice as a data structure to store sparse maps in GPU.Unlike scene reconstruction approaches, autonomous mobile robot systems rely on free and unknown space representation. In comparison, such cases require a massive number of voxels to be registered on the map.

The incorporation of GPU acceleration into autonomous mobile robot 3D mapping systems with suitable data structures will minimize the latency of map building. But we need to keep in mind that both occupied and free space need to be mapped, so the map registering latency values are expected to be lower than scene reconstruction approaches.

Table 2.2: GPU-accelerated scene reconstruction related work summary

System	Application domain	Map type	Map data structure	Special features
Zhou et al. [29]	Surface reconstruction	Triangular mesh	Octree	Precomputed LUTs for parent and child nodes
Pirker et al. [30]	Environment reconstruction	3D occupancy grid map		Image pyramid generation
Crassin et al. [31]	Large-scale video scene reconstruction	Voxel grid	Octree based voxel pyramid	Combine occupancy grids and pyramids
Zeng et al. [32]	Realtime dense 3D reconstruction	SDF map	Octree	GPU utilized version of KinectFusion
Niebner et al. [34]	Real time dense 3D reconstruction	3D surface map	Hash tables with voxel blocks	Occupied SDF information stored inside voxel blocks
Kahler et al. [41]	Dense SLAM	TSDF map	Hash tables with voxel blocks	Extension of [124]

Kleinschmidth et al. [36]	Virtual reality for SAR operations	Multi-channel point cloud		Merge multiple image types: RGB, thermal, depth
Dai et al. [37]	Real time 3D reconstruction	Mesh geometry models		Globally consistent pose estimation in real-time
Dong et al. [39]	Dense real-time reconstruction	Triangle mesh maps with TSDF	Hash table with blocks and cubes	Surface and TSDF are stored as cube vertices and corners
Dong et al. [40]	Dense real-time reconstruction	TSDF volume maps, triangle mesh maps	Hash table with buckets and linked lists	Colored ICP is used for registration, loop closure and refinement steps

2.4 Inter-robot system coordination and communication

In this section the robot coordination mechanisms are labeled as three types

(1) Centralized coordination

(2) Decentralized coordination

(3) Hybrid coordination

Inter-robot communication mechanisms are classified into three categories.

(1) Continuous connectivity

(2) Event triggered connectivity

(3) No connectivity

A summary of major work relating to inter-robot communication and coordination s shown in Table 2.3.

Table 2.3: Inter-robot system coordination and communication related work summary

System	**Communication**			**Inter-robot Coordination**			**Special features**
	No connectivity (passive)	Continuous connectivity	Event-triggered connectivity	Centralized	Decentralized	Hybrid	
Wang [42]	✓				✓		Sign board
Yamauchi [43]		✓			✓		Broadcast
Alami et al. [44]			✓	✓			Broadcast, one-to-one
Simmons et al. [45]		✓				✓	Bid based tasks
Gerkey et al. [46]			✓		✓		Pub-sub, fitness task negotiation

Mosteo et al. [47]		✓					MANET
Doriya et al. [48]		✓				✓	Cloud based
Stephan et al. [49]		✓				✓	Ad-hoc
Cesare et al. [50]			✓		✓		Battery life
Kantaros et al. [51]		✓			✓		Electing leaders
Sinay et al. [52]		✓					Tree coverage
Kemna et al. [53]			✓		✓		Underwater
Banfi [54]				✓			Reconnect plans

After going through above related work, the following important information was gathered about inter-robot system coordination and communication.

- Decentralized coordination can achieve robustness and fault tolerance of the system. But to ensure adaptability in a fully decentralized system, homogeneous robots should be used. Each robot in the system must calculate its decisions, which leads to high computational load and repeated calculations. If data sharing mechanism is poor, multiple robots end up exploring the same area [43].

- Centralized coordination requires complete connectivity to the system while it can lead the system towards a single point of failure. But when creating exploration plans and assigning robot goals, centralized systems have reduced complexities compared to decentralized systems. A single node makes all system decisions, and the rest of the robots handle their navigation plans.

- Hybrid approaches try to incorporate the advantages of both centralized and decentralized mechanisms. The most common hybrid systems are cluster-based. While the cluster leader takes centralized decisions about connectivity and plan execution of the local team, the clusters act as individual distributed units. Heterogeneous robots can be used in this approach.

- Continuous connectivity communication methods can be implemented for known environments. Here the known environment means we already know its extent and signal transmission quality. In such scenarios, robot teams are

designed to explore within the team's communication range while exploring and acting as relays. But most of the systems do not focus on a coordination plan after a relay or access point (AP) failure.

- Event-based communication is mostly designed for inter-robot systems operating in unknown environments with communication constraints. The robots have a rendezvous plan to meet at some predefined point, or after a predefined time. In some systems, the robots use communication maps [55] to travel to a point with a promised link and bandwidth after disconnection.

2.5 Autonomous exploration

Autonomous exploration is another crucial element in the proposed system. Within the discussed application scenario, the goal of the exploration sub-system is to achieve maximum coverage in minimum operational time. The prominent literature studied is classified into four areas.

(1) Frontier based exploration
(2) RRT (Rapidly exploring Random Tree) approaches for frontier detection
(3) NBV (Next Best View) exploration
(4) Hybrid approaches

The related work in autonomous exploration field is summarized in Table 2.4.

Table 2.4: Autonomous robot exploration related work summary

Work	Robot type	Map type	Exploration strategy	Special features
Yamauchi [56]	Wheeled robot	2D evidence grids	Frontier based	Introduction of frontier-based exploration
González-Baños et al. [57]	Wheeled robot	2D Polygonal map	NBV	Handles odometry errors and uncharted obstacles
Surmann et al. [58]	Wheeled robot	3D Metric feature map	NBV + art gallery problem	Greedy extension of art gallery problem
Zhu et al. [59]	Aerial robot (MAVs)	3D occupancy grid	Frontier based	Incremental extraction of frontier cells
Bircher et al. [60]	Aerial robot (MAVs)	3D occupancy grid map	RRT	Sample based receding horizon path planner

Senarathne et al. [61]	Wheeled robot	3D occupancy grid map	Surface frontier based NBV	Expanding mapped surfaces at frontiers
Papachristos et al. [62]	Aerial robot	3D occupancy grid map	RRT	Receding horizon belief-based approach
Umari et al. [63]	Wheeled robot	2D Occupancy grid map	Multiple RRT	Local + global RRT for speed and full coverage
Dai et al. [64]	Aerial robot (MAV)	3D occupancy grid map	Frontier + sampling	Morton code supports frontier cell grouping
Selin et al. [65]	Aerial robot	3D occupancy grid map	NBV + RRT + frontiers	Local + global exploration using NBV and frontiers

After analyzing the prominent exploration methods used in the literature, the following information was gathered.

- Frontier-based exploration and NBV exploration are the most used exploration strategies. RRT algorithms are used for frontier identification and NBV identification. But it is mostly used with frontier-based exploration.
- Frontier-based exploration approaches perform the best in large environments. But they are unable to achieve full exploration by covering all small map segments. NBV approaches get stuck in large environments, but they are ideal for exploring small areas. Hybrid approaches have achieved the expected time and computational complexity, and full coverage.
- RRT techniques are popular because they can efficiently search paths in the entire free cell space. It is biased towards the free space and provides completeness.

2.6 Literature evaluation of the base prototype

As an existing base prototype is used [1], it's the basic components are evaluated against the findings in sections 2.2-2.5.

The multi-robot system builds a 3D representation of an unknown environment without an initial map. It does not discuss multi-robot system coordination, but all other necessary sub-systems are comprehensively studied to find the best available technique. 3D mapping, 3D map merging, autonomous navigation, and inter-robot

communication network are the above-mentioned sub-system. The hardware simulation of the system is prototyped using Quanser Qbot2, Microsoft Kinect 360, and Raspberry Pi 3b module as the processing unit. Given the computational power of the single-threaded, mobile processor of the Raspberry Pi 3b, the 3D mapping system is functional, but not with the expected latency.

In the upcoming subsections, three of the essential sub systems of the base prototypes are discussed.

2.6.1 3D map building system

The 3D mapping system is designed after evaluating two ROS-compatible 3D mapping frameworks: RTABMap [66] and OctoMap [7]. RTABMap is a point cloud 3D map, and OctoMap is a 3D occupancy grid map. As observed in the literature, the authors have evaluated two popular map types. Despite the real-world-like appearance of the 3D model, RTABMap displays undesired behaviors such as the high memory footprint and high access time. The authors select OctoMap as the 3D mapping framework considering memory efficiency, navigation support, lossless compressibility, and inbuilt navigation support.

As the system runs on Raspberry-Pi 3b module, OctoMap fails to provide the real-time map building latency in high-resolution maps. Therefore, the 3D mapping system should be enhanced to accommodate the objectives of this research.

2.6.2 Inter-robot communication network

The inter-robot communication network is designed after evaluating four ROS-compatible Wireless communication paradigms: Publisher/Subscriber, Service/Client, Single-master/multi-slave, and multi-master communications. The authors have selected the single master/multi-slave system due to the high network reliability of its Transmission Control Protocol (TCP) layered architecture. Continuous connectivity is listed as a requirement of this protocol. But the authors validate this selection stating that frequent map exchanges can reduce the bandwidth demand at a given time.

The authors also point out *single point failure* and high connectivity as the disadvantages of a single master/multi-slave system. Since these drawbacks cannot be overlooked in an autonomous multi-robot system, the inter-robot communication

system must be reevaluated and enhanced for this research.

2.6.3 Autonomous navigation system

The work includes an autonomous navigation system based on Dijkstra's shortest path algorithm [66] for global planning, and teb_local_planner [67] on top of ROS navigation stack. When a manual goal is provided to a single robot, it succeeds to navigate to that goal point autonomously using the shortest path and avoiding obstacles.

The system is not capable of exploring autonomously without human involvement in goal selection. Hence, the navigation system must need further enhancements to achieve full autonomy.

2.6.4 Evaluation results

OctoMap as a 3D occupancy grid mapping technique is a suitable choice according to the literature survey presented in section 2.2. However, it was observed several practical drawbacks of OctoMap, which will be discussed in the next chapter (in section 3.2.2).

As observed in the literature survey, master-slave inter-robot communication is not suitable for this research. There are two reasons for this conclusion: the single point failure and the effect of continuous connectivity to robot distribution in a large and unknown environment. The navigation system of the base prototype lacks the autonomous exploration component. The literature review in section 2.5 can be used to select a suitable strategy for the autonomous exploration of this research.

As mentioned in section 1.5, the existing prototype should be enhanced to meet the requirements of this research, and the next chapter is used for selecting the most suitable techniques.

2.7 Summary

This chapter presented a literature review related to research areas in autonomous robot 3D mapping and exploration.

Mobile robot 3D mapping techniques were discussed in three categories: point cloud maps, grid-based maps, and polygonal maps. 3D reconstruction work was

studied to discover GPU-based data structures and algorithms used in them. Multi-robot coordination techniques were classified as decentralized, centralized, and hybrid mechanisms, while inter-robot communication was categorized as continuous connectivity methods, event-triggered mechanisms, and passive methods. Autonomous exploration was studied under four areas: frontier-based, RRT techniques, NBV exploration, and hybrid approaches.

The findings from the literature survey were used to evaluate the subsystems of the base prototype in the next chapter, above classifications will be compared against their pros and cons, and the best techniques for the application scenario will be selected.

3 METHODOLOGY

3.1 Introduction

This section discusses the selected methodology for the main subsystems in the work. The literature review from the previous chapter is incorporated to identify state-of-the-art techniques used in the research community. They are analyzed against the application requirements, and suitable techniques are incorporated into the system design.

The overall methodology including all sub processes is illustrated in Figure 3.6.

3.2 3D mapping

From sections 2.2 and 2.3 it was identified that there are two ways 3D mapping has been carried out in the literature, autonomous robot 3D mapping and 3D scene reconstruction. The key properties of the two methods are compared using Table 3.1.

Autonomous robot 3D mapping methods are used in many application domains such as unknown terrain and mine mapping, urban modeling, and navigation. Scene reconstruction approaches are mostly used in static environments and surface modeling applications. This thesis is closer to the application domain of autonomous robot 3D mapping. Scene reconstruction methods only focus on mapping static environments. The exploration algorithms seek unexplored segments of maps; hence all three map states (occupied, free, unknown) are required to be included, which is a property only achieved by robotic 3D mapping strategies. It is observed that both mapping techniques use a variety of data structures. However, selecting a suitable data structure should be done after finalizing the 3D map type.

Based on the application domain, map states and possible environmental models, **autonomous robot 3D mapping techniques** are selected as the mapping method.

After analyzing the map building speeds of GPU-based scene reconstruction methods and high latency of CPU-based 3D map generation techniques, it is decided to incorporate GPU techniques with the map building process. Since the system requires a CPU for running hardware drivers and other conventional middleware, a **CPU and GPU mixed approach** is proposed.

Table 3.1: A summary of techniques used in autonomous robot 3D mapping and scene reconstruction

Criteria	Autonomous robot 3D mapping	Scene reconstruction
Application domain	Long range terrain mapping [23], [68], [16], [28], [11] Urban modeling [25], [27], [21] Mine mapping [19] Intelligent robot navigation [8], [9], [10], [15], [17]	Surface modeling [29], [37], [39], [40] Environmental reconstruction [30], [31], [32], [34], [41] Virtual reality [36]
Map states	Occupied [23], [24], [25], [19], [27], [68], [20], [21], [9] free + occupied [10], [12], [17] free + occupied + unknown [8], [16], [15], [7], [11]	Occupied [29], [37], [32], [34], [36], [37], [39], [40] Free + occupied [30]
Environment model	Static Dynamic [7], [8], [15], [16]	Static [40], [39], [37], [41], [69], [34]
Data structures	Multi-dimensional grids [8], [12], [16] Hash table [9] Trees [28], [7], [11], [17] Trees + hashing [23], [19] Trees + Morton codes [15] Lists [70]	Trees [29], [31], [32] Hash table [34], [41], [39], [40] Morton coded trees [58]

3.2.1 Selecting a 3D map type

Selecting a suitable 3D map type is a crucial decision as it decides the time and space efficiency of the entire system. Three popular map types are found among autonomous robot 3D mapping techniques: point clouds [19], [20], [21], occupancy grids [8], [15], [68], [12], [7], [16], and polygonal maps [23], [24], [25], [27], [28]. They are compared in Table 3.2.

As explained earlier, representing all environment states is the main requirement which is only achieved by occupancy grid maps. Also, occupancy grid maps are the only map type that considers dynamic objects in the world. Point cloud maps and occupancy grid maps are capable of resolution control. Once downsampled, point clouds cannot be recovered to their original population. Occupancy grids designs like OctoMap [7] can change resolution up and down preserving the original population. Point cloud maps are the least complex map type since they are built with simple arrays. Occupancy

grids provide the best navigation support due to their discrete representation. Polygonal maps have the lowest memory footprint, as they represent surfaces as equations. Occupancy grid maps are highly compatible, hence allow fast exchange through wireless media. Since point clouds are the basic form of maps, they can be easily converted into other map types and support further processing. Occupancy grid maps also support a limited number of operations such as exploration and path planning.

Table 3.2: Comparison of 3D map types

Properties	**Point cloud map**	**Occupancy grid maps**	**Polygonal maps**
State Representation	Occupied points	Occupied + free + unknown	Occupied surfaces
Dynamic objects	No	Yes	No
Resolution control	Yes (lossy)	Yes (lossless)	No
Complexity	Low	Medium	High
Navigation support	Low	Very high	High
Memory consumption	High	Medium	Low
Compactness	Low	High	No
Data structures	Arrays	Grids, trees, hash functions	Set of equations
Further operations	Highly supported	Supported	Less supported

Due to the following outstanding strengths, **occupancy grid maps** are selected as the environment representation.

- Simple discrete level space representation
- Ability to model occupied, free, and unknown environmental states
- Dynamic object identification
- Lossless resolution control and high compactness
- Direct navigation support
- Machine friendly
- Can link voxels to additional information

However, state-of-the-art occupancy gird maps have the weaknesses of high memory consumption for large maps. Also, the spatial octree data structures have high access time in the CPU depending on the number of depth levels. Section 3.2.2 provides proof why OctoMap, the 3D mapping technique used in the based prototype does not meet the real-time objective of this thesis.

3.2.2 OctoMap 3D map building system

There are several reasons for OctoMap's popularity in mobile robotics.

- OctoMap is an open-source C++ library which can be used with or without ROS
- Probabilistic occupancy estimations support mapping dynamic, noisy, and uncertain environments.
- Representation of free, unknown and occupied volumes
- Publish 2D cost maps for path planning
- Underlying octree data structure provides spatial information, resolution control and fast access time
- Lossless map compression mechanism

After studying the OctoMap research paper [7] and the source code, the process flow of OctoMap has been identified, and it is abstractly illustrated using Figure 3.1.

First, an RGB-Depth image frame is received from the camera sensor. ROS freenect driver or an equivalent driver converts the laser scans into a ROS readable RGB-XYZ data array. It is sent to ROS as a SensorMsgs/PointCloud2 message.

After data preprocessing, *OctoMap Server* generates the free voxels along the path of input rays.

OctoMap tree nodes are generated with (X, Y, Z) position as the node key. The node value holds extra information like RGB value, resolution at the depth level, and the occupancy of the node. These nodes are added to the octree. If a node key is already present in the octree, the node's occupancy is updated.

OctoMap server publishes the OctoMap in several formats: an RGB colored

OctoMap, a binary OctoMap, an array of voxel centers, and a 2D Ground cost map. An OctoMap reset function is also provided for the user to reset the system.

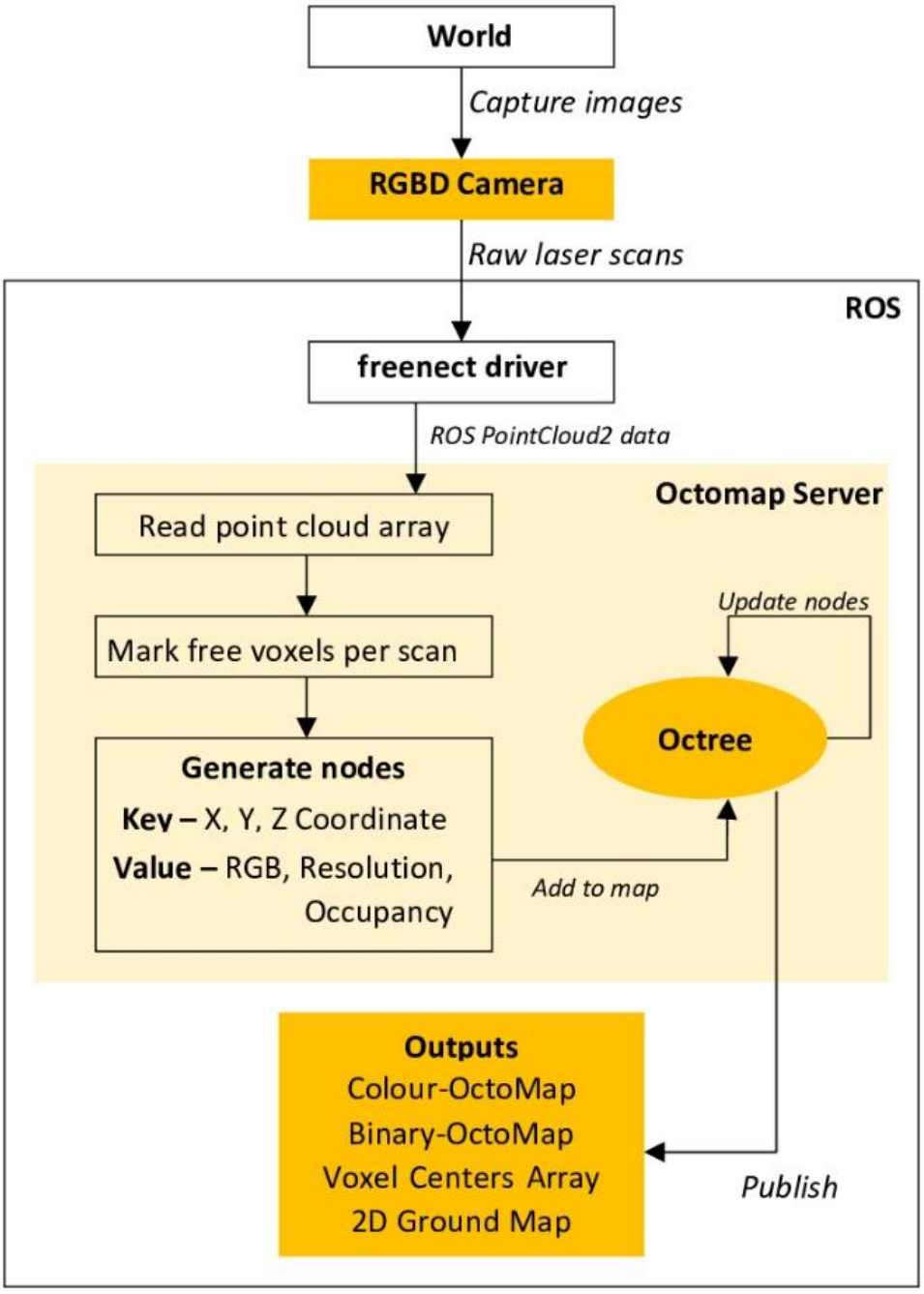


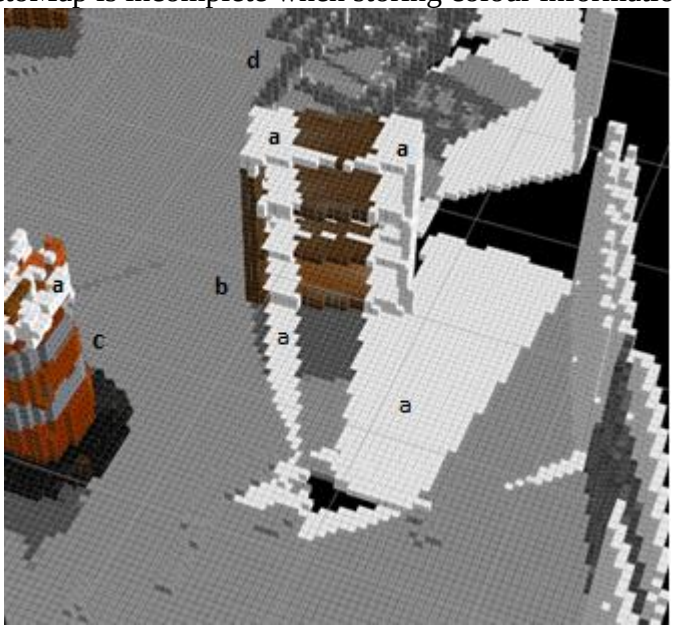
Figure 3.1: Abstract process flow of OctoMap Server

3.2.2.1 Observations

The following practical drawbacks of OctoMap were observed while using it as the 3D mapping technique of the system prototype.

- Octomap does not cope well with finer resolutions because its update time exceeds real time concerns. (Figure 3.2)

OctoMap is incomplete when storing colour information (



- Figure 3.3)
- The missing information is not recovered if not revisited by the robot
- Map update and visualization time grows with the map size.

The author has performed a time analysis within the source code to find the reasons for above observations.

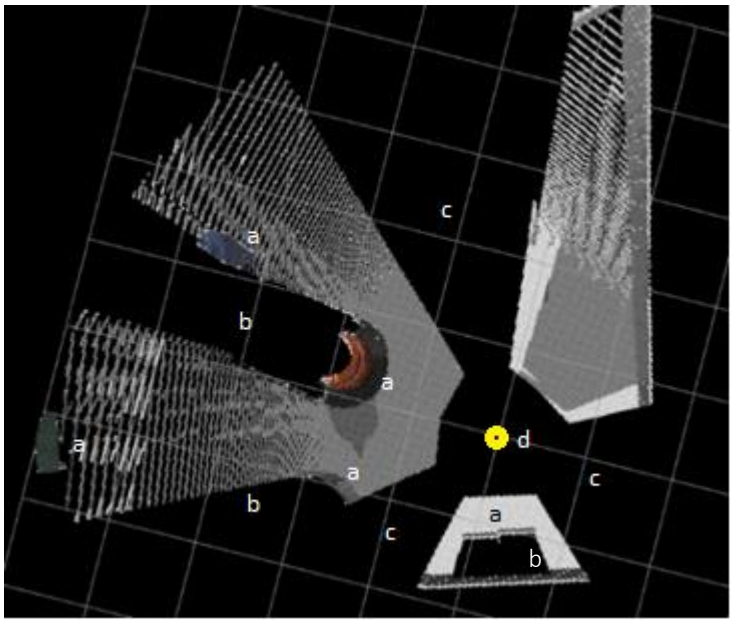


Figure 3.2: Missing scans in an OctoMap generated by a rotating robot for a simulated environment. (a) Obstacles (b) absent point cloud segments due to obstacles (c) missing scan segments (d) the robot

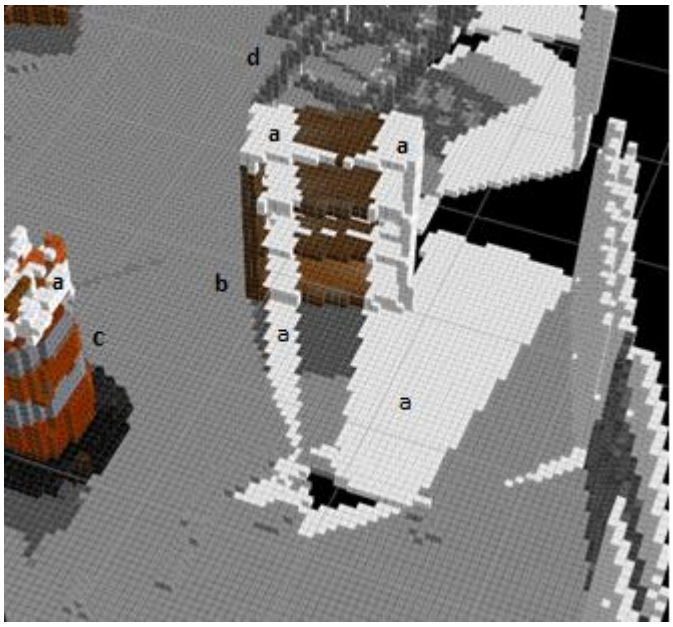


Figure 3.3: An incomplete color registration in a colored OctoMap of a simulated scene. (a) indicates incomplete-colored voxels in white. (b) bookshelf, (c) traffic cone, (d) a step ladder

3.2.2.2 Resource consumption

Table 3.3 displays the average time consumption observations of OctoMap's

processes and their sub tasks.

97.8% of OctoMap's processing time is consumed by point cloud registration since it is the main process of any map building system. The process consists of several sub-tasks: removing floor points, removing out of range points, ray tracing for generating free space voxels, generating voxel indexes, and updating occupancy log odds.

All the point cloud registration operations are performed on individual rays (points). These rays are independent of each other and not necessarily processed in the original order. As OctoMap is developed as a single thread, all the rays are processed serially. This can be considered a waste of time within multi-core processors.

When the computation load of serial processing is too high, the operating system (OS) discards the unprocessed point cloud information from the buffer. This buffer overflow is reflected by missing scan segments (Figure 3.2) and incomplete color information (

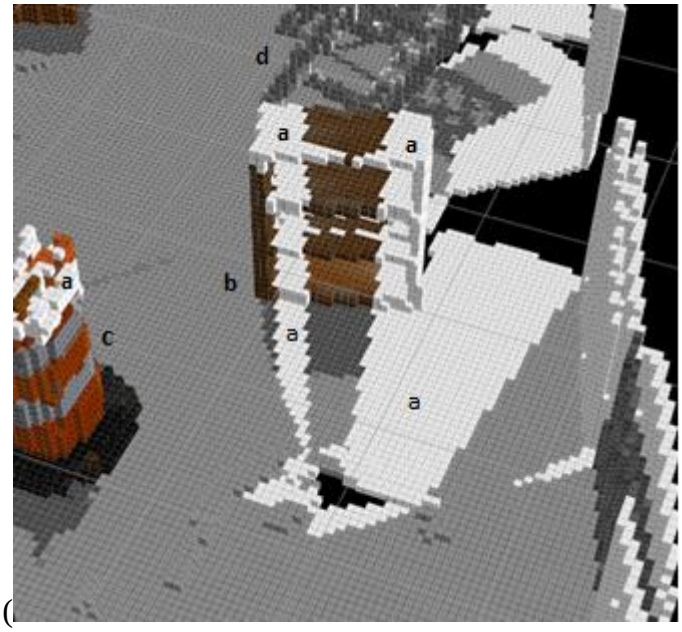


Figure 3.3) in maps. The growth of update and visualization times in large maps is caused by the growth of CPU memory dedicated for the octree.

It can be concluded that OctoMap is not a suitable solution for 3D mapping with RGBD cameras in large and unknown environments. But its process flow can be parallelized to enhance the latency of the point cloud registration step.

Table 3.3: Average time consumption by the processes in OctoMap

Process	Description and sub-tasks	Average time taken (in Clock Cycles)
Point cloud registration	(1) Remove floor, and distant points (2) Mark free space (3) Generate voxel Indexes (4) Update occupancy log odds	3,257,879 (97.8 % from the whole process)
Publish binary OctoMap	Lighter version of OctoMap	19,049
Publish full OctoMap	Complete version of OctoMap	46,488
Publish visualization array	Information array for visualization	7,708

3.2.3 Selected methodology

The primary objective of this thesis is to reduce the map building latency. Therefore, it is decided to replace hierarchical octrees with linear octrees. **Morton codes** represent hierarchical information of an octree node linearly, and they can be efficiently built using GPU techniques. Inspired from the data structures presented in Table 3.1, The Morton codes, RGB and occupancy values are saved in a **hash table**.

The summary of the selected methodology for 3D map generation process is presented in Table 3.4.

Table 3.4: Selected Methodology for 3D map generation

Feature	Selected technique	Selected technology / tool
3D map representation	3D occupancy grid maps	OctoMap as the base structure. Later replaced with G3DODMap.
Processing unit	GPU + CPU	CUDA with NVIDIA + CPU running ROS and C++
Map data structure	Octree Hash table	Unordered map in C++
Node representation	Morton code	Implement with C++

After the induvial robot maps are generated using the discussed methodology, the next process is bringing all the local maps into a unified frame. The next section

addresses the selected methodology for multi-robot coordination and inter-robot communication.

3.3 Inter-robot communication and multi-robot coordination

The inter-robot communication and coordination system should fulfil the following requirements.

- Non-continuous connectivity in the network to explore large environments using small number of robots
- The independent behavior of robots should be allowed
- Failure of one or several nodes should not fail the communication network
- Should be able to exchange files within the least time duration

In section 2.4, it was observed that there are two active communication methods used in inter-robot systems: event-triggered communication and continuous communication. In continuous communication mechanisms, the fleets are coordinated such that all robots stay within the communication range. Hence, it is a suitable mechanism for environments with known signal qualities and extents. For this work, **event-triggered communication** is proposed because the robots explore an unknown environment as independent units.

The next step is discovering a suitable mechanism for inter-robot communication and multi-robot coordination. Based on the main research objectives, it is sufficient to select a technique from existing technologies.

3.3.1 ROS compatible communication methods

As the multi-robot 3D mapping and exploration system is implemented on ROS, the inter-robot communication should also be compatible with ROS.

There are four inter-robot communication and multi-robot coordination systems compatible with ROS, which are discussed in the author's previous work [1].

(1) Publisher/Subscriber
(2) Service/Client
(3) Single master/Multi-slave (Leader following protocols)

(4) Multi-master.

The findings are summarized in Table 3.5 in a comparative format and Figure 3.4 presents their abstract architectures.

It is proposed to use a **multi-master** communication protocol based on the system requirements of event-triggered connectivity and the resilience against single point failure.

Table 3.5: A comparison of available communication methods in ROS

Communication method	**Communication direction**	**Connectivity**	**Number of ROS cores**	**Single point failure**
Publisher – Subscriber communication	Many-to-many	Continuous	One	Yes
Service – client communication	One-to-one	Continuous	N/A	Yes
Single master, multi-slave system	One-to-many	Continuous	One	Yes
Multi-master system	Many-to-many	Event-based	Many	No

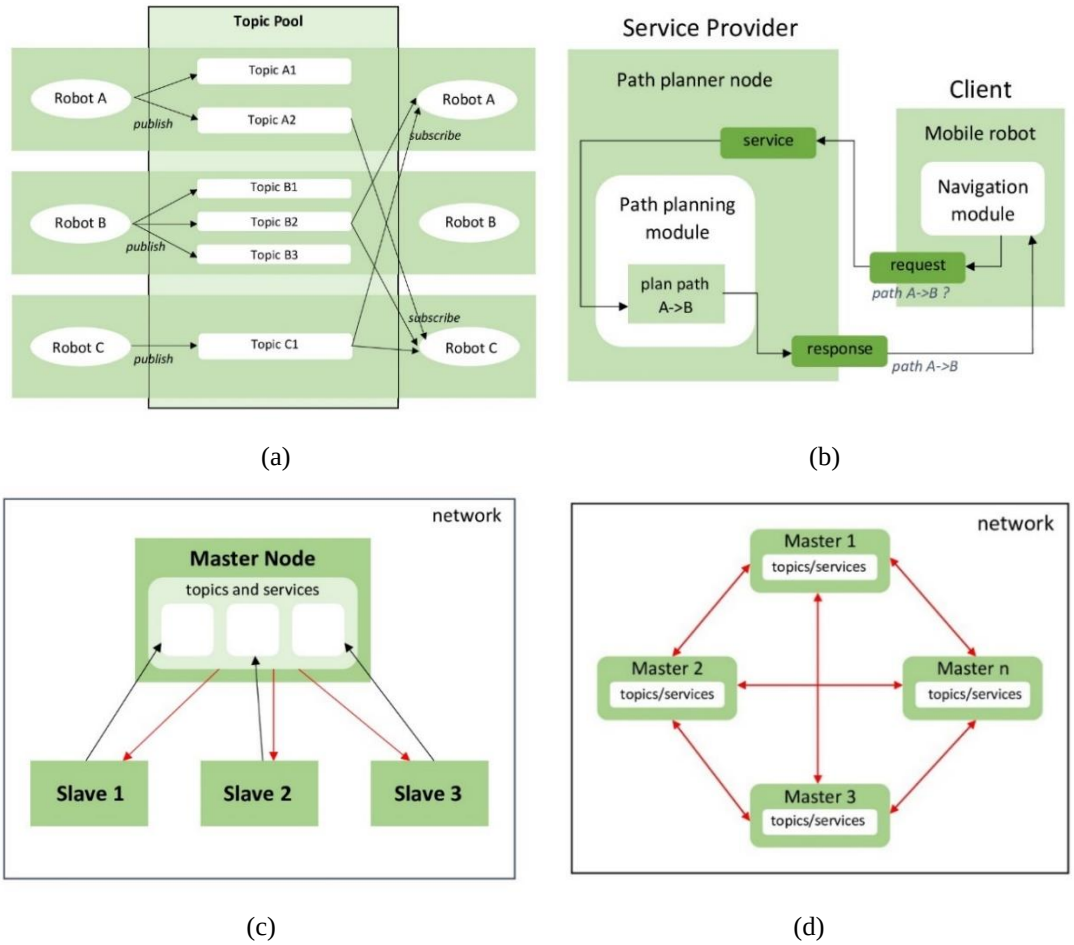
3.3.2 Multi-robot coordination

As the base prototype does not discuss about multi-robot system coordination scheme, the decision should be based on the literature survey in section 2.4.

The multi-robot coordination mechanisms belong to three categories: decentralized [42] [43] [46] [50] [51] [53], centralized [44] [54], and hybrid [45] [48] [49] systems. To select the most suitable method, Table 3.6 compares the key features of the three mechanisms.

All robots in decentralized systems behave independently, hence they are more fault tolerant than centralized mechanisms. The failure of one or a few robots will not affect the overall operation. Misfunctioning of the central controller leads a centralized system into a single point failure. A decentralized system can adapt to a cluster leader

failure by electing a new leader robot.



(a) (b)

(c) (d)

Figure 3.4: ROS compatible communication system architectures. (a) ROS Publisher/Subscriber communication (b) ROS Service client communication (c) ROS master Slave communication (d) ROS multi-master system

Table 3.6: Comparison of Multi-robot coordination schemes

Properties	Decentralized systems	Centralized systems	Hybrid systems
Fault tolerance	High	Low	High
Single point failure	Adaptable	Yes	Adaptable
Computational load on single agent	High	Low	Depends on the role
Repeated calculation	High	Low	Medium
Use of heterogeneous robots	Complex	Easy	Easy
Connectivity requirement	Low	High	Depends on the design
Complexity	High	Low	High

Extendibility	Easy	Complex	Easy

Due to the independent behavior, a robot in a decentralized system carries a high computational load, and the overall system may perform repeated calculations within units. As an example, multiple robots in a decentralized system may reach the same goal [43] and block each other. A centralized system divides its computational load among robots based on their capabilities. Therefore, it is possible to use heterogeneous robots in a centralized system.

On the other hand, centralized systems require high connectivity as the robots are unable to find new goals without the guidance of the central controller. This is an undesirable property for a multi-robot system operating in an unknown environment. A robot's mechanism in a decentralized coordination system is more complex because it must handle all the systems within one unit. But it is easier to extend the number of robots in a decentralized system since task division and role selection are not required.

It is observed that both centralized and decentralized systems have their fair share of advantages and disadvantages. Therefore, prominent works have suggested hybrid mechanisms using cluster-based fleets. In cluster-based architectures, the clusters act as decentralized units and centralized mechanisms are operated within the clusters or vice versa. This solves the drawbacks of purely centralized or purely decentralized systems discussed earlier. Hence, **a hybrid coordination mechanism** is suggested for this work.

3.3.3 Selected methodology

Looking at the requirements listed earlier in this section, a multi-master protocol is suitable for this work. Multimaster-fkie [71] ROS package was selected by considering the availability with ROS.

Multimaster_fkie system allows the masters to connect and disconnect to the network without intermediate configuration. As the master_discovery node advertises its presence periodically, a robot can easily make itself present in the network. If a master navigates out of range or disappears from the network, the protocol does not waste resources looking for its presence. Hence this package can achieve non-

continuous, event-based connectivity.

A robot can be configured as a master while its processes can be considered as nodes. Therefore, a single robot can be considered as a sub-system, hence the robot's independent behavior can be achieved.

Multi-master system is resistant to single point failure. The objective of creating a communication network in this application is to exchange environmental information among robots. If one robot fails, the system still can benefit from the information gathered from the remaining robots. Even a single unit can achieve the goals with less efficiency. Multimaster-fkie can function without discovering all the masters which are pre-configured in the system.

Master_sync node can synchronize a selected set of topics and services. In this application scenario, the 3D maps and pose information are the only required topics to be exchanged among remote masters. Hence the available bandwidth can be efficiently used.

By the nature of a multi-master system, multimaser-fkie is open to architectural extensions. Figure 3.5 displays the clustered, **hybrid multi-robot coordination** architecture that is proposed for this system. Small and centralized robot clusters are based on *master-slave* protocol. The clusters maintain *event-triggered* and *distributed* multi-master architecture between each other.

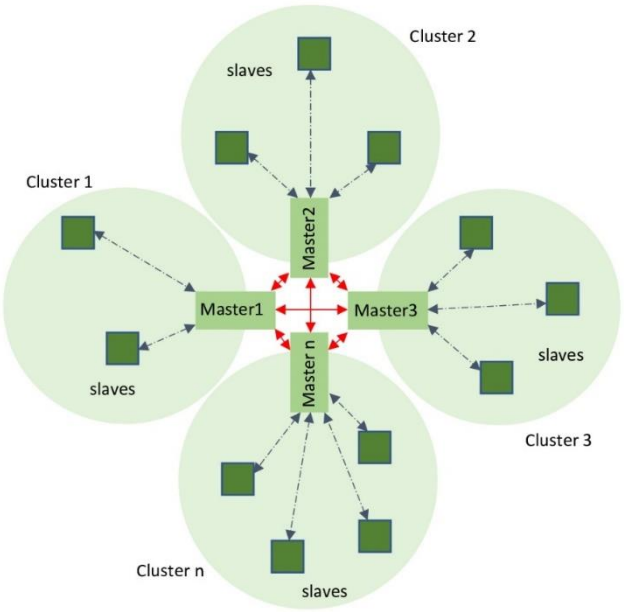


Figure 3.5: Extended clustered architecture of a multi-master system

The selected methodology for multi-robot coordination and inter-robot communication subsystem is summarized in Table 3.7.

Table 3.7: Selected methodology for multi-robot coordination and inter-robot communication

Feature	Methodology	Selected Technology
Multi-robot coordination	Hybrid coordination (as shown in Table 3.6)	Multimaster-fkie framework with clustered, hybrid multi-robot coordination
Inter-robot communication method	Event-triggered communication (as shown in Table 3.5)	

3.4 Autonomous Exploration

As the base prototype lacks autonomous exploration, a ROS compatible autonomous exploration system should be selected.

There are two main types of popular exploration strategies discovered in the section 2.5: frontier-based exploration and NBV exploration. As this work expects to cover a

large and unknown environment, **Frontier-based exploration** is selected.

Explore-lite [72] ROS package is selected as the fronter-based autonomous exploration system.

The selected methodology for this section is summarized in Table 3.8.

Table 3.8: Selected methodology for the autonomous exploration

Feature	Methodology	Selected Technology
Exploration method	Frontier-based	Explore-lite ROS package

3.5 Summary

The overall system methodology proposed through this section is illustrated in Figure 3.6. The next chapter discusses how the proposed methodology has been incorporated into the system design using suitable software tools and hardware platforms.

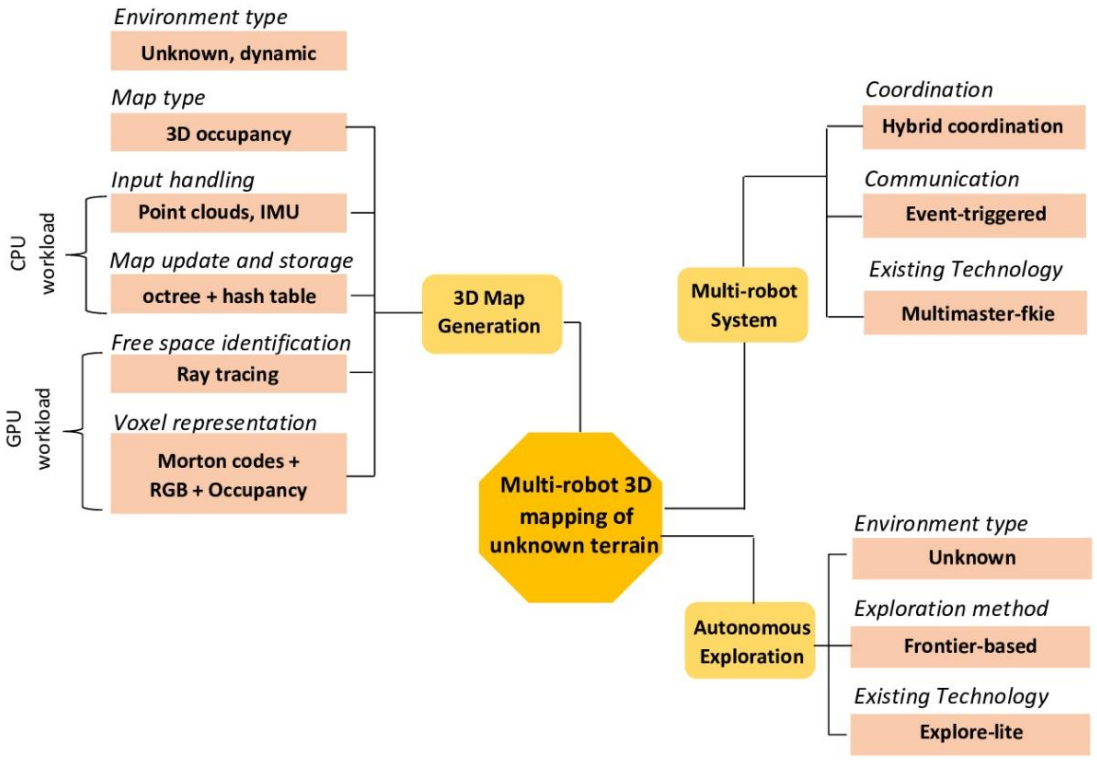


Figure 3.6: Overall system Methodology

4 SYSTEM DESIGN

4.1 Introduction

The previous chapter proposed the system methodology for high-performance multi-robot 3D mapping of unknown environments using parallel computing. Based on requirement comparison, the following techniques were selected.

- GPU parallelized 3D occupancy grid maps with Morton codes stored in hash tables
- Hybrid multi-robot coordination with event-triggered inter-robot communication
- Frontier based autonomous exploration

This chapter develops the system architecture of the system based on the selected techniques. It also selects the available state-of-the-art software and hardware tools. If any software tool is not available, it should be developed as part of the research.

4.2 System architecture

The system consists of hardware, supporting software frameworks and four main functional software components.

As the hardware platform, Qunaser QBot2 is used. Qbot2 is equipped with a Kobuki mobile base and a Microsoft Kinect 360 RGBD camera. NVIDIA Jetson TX2 is used as the mobile processing unit of the system. A general-purpose wireless router that is not connected to the internet is used to build the inter-robot communication network.

The system runs on Ubuntu 16.04 LTS operating system. ROS Kinetic distribution is used as the middleware between system hardware and software. CUDA architecture is used to parallelize possible computational tasks.

The four functional components of the final product are listed as following.

- GPU accelerated 3D Occupancy grid map building system
- Map merging system based on known initial poses
- Inter-robot communication system based on multi-master protocol

- Frontier based autonomous navigation system

From the above four components, GPU accelerated 3D occupancy grid mapping system is identified as parallelizable. The RGBD point cloud obtained from the depth camera has a resolution of 640x480, which is 307 200 points per frame. Systems like OctoMap [7] process the above-mentioned points serially. In this work, CUDA with C++ is used to process the points cloud parallelly inside the GPU. This GPU Accelerated 3D occupancy grid map building system (G3DOMap) is the main research area addressed in this thesis.

Map merging component is built on the assumption that the system has access to accurate initial poses of the robots. A leader node is selected to merge and store the central map.

Inter-robot communication is done through the multi-master communication protocol. This protocol is selected because of the uncertainty of the robots' locations throughout the exploration. The robots can synchronize data when possible and disconnecting will not fail the entire network. Multimaster_fkie [71], an open-source ROS package was used to integrate this functionality into the system.

A frontier-based, greedy exploration method is used for the autonomous navigation system. An individual robot looks for frontiers in its self-created map, travels to the frontier with the least exploration cost while locating itself in the 2D navigation map. Explore_lite [73] ROS package was used to integrate autonomous exploration functionality into the system.

An indoor office area was used as the real-world mapping environment. When appropriate environmental conditions like non-slippery floors and lighting are not available, Gazebo simulated worlds are used. All mapping and exploration visualizations are done in the RViz visualization tool.

Figure 4.1 represents the overall architecture of the system. The interactions between components will be discussed in the next section.

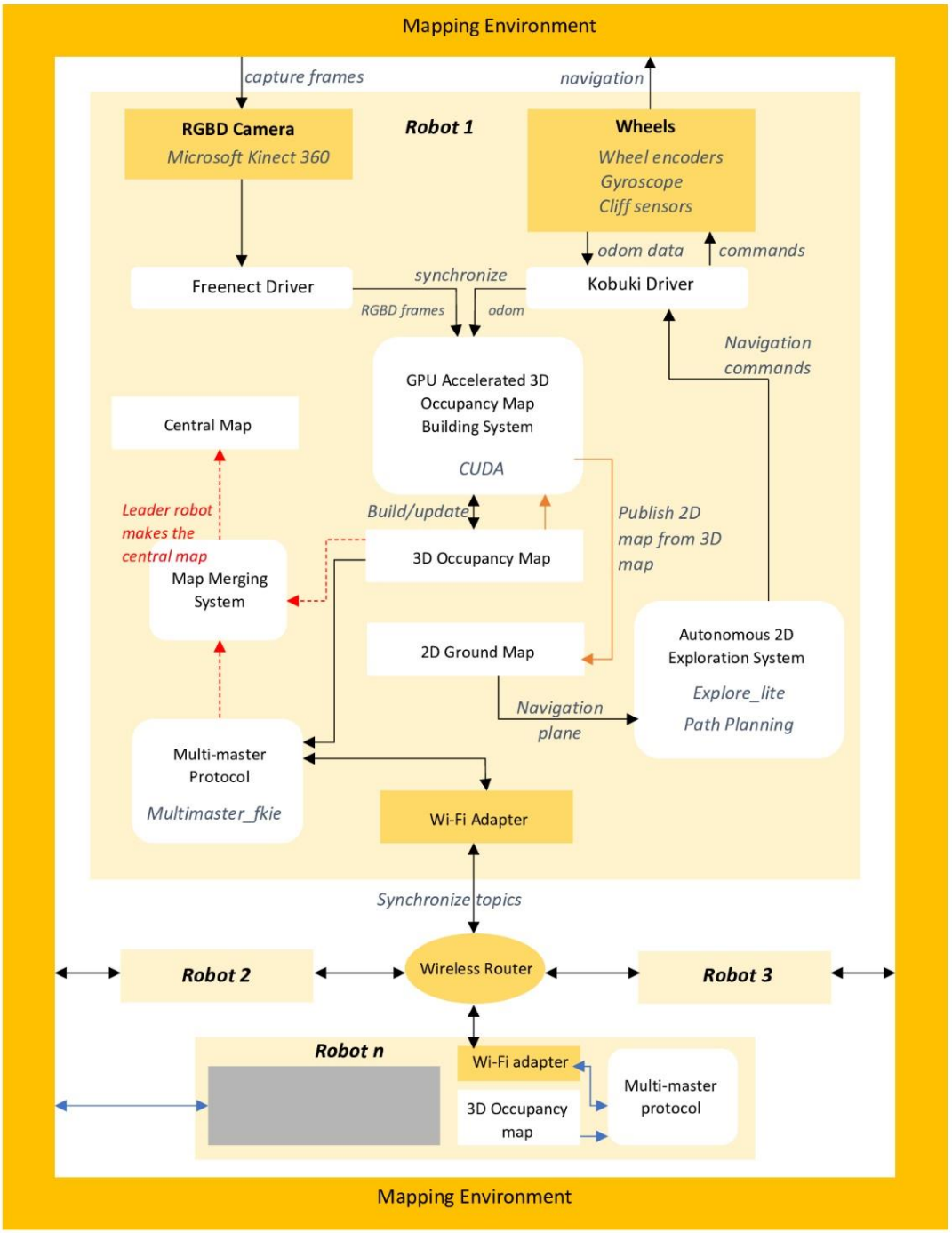


Figure 4.1: Overall system architecture

4.3 System data flow

As shown in Figure 14, the first step of the process is capturing real-world visual information using the depth sensor. Microsoft Kinect provides a 640x480 resolution RGB depth image at a frequency of 30 fps. Freenect ROS camera driver [74] is responsible for converting the image into a point cloud which is a readable array of (R, G, B, X, Y, Z) points. R G, B contains the color information, and X, Y, Z represents cartesian coordinates of the point with reference to the camera frame. In addition to visual data, it also obtains odometry data of the robot through the Kobuki ROS driver [75]. Knowing the current pose of the robot helps to directly transform the point cloud into the world coordinate frame. After receiving the current point cloud and current pose, they are synchronized and used to update the 3D occupancy grid map.

The 3D occupancy grid mapping system accepts a point cloud and the current pose as its inputs. The points are transformed and added to the 3D occupancy grid map. When the robot moves, the map is updated with new information. The 3D occupancy grid mapping system used in this work is named G3DOMap, and it will be discussed thoroughly in chapter 5. G3DOMap uses a hash table to save map nodes as voxels, and it can publish ground plane information about the environment.

The 2D ground map is used for the 2D navigation of the robot in unknown environments. At this point of the research, the environment is assumed to be even-surfaced. Hence a 2D ground map is sufficient for obstacle-free navigation. The autonomous exploration system uses a greedy approach to identify frontiers of the given map, then creates a safe path to navigate into the least costly frontier. Thus, the robot navigates in the environment while expanding the 3D map.

When multiple robots are involved, the system needs to share the 3D maps created by each robot and merge them into a central map. Multi-master protocol synchronizes all the ROS topics in the multi-robot system. Hence a robot can listen to the 3D maps of other robots. A selected node can merge the 3D maps of the environment and create a unified view of the world.

The hardware and software tools and their parameters are discussed in sections 4.4 and 4.5.

4.4 Implementation tools and technologies

The overall system is prototyped using following hardware equipment

- Microsoft Kinect 360 [76]
- Quanser Qbot2 [77] with Kobuki base
- NVIDIA Jeston TX2 [78] ,with GPU with 256 CUDA cores

The following software tools and technologies were used in the development and prototyping.

- Ubuntu 16.04 LTS
- Robot Operating System – Kinetic Kame version
- CUDA with C++ programming language

4.5 Evaluating CUDA performance compared to single threaded processing

A GPU can process large blocks of data in parallel, which makes more sense to image inputs because they have streams of localized data generated parallelly. Data in RGBD camera frames can be indexed meaningfully based on the pixels' locations in the frame. In this system, ROS depth camera drivers are responsible for labeling and indexing RGBD points. Using an Application Programming Interface (API) like CUDA leads to easy manipulation of the GPU, and it can achieve an acceptable level of speedup for the 3D map building process.

Table 4.1 shows CUDA capabilities of the selected processing unit, NVIDIA Jetson TX2. Note that NVIDA Jetson TX2 has an 8Gb main memory shared between CPU and GPU.

As a performance comparison between CPU and GPU computation time, a simple mathematical equation (1), and data sets of different sizes were used.

$$a^2 + b^2 = c \tag{1}$$

The results in Table 4.2 were observed after solving the vector Equation (1) using CPU and GPU. In both scenarios, time is measured only for the mathematical operation. Clock cycles taken for memory allocation, deallocation and memory exchanging have been excluded.

Table 4.1: CUDA parameters of NVIDIA Jetson TX2

Compute Capability [79]	6.2
Number of cores	256
Total global memory	8 Gb
Number of Multi processors	2
shared memory per Block	49,152 bytes
Number of Registers per Block	32,768
Threads in warp	32
max. threads per block	1024
max. thread dimension	$1024x1024x64$
max. grid dimension	$2{,}147{,}483{,}647x\ 65{,}535x\ 65{,}535$

Table 4.2: Simple vector equation time comparison between CPU and GPU

Data set size	**Time taken for the mathematical operation**		**Speedup**
	Serial time (Clock cycles)	**Parallel time (Clock cycles)**	
1,000,000	83,650	83	1008 X
2,560,000	122,824	88	1396 X
5,120,000	211,257	80	2641 X
51,200,000	1,658,140	80	20,727 X

It can be observed that the higher the input data size, the better the time performance. But the time taken for exchanging data between CPU and GPU has significantly affected the overall time performance. In the overall scenario, the system might not achieve the results indicated in Table 14. But GPU acceleration can provide a significant speedup to a parallelizable pipeline. After experimenting with different values, 256 was observed as the best number of threads per block for NVIDIA Jetson TX2.

4.6 Summary

This chapter presented the overall architecture of the system. It discussed about the four sub systems, how they interact with each other and what types of data streams are exchanged between them. As a special discussion, CUDA capabilities of NVIDIA Jetson TX2 were studied together with a performance comparison between CPU and

GPU.

The next four chapters will describe the four subsystems with design decisions, implementation details and results. The next chapter presents the implementation process of G3DOMap.

5 GPU ACCELERATED 3D OCCUPANCY GRID MAP BUILDING

5.1 Introduction

The section 3.2.2 analyzed OctoMap [7], the state-of-the-art 3D occupancy grid map building system. It concluded that OctoMap's time performance is limited by serial CPU threads, and it can be enhanced by parallelizing the point cloud registration process.

This chapter is dedicated to G3DOMap, the novel GPU accelerated 3D map generation system, its system design, implementation details and results. In the end, G3DOMap's quantitative and qualitative results are compared with OctoMap.

5.2 Parallel computing for 3D Occupancy grid map building

The main disadvantage of volumetric 3D representations like OctoMap is the large memory footprint [41]. As the number of voxels or the volumetric space grows, the memory consumption also grows linearly. This increases access and update times. Hence, the drawbacks discussed in the section 3.2.2 are experienced. These conditions are true for all the occupancy grid-based systems.

Over the years, the researchers have developed numerous algorithms for efficient maps. However, the computing community is rapidly switching towards parallel programming approaches to increase the throughput of their applications.

3D occupancy grids are built with voxels which are discrete and independent random variables [30]. With the recent availability of GPUs in mobile platforms, the 3D voxel operations can be easily parallelized. Following are such parallelizable voxel operations.

- Point transformation – when the robot pose is determined by the system, the point cloud should be transformed into the map's coordinate frame. All the points in the point cloud instance can be simultaneously transformed.
- Ray tracing in free space – for a given endpoint of the point cloud, the free space along that ray is considered free. A new free voxel can be generated for each

discrete free space unit. Since a ray is independent, ray tracing can be parallelized per endpoint.

- Generating voxels – G3DOMap uses an octree-based map, hence all the voxels are represented by tree nodes. The cartesian position, color, and occupancy are stored per node. All nodes can be created parallelly.

5.3 System Design of G3DOMap

5.3.1 Introduction

Using GPUs for 3D occupancy grid mapping is rather new to the mobile robotics field. GPU acceleration is widely used by scene reconstruction and object modeling approaches. But they belong to a different application field.

By looking at the prominent work listed in section 2.3, it is observed that similar systems such as mobile robot SLAM have used a CPU-GPU hybrid pipeline to achieve their performance while video rendering and object modeling systems entirely run in GPUs.

G3DOMap uses GPU acceleration for the point cloud registration process including point cloud transformation, ray tracing, and creating new voxels.

5.3.2 Architecture

Two mapping modes have been developed in the G3DOMap. The first mode is simpler and more lightweight, as it registers only the occupied voxels. This is an efficient and minimalistic design for mapping large and static areas for human operators. The second mode is a full environmental model which registers occupied, free, and unknown voxels. It can be used to model dynamic environments and derive navigation maps.

For the real-world testing of G3DOMap, Quanser Qbot with a Kobuki base and a Microsoft Kinect 360 RGBD camera is used. It has also used Gazebo simulated worlds with turtlebot platform. The computation was done on a computer with Intel(R) Core (TM) i7-9750, 2.60 GHz CPU, NVIDIA GeForce RTX 2070 GPU, and 16 GB RAM.

5.3.3 Data flow

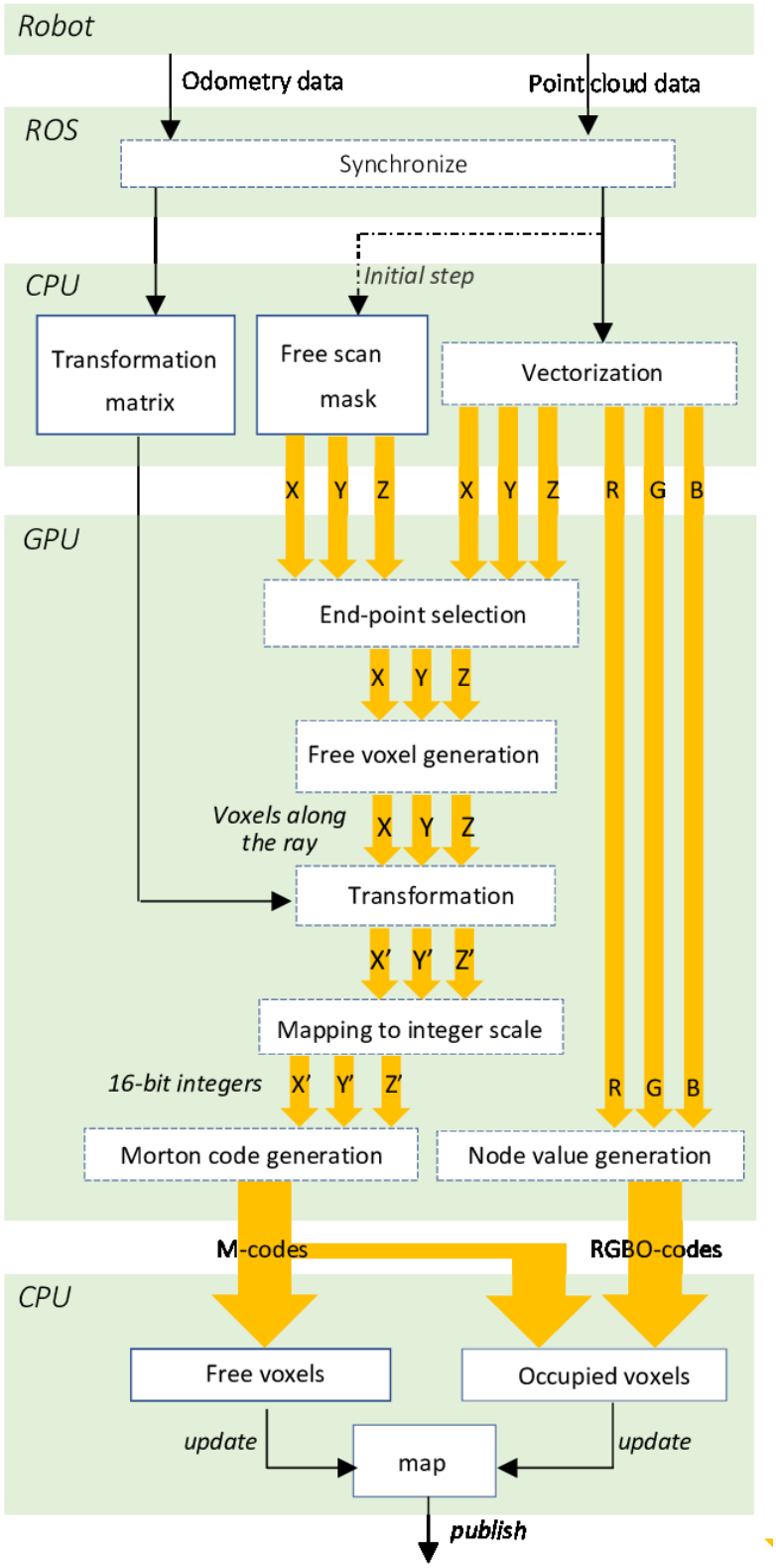


Figure 5.1: Data Flow Diagram of G3DOMap

Figure 5.1 shows the abstract data flow among Robot, ROS, CPU and the GPU. First, it obtains Robot's RGBD data and IMU data as ROS point cloud messages and ROS odometry messages respectively. The two message streams are synchronized through ROS. A special fully observed point cloud is used to generate the *free scan mask* prior to the mapping process. After reading the ROS point cloud message, the X, Y, Z, R, G, B data are arranged into separate ordered arrays. The transformation matrix is obtained based on the odometry data. At the end of this stage, the arranged data is fed into the GPU.

The entire node generation process of G3DOMap is executed within the GPU. In there, positional data is used to identify the free space, the points are transformed into the original coordinate frame, and those points are distributed within an integer scale based on the required resolution. Once the positional data are converted into integers, Morton codes (M-codes) are generated per voxel. R, G, B data arrays and observed occupancy are combined to generate the voxel values. These node values are named as RGBO-codes. 64-bit long M-codes and 32-bits long RGBO-codes are sent into the CPU, where the central map is maintained as a hash table.

The upcoming sections (5.4 - 5.10) will discuss each stage of G3DOMap illustrated in Figure 5.1.

5.4 Preparing data arrays

The very first step of the system is synchronizing input data. The point cloud data and the odometry data are considered time-sensitive, especially when the robot is moving fast. At this stage, it is assumed that the robot's odometry data are either reliable or have been corrected by some previous step.

After synchronizing, the transformation matrix in equation (12) is obtained from the odometry data instance. The transformation matrix is used later for registering the obtained point cloud instance into the world coordinate frame. The vectorization process involves decoding the ordered point cloud message and preparing it as separate arrays to send into GPU as vectors. The useful information from the point cloud is extracted to prepare the following arrays in equations (2) - (7). Here $n*m$ is the resolution of the RGBD camera.

$$X = [x_0, x_1, x_2, \dots, x_{nm-1}] \quad (2)$$

$$Y = [y_0, y_1, y_2, \dots, y_{nm-1}] \quad (3)$$

$$Z = [z_0, z_1, z_2, \dots, z_{nm-1}] \quad (4)$$

$$R = [r_0, r_1, r_2, \dots, r_{nm-1}] \quad (5)$$

$$G = [g_0, g_1, g_2, \dots, g_{nm-1}] \quad (6)$$

$$B = [b_0, b_1, b_2, \dots, b_{nm-1}] \quad (7)$$

For the Microsoft Kinect 360, the RGBD resolution is 640x480. Hence above (2) - (7) arrays have 307,200 elements per instance. X, Y, Z arrays contain positional information of each point of the cloud with reference to the camera frame. These vectors are stored as three *float* type arrays. R, G, B arrays store color information of points and they are defined as *8-bit unsigned integer* arrays to allocate minimum memory for them.

At the initial step of the map building process, the free scan mask (section 5.5.1) is also arranged into vectors. Since the free scan mask is a constant point cloud obtained under special circumstances, this step only needs to be executed once. The following arrays in equations (8) - (10) are arranged from the free scan mask.

$$X_{end} = [x_0, x_1, x_2, \dots, x_{nm-1}] \quad (8)$$

$$Y_{end} = [y_0, y_1, y_2, \dots, y_{nm-1}] \quad (9)$$

$$Z_{end} = [z_0, z_1, z_2, \dots, z_{nm-1}] \quad (10)$$

Arrays in equations (2) – (10) and the transformation matrix in equation (12) are sent into the GPU for parallel processing.

5.5 Ray tracing in free space to mark free occupancy voxels

Ray tracing in free space is the process of identifying all the free voxels along a ray, starting from the origin endpoint. In a point set with *n*m* number of points, it is common to find rays that do not include data. In a cluttered environment, the number of informative rays is high as a larger number of rays meet obstacle surfaces. When registering the points into the occupancy grid map, the certainty range of the RGBD

camera should be considered. For Microsoft Kinect 360, the certainty range is defined between 0.8 m and 3.5 m [76] in the X-direction. Figure 5.2 shows the variability of the density of the point cloud with the distance from the camera.

Figure 5.3 shows a real-world example of the variability of point cloud density with the distance from the camera. The point density of the bear and the side of the couch is high as these objects sit within the camera's certainty bounds. But the cardboard box that is out of the certainty range looks more faded. The reason for this is the spherical emission of laser beams by the RGBD camera, which makes the data points closer to the depth camera denser [19]. The cardboard box became visible when the point size was increased to 5 mm.

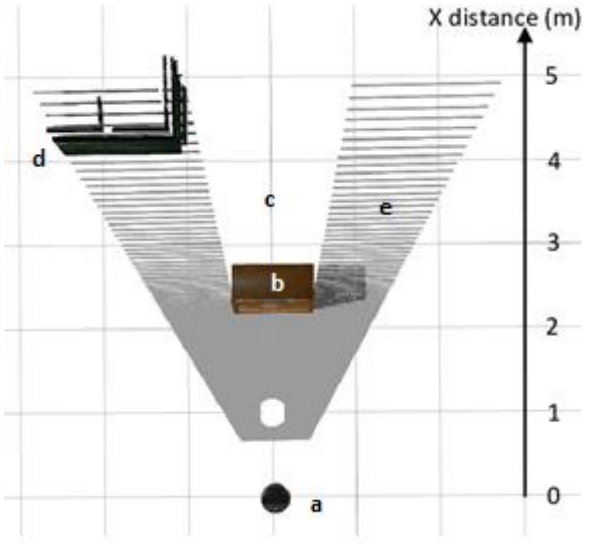


Figure 5.2: Bird eye view of a point cloud obtained from a simulated scene. (a) the robot, (b) bookshelf, (c) the space where the point cloud is absent due to b, (d) a trash bin with open slits, (e) the point cloud.

Due to the lower point densities of the clouds out of the certainty range, a simple miscalculation or an error of the point index can lead to a higher positional error. Thus, it can be established that the points within the certainty bounds are more promising and reliable for mapping. Therefore, it is decided to neglect the points which lie out of certainty bounds, even for marking free space.

The next problem arises when mapping proportionally empty regions. When most

of the scans fall at relatively far surfaces, and the number of empty scans is high, identifying free space around the robot becomes problematic. The *free scan mask* is introduced as a solution to the above problem.

Figure 5.3: Effect of the distance to the point cloud density (a) RGB camera view of the scene. Selected objects are marked in numbers: (1) Bear – 2m from the camera, (2) – couch – 2.6 m from the camera, (3) – cardboard box – 4 m from the camera. (b) the point cloud displayed in 1mm points, (c) the point cloud displayed in 5mm points

5.5.1 Free scan mask

As introduced before, the free scan mask is a special point cloud obtained before the map building process. All the *n*m* points in the free scan mask lie between the sensor's certainty range, and there are no empty scans. It acts as a set of endpoints per each scan when the actual endpoints are not known. Figure 5.4 shows the free scan mask used in this system. It has been obtained by placing the RGBD camera in front of a wall, having approximately 3 m distance between them. The evenness and the texture of the wall are not important parameters as long as all the *n*m* rays fall on it.

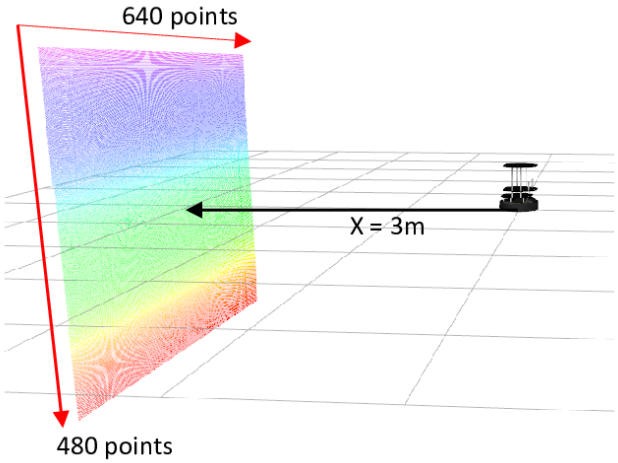


Figure 5.4: The free scan mask for Microsoft Kinect 360

The free scan mask is stored with reference to the camera frame. Because of that, it needs to be transformed to the current pose of the robot. But the free scan mask can be re-used for a device if the rays pass through the air.

Now that the system has obtained an array of endpoints to all possible rays in a scan instance, it can generate the free voxels along each ray. Whenever a ray does not have a valid endpoint, it can be replaced by the corresponding point in the free scan mask.

5.5.2 Free space voxel identification

The 3D version of Bresenham's line algorithm [80], [81] is used to identify the occupancy grid voxels crossed by a ray. Algorithm 1 shows how the above algorithm generates an array of free space voxels per ray. The algorithm is executed *m*n* times per point cloud instance.

The size of ray[*n*] array can be predefined in terms of current mapping parameters. The maximum number of free voxels (*n*) for a ray is generated when the end point almost falls on the certainty bound. Equation (11) can be used to calculate *n*.

$$n = ceil\ (\frac{maximum\ sensor\ radius}{resolution}) \tag{11}$$

Although, the occupied points and free space points are known at this step, the point cloud is still in its original resolution except for the free space points.

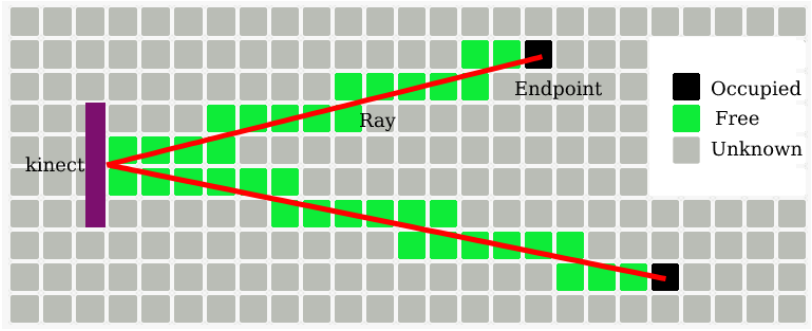


Figure 5.5: Ray tracing in free space to identify free voxels

Algorithm 1: Ray tracing in free space using 3D Bresenham's line drawing algorithm

Algorithm 1: Ray Tracing

Input: $Pi(X_i, Y_i, Z_i)$, X_{end}, Y_{end}, Z_{end}, $resolution = res$
Output: $ray[n]$

```
1  (X1, Y1, Z1) = (0, 0, 0) //origin
2  (X2, Y2, Z2) = (Xend, Yend, Zend) //end point
3  if Pi within certainty bounds then
4  |  (X2, Y2, Z2) ← (Xi, Yi, Zi) //actual end point
5  (dx, dy, dz) = (|X2–X1|, |Y2–Y1|, |Z2–Z1|)
6  sx = sy = sz = res //step size
7  if (X2–X1) ≤ 0 then
8  |  sx = -res
9  if (Y2–X1) ≤ 0 then
10 |  sy = -res
11 if (Z2–Z1) ≤ 0 then
12 |  sz = -res
13 if (dx ≥ dy) AND (dx ≤ dz) then
14 |  //the ray drives along X-axis
15 |  py = 2dy – dx
16 |  pz = 2dz – dx
17 |  index = 0
18 |  while (|X1–X2| > res/2) do
19 |  |  X1 ← X1 + xs
20 |  |  if py ≥ 0 then
21 |  |  |  Y1 ← Y1 + sy
22 |  |  |  py = py–2dx
23 |  |  if pz ≥ 0 then
24 |  |  |  Z1 ← Z1 + sz
25 |  |  |  pz = pz–2dx
26 |  |  py = py + 2dy
27 |  |  pz = pz + 2dz
28 |  |  ray[index] = (X1, Y1, Z1)
29 |  |  index ← index + 1
30 else if (dy ≥ dx) AND (dx ≤ dz) then
31 |  //the ray drives along Y-axis
32 |  Equivalent process as (13-29)
33 else if (dz ≥ dx) AND (dz ≤ dy) then
34 |  //the ray drives along Y-axis
35 |  Equivalent process as (13-29)
36
```

5.6 Transformation and mapping into integer scale

Up to this stage, the points were in the reference frame of the RGBD camera. They need to be transformed into the map's coordinate frame. The point cloud can be transformed according to the homogeneous equation (12), given that $P(X, Y, Z)$ is the original point, $P'(X', Y', Z')$ is the transformed point, $x_{trans}, y_{trans}, z_{trans}$ are the linear translations along respective axes, and α, β, γ are the yaw, pitch and roll of the robot.

$$\begin{pmatrix} X' \\ Y' \\ Z' \\ 1 \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \begin{pmatrix} cos\alpha.cos\beta & cos\alpha.sin\beta.sin\gamma - sin\alpha.cos\gamma & cos\alpha.sin\beta.cos\gamma + sin\alpha.sin\gamma & x_{trans} \\ sin\alpha.cos\beta & sin\alpha.sin\beta.sin\gamma + cos\alpha.cos\gamma & sin\alpha.sin\beta.cos\gamma - cos\alpha.sin\alpha & y_{trans} \\ -sin\beta & cos\beta.sin\gamma & cos\beta.cos\gamma & z_{trans} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (12)$$

As the map building system does not depend on specific hardware constraints, all the parameters in the transformation matrix are used.

After transformation, the positional coordinates of the points are still in floating-point format. As the octree nodes will be addressed using Morton codes, they must be represented as unsigned integers. Using the same configuration as OctoMap, G3DOMap defines its tree depth as 16. Recursive division of 3D space into eight voxels can be simply interpreted as dividing each dimension into two equal sections. Thus, for a depth of 16, a maximum number of 216 voxels are required in one dimension. This leads to total $(2^{16})^3$ voxels in the whole octree with the above depth. The above floating points must be mapped into unsigned integers in the range of [0, 65535] as required by the Morton code generation process. The process is defined in Equations (13) - (15).

$$X_{int} = (2^{depth-1}) - 1 + ceil\left(\frac{X_{float}}{resolution}\right) \quad (13)$$

$$Y_{int} = (2^{depth-1}) - 1 + ceil\left(\frac{Y_{float}}{resolution}\right) \quad (14)$$

$$Z_{int} = (2^{depth-1}) - 1 + ceil\left(\frac{Z_{float}}{resolution}\right) \quad (15)$$

Considering two example points P_1 (1.34442, 2.00453, 0.43599), and P_2 (1.34442,

2.00453, -0.43599) with the depth 16 and resolution 5cm, the following integer coordinates can be calculated.

$P_1 \rightarrow$ (32 794, 32 808, 32 776)

$P_2 \rightarrow$ (32 794, 32 808, 32 759)

the equations are designed such that negative floating-point values are converted into integer values in the range [0, $2^{depth-1}$). To represent the complete set of unsinged integers in one axis, a memory of 16 bits is required. For a point (X, Y, Z), a total memory of 48 bits is required.

5.7 Generation of Morton codes and octree node values

Once the positional coordinates are converted into unsigned integers, they can be used for generating Morton codes.

A Morton code, also known as the Z-order function, is a bitwise technique used for linearly representing hierarchical trees. It is suitable for generating octree nodes inside GPUs because the tree hierarchy can be built without pointers and class objects. Calculating Morton code for a 3D point involves simple bit operations. Let's consider the above point P1 which is already converted into unsigned integers. The integer coordinates should be first represented in binary format.

$$P_1 \rightarrow (32794, 32808, 32776)$$
$$\rightarrow (1000000000011010, 1000000000101000, 1000000000001000$$

The above 16-bit binary codes can be generally represented using Equations (16) - (18).

$$P_x = X_{15}X_{14}X_{13} \dots X_2X_1X_0 \quad (16)$$
$$P_y = Y_{15}Y_{14}Y_{13} \dots Y_2Y_1Y_0 \quad (17)$$
$$P_Z = Z_{15}Z_{14}Z_{13} \dots Z_2Z_1Z_0 \quad (18)$$

Morton code is generated by interleaving the bits in each binary number. Equation (19) presents the generalized Morton code of $P(P_x, P_y, P_z)$ for 16-bit coordinates.

$$Morton\ code\ of\ P = Z_{15}Y_{15}X_{15}Z_{14}Y_{14}X_{14} \dots \dots Z_1Y_1X_1Z_0Y_OX_0 \quad (19)$$

Hence the Morton code of P_1 is:
1110000000000000000000000000000010001111000001000 (48 bit)

The Morton codes of G3DOMap are 48 bits long. However, a 64-bit data word is required for storage with extra 16 bits remaining empty. If the whole length of the 64-bit long data word is utilized, it can be used to represent an octree depth of 21.

The most special feature of Morton code is its representation of tree hierarchy. An octree node's eight children can be uniquely labeled using 3 bits (Figure 5.6).

Thus, the Morton code in above equation (19) can be grouped as 3-bit chunks as in the equation (20). There are 16 groups, each 3-bits representing the corresponding octree node to traverse to find the endpoint.

$$Morton\ code = (M_{47}M_{46}M_{45})(M_{44}M_{43}M_{42}) \dots \dots (M_5M_4M_3)(M_2M_1M_0) \quad (20)$$

5.8 Creating octree node values

By this stage, the octree nodes are ready as 64-bit Morton codes. The next step is to prepare the information of occupied nodes in a linear format. This procedure is straight forward as the RGB values are already in the format of unsigned 8-bit integers. The observed occupancy value is also represented as an 8-bit unsigned integer. An occupied node's value (RGBO-code) can be represented using a 32-bit data word as in equation (21).

$$RGBO - code = (R_7R_6 \dots R_1R_0)(G_7G_6 \dots G_1G_0)(B_7B_6 \dots B_1B_0)(O_7O_6 \dots O_1O_0) \quad (21)$$

Finally, the array of M-codes for occupied voxels and free voxels, and the array of RGBO-codes for occupied voxels are output from the GPU. Larger the voxel size, higher the number of redundant M-codes in the output array. Nevertheless, the redundant octree nodes can increase the number of observations for a voxel and help increasing its occupancy value.

5.9 Updating the map

This section discusses how the octree nodes are stored, updated, and published upon

request. At the end of this section, the Morton-coded octree will be compared with the traditional octree.

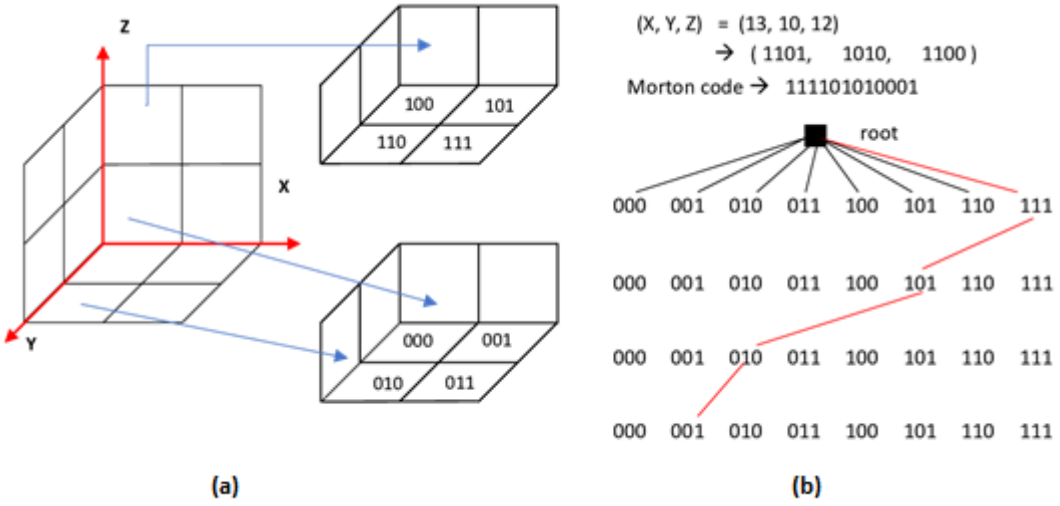


Figure 5.6: Morton code for an octree structure. (a) labelling eight children of an octree node based on their position. (b) locating a 3D point in a 4-level hierarchical octree using a 12-bit Morton code

5.9.1 Traditional octree

An octree is a hierarchical, recursive data structure which stores three dimensional nodes based on their position in 3D space. Algorithm 2 defines the most primitive structure of an octree node.

Algorithm 2: Traditional octree node

Algorithm 2: OctreeNode

```
1  Struct OctreeNode{
2     float X,Y,Z ⌝//Node's position
3     uint8_t R,G,B ⌝//Node's color
4     float occupancy ⌝//Occupancy value
5     OctreeNode *BNE ⌝//Bottom North East node
6     OctreeNode *BSE⌝//Bottom South East node
7     OctreeNode *BSW⌝//Bottom South West node
8     OctreeNode *BNW ⌝//Bottom North West node
9     OctreeNode *TNE⌝//Top North East node
10    OctreeNode *TSE⌝//Top South East node
11    OctreeNode *TSW⌝//Top South West node
12    OctreeNode *TNW ⌝//Top North West node
13 }
```

Each OctreeNode uses X, Y, Z position as the node identifier, and stores RGB values and occupancy value as node information. Each node carries pointers to its eight children. The leaf nodes exist at the maximum depth of the tree and their child nodes are empty. Figure 5.7 illustrates the traditional octree hierarchy.

Cutting the octree at any depth level provides access to a less resolution level of the tree. If the octree with ***d*** levels has a maximum resolution of ***m***, the resolution at level ***l*** is given by Equation (22).

$$resolution\ at\ level\ l = m.2^{d-1} \tag{22}$$

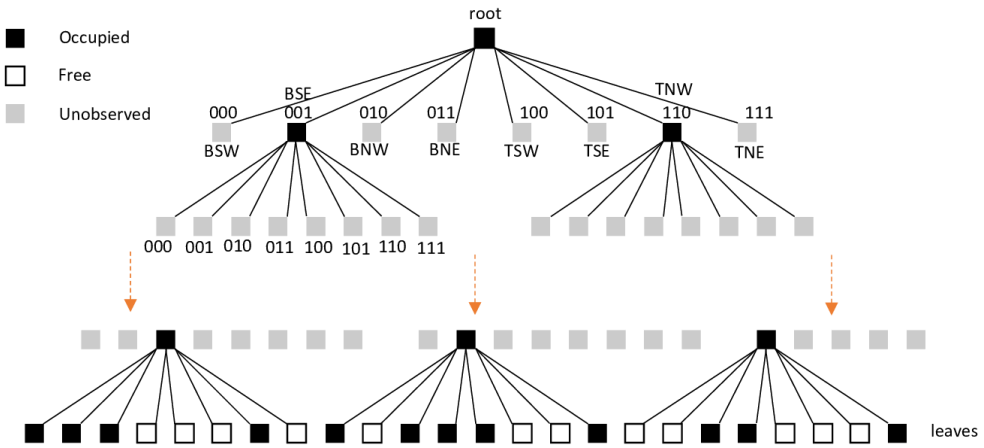


Figure 5.7: Traditional octree hierarchy with children node identifiers

5.9.2 Morton coded octree

Morton codes can be used for a low-level implementation of hierarchical octrees. Recursive tree traversal is not required when updating nodes due to the linear hierarchy. Thus, the octree can be treated as an ordinary array of data. In this implementation, the *M-codes* and *RGBO-codes* are stored in an unordered map that has an underlying hash table data structure. A random node search in the hash table has the time complexity of *O(1)* at the optimal case. Since the RGBO-codes are 32-bit binary data words, the node updating algorithms can be written as bit-level operations. Algorithm 3 and Algorithm 4 represent the algorithms for updating a voxel, which is last observed as occupied and free respectively.

Algorithm 3: Update Morton coded octree for an occupied voxel

Algorithm 3: UpdateTree

Input: $M-code$, $RGBO-code$, $Tree$

$Tree.search(M-code)$
if *(Node not found)* **then**
  $Tree.insert(M-code, RGBO-code)$
else
  $N = Node.value$ //Existing node value, 32-bits
  $M = RGBO-code$ //Current RGBO-code, 32-bits
  R, G, B //8-bit colors
  O //8-bit occupancy
  if *(Node has color information)* **then**
    //Average using bit operations
    $R = [(N_{31}N_{30}\dots N_{25}N_{24}) >> 1] + [(M_{31}M_{30}\dots M_{25}M_{24}) >> 1]$
    $G = [(N_{23}N_{22}\dots N_{17}N_{16}) >> 1] + [(M_{23}M_{22}\dots M_{17}M_{16}) >> 1]$
    $B = [(N_{15}N_{14}\dots N_{9}N_{8}) >> 1] + [(M_{15}M_{14}\dots M_{9}M_{8}) >> 1]$
  else
    $R = (M_{31}M_{30}\dots M_{25}M_{24})$
    $G = (M_{23}M_{22}\dots M_{17}M_{16})$
    $B = (M_{15}M_{14}\dots M_{9}M_{8})$
  $O = (N_7N_6\dots N_1N_0)$
  if $(O < 32)$ **then**
    $O \leftarrow O + 4$
    if $(O > 32)$ **then**
      $O \leftarrow 32$ //Integer Upper bound for occupancy
  $RGBO-code = (R << 24)\ \&\ (G << 16)\ \&\ (B << 8)\ \&\ O$
  $Node.value \leftarrow RGBO-code$

Algorithm 4: Morton code octree update algorithm

Algorithm 4: UpdateTree

Input: $M-code$, $Tree$

$Tree.search(M-code)$
if *(Node not found)* **then**
  $Tree.insert(M-code, uint\_32(15))$ //15 is the upper bound for free
else
  $N = Node.value$
  $O = (N_7N_6\dots N_1N_0)$
  if $(O > 0)$ **then**
    $O \leftarrow O - 1$
  $N = (N\ \&\ 11111111111111111111111100000000)\ \&\ O$
  $Node.value \leftarrow N$

When updating a voxel as occupied, it is given a higher weight than a free voxel because observing an obstacle node is more critical for a self-navigating robot. When assigning occupancy values for a node, values in the range [0,15] are assigned for free voxels where 0 is the lowest possible occupancy. The value 16 is assigned for unknown nodes, and the values in the range [17,32] are assigned for occupied voxels. 32 is the highest occupancy a voxel can possess. Since there are eight bits assigned for the occupancy value, it is possible to adjust these ranges according to the applicational need.

The resolution control rules mentioned in section 5.9.1 are also valid for Morton coded octrees. Consider the voxel P_1 from the previous example and group its M-code as in equation (20). Table 5.1 simplifies the octree search by identifying each octree depth level and the corresponding child node to derive from the immediate parent node.

P_1 = 11100000000000000000000000000010001111000001000

Table 5.1: Octree hierarchy derived from Morton code

M-code	11100000000000000000000000000010001111000001000															
	111	000	000	000	000	000	000	000	000	000	010	001	111	000	001	000
depth	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
child	TNE	BSW	BSW	BSW	BSW	BSW	BSW	BSW	BSW	BSW	BNW	BSE	TNE	BSW	BSE	BSW

These nodes can be cut at any depth level and derive maps with less resolutions.

5.9.3 Comparison with traditional octree

To prove that Morton coded octrees have better time performance than traditional octrees, a comparison is performed using large sets of (X, Y, Z) and RGB values. Figure 5.8 presents the results of tree update times of Morton coded octree and traditional octree. The time measurements include CPU-GPU memory exchange times. Both algorithms are executed in a CPU and GPU mixed environment.

It can be observed that the Morton coded 3D occupancy grid map generation is over 2 times as fast as the traditional octree-based method.

5.10 Decoding and publishing the map

The previous sections discussed the 3D occupancy grid map building process of G3DOMap. The map is stored as a hash table where the element keys are 64-bit Morton codes. Therefore, when the map is requested for publishing, the voxels need to be decoded for deriving actual 3D space coordinates. The first step of decoding is rearranging the M-code bits to obtain 16-bit integer coordinates as in equations (23) - (25).

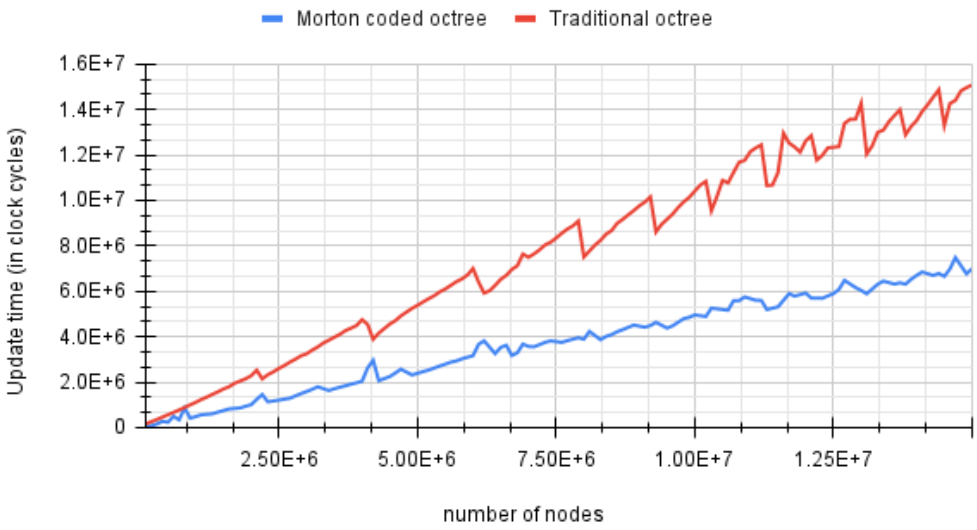


Figure 5.8: Tree update time with the number of nodes: Morton coded octree Vs. traditional octree. The measurements include the time taken for memory exchange between CPU and GPU.

$$M - code = M_{63}M_{62}M_{61} \dots \dots M_3M_2M_1M_0$$

$$X_{int} = M_{45}M_{42}M_{39}M_{36} \dots \dots M_9M_6M_3M_0 \quad (23)$$

$$Y_{int} = M_{46}M_{43}M_{40}M_{37} \dots \dots M_{10}M_7M_4M_1 \quad (24)$$

$$Z_{int} = M_{47}M_{44}M_{41}M_{38} \dots \dots M_{11}M_8M_5M_2 \quad (25)$$

The next step is converting the unsigned integers back to the floating-point values using equations (26) - (28).

$$X_{float} = (X_{int} + 1 - 2^{depth-1}).resolution \quad (26)$$

$$Y_{float} = (Y_{int} + 1 - 2^{depth-1}).resolution \quad (27)$$

$$Z_{float} = (Z_{int} + 1 - 2^{depth-1}).resolution \quad (28)$$

It should be noted that this procedure cannot restore the *P (X, Y, Z)* point in its original precision. The result will be a precision reduced value depending on the map resolution.

After extracting RGB values from obstacle nodes, the points are published for visualization.

5.11 Resource consumption and results

5.11.1 System parameters

The results presented in this section are based on the Gazebo simulated environment displayed in Figure 5.9. There are obstacles with multiple shapes, outlines and surface textures in the simulated environment. The system was run on a computer with the following speculations.

- CPU - Intel(R) Core (TM) i7-9750H, 2.60 GHZ, 16 GB Memory
- GPU - NVIDIA GeForce RTX 2070, 8 GB memory

5.11.2 Static map building

Static environment 3D occupancy grid mapping pipeline can be broken down into five independent tasks. Three of those tasks are identified as parallelizable. Table 5.2 sums up the average times taken by each process when executed serially and parallelly

Table 5.2: Process breakdown and time comparison in static map model

Process		**Average time taken (clock cycles)**	
		Serial data processing	Parallel data processing
1	Vectorize the point cloud	10,968	serial
2	Mapping to integer scale and transform points	8084	27
3	Generate Morton codes	15,272	
4	Generate node values	2031	
5	Update the octree	31,812	serial
Total time (including setup, memory transactions)		68,197	41,488
Speedup		*X*	1.64 *X*

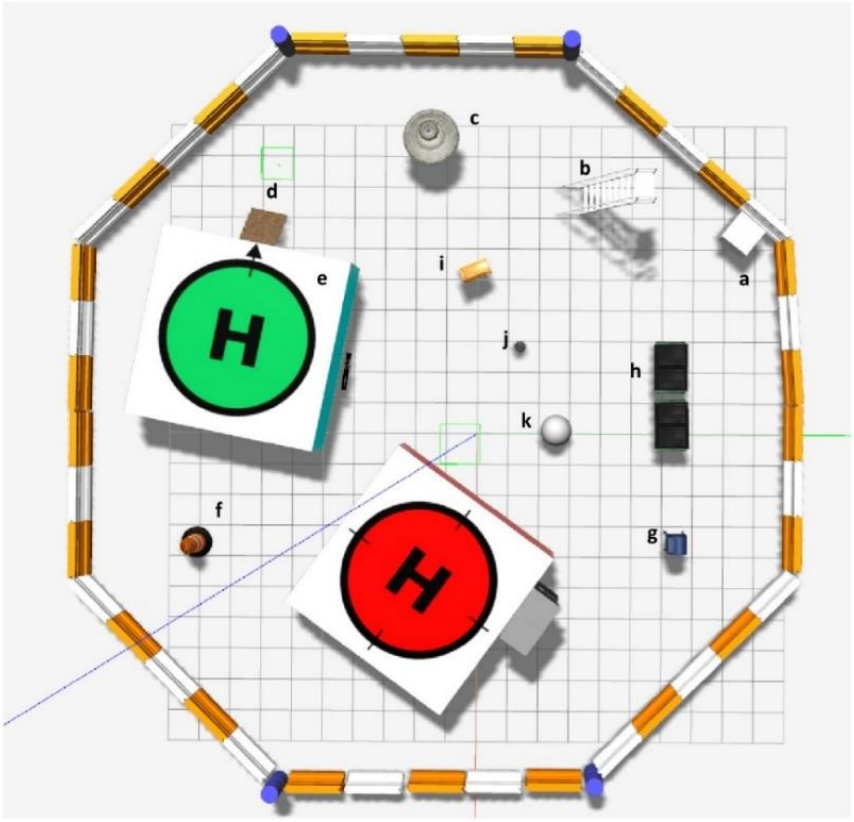


Figure 5.9: The Gazebo simulated world. (a) cube (b) step ladder (c) cement fountain (d) stairs (e) rectangular building (f) traffic cone (g) mailbox (h) duster (i) bookshelf (j) Qbot2 robot (k) sphere

5.11.3 Dynamic environment map building

The dynamic environment map building pipeline can be divided into six independent stages. In addition to the processes in the static map model, it contains ray tracing. Table 5.3 summarizes the task breakdown and their time consumptions as serial and parallel processes.

When comparing the results in Table 5.2 and Table 5.3, the better speedup is observed in the dynamic environment mapping pipeline. The reason for this is the higher increment of the number of nodes in the octree due to free space mapping (Figure 5.10). The fraction of free space in a map is often higher than the fraction of

occupied space. The number of new nodes generated after ray tracing a point cloud instance is represented by equation (29).

$$new\ voxels\ per\ scan_{(maximum)} = m.n.\left(\frac{maximum\ sensor\ radius}{resolution}\right) \quad (29)$$

Table 5.3: Process breakdown and time comparison in dynamic map model

Process		Average time taken (clock cycles)	
		Serial data processing	Parallel data processing
1	Vectorize the point cloud	10,639	serial
2	Mapping to integer scale and transform points	7798	30
3	Ray casting for free space	688,069	
4	Generate Morton codes	957,909	
5	Generate node values	1954	
6	Update the octree	1,906,506	serial
Total time (including setup, memory transactions)		3,585,312	1,352,503
Speedup		X	2.65 X

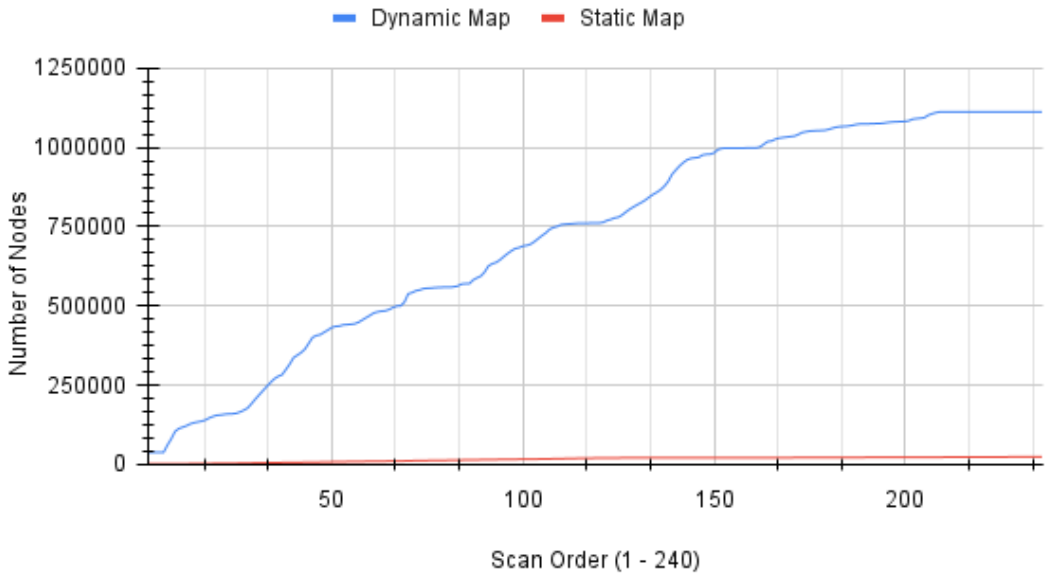


Figure 5.10: Number of tree nodes after registering each of first 240 scans

5.12 Comparison between OctoMap and G3DOMap

5.12.1 System Parameters

The mapping environment, and processing unit are similar to the parameters mentioned in 5.11.1. The hardware and the software used are similar in both setups. Both mapping systems are visualized in RViz using flat squares to represent voxels.

Rosbags have been used to record the robot's odometry frames when it is required to drive the robot in the same path for both mapping approaches.

5.12.2 Visual comparison

Figure 5.11 displays a visual comparison between Octomap and G3DOMap. Both maps are built at 4 cm resolution and using a 3-meter maximum sensor radius. The two mapping methods are executed at their own time. G3DOMap took 12 minutes to reach the displayed state, while OctoMap consumed 25.5 minutes to reach the result with incomplete color information.

Figure 5.12 compares the two mapping approaches under similar operational conditions, developed by a robot running with same rosbag as the input. The series of snapshots displays OctoMap and G3DOMap generation derived at the same time stamp.

In Figure 5.13, it is displayed a set of 3D occupancy grid maps generated by G3DOMap and OctoMaps at different resolutions. The visual quality of maps is almost the same except for the white colour voxel chunks in OctoMap.

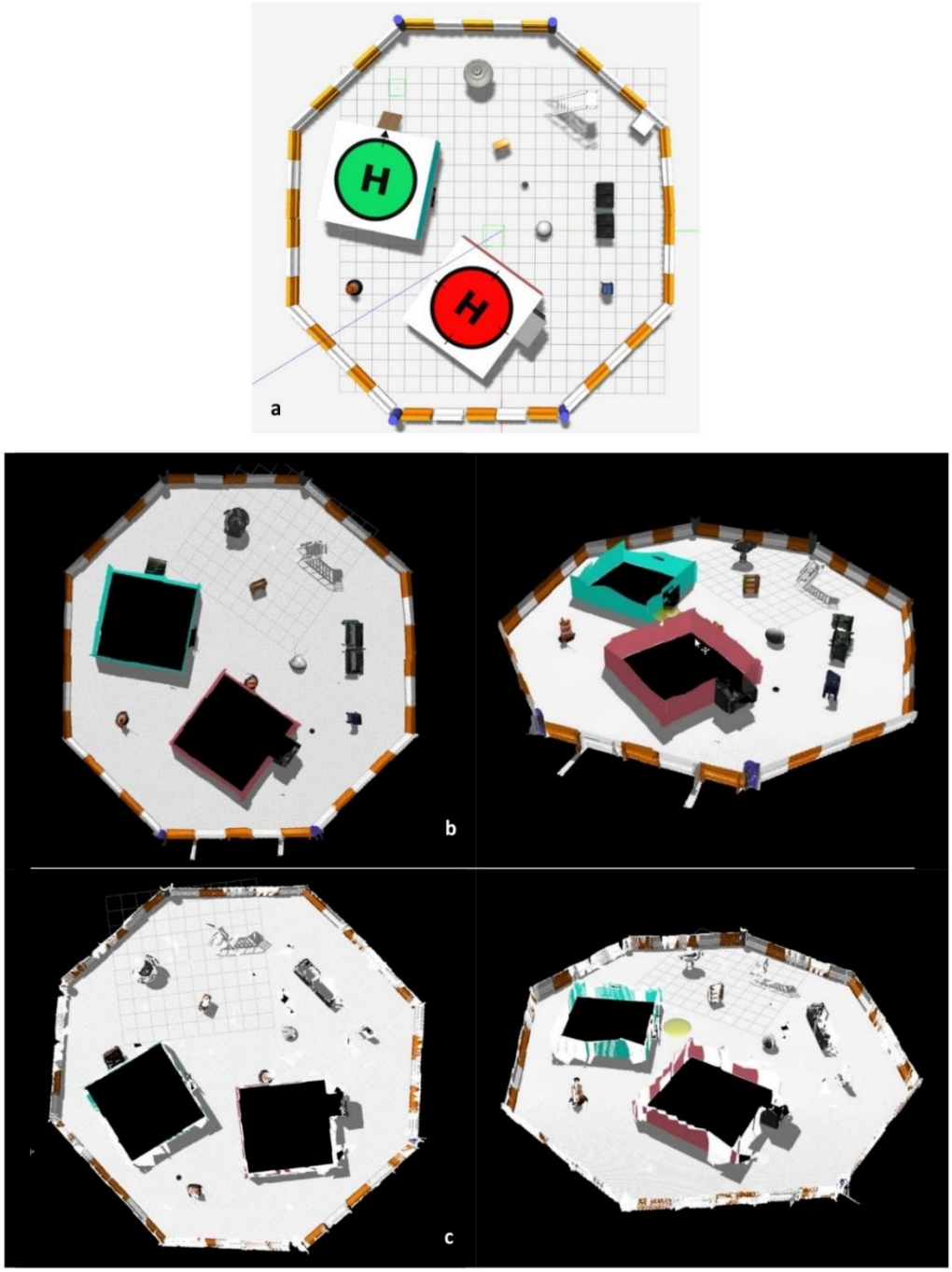


Figure 5.11: Full map comparison of G3DOMap and OctoMap (a) The simulated 10x10 environment used for testing. (b) the 3D occupancy grid generated by G3DOMap in 12:13 mins (c) the 3D occupancy grid map generated by OctoMap in 25:33 mins.

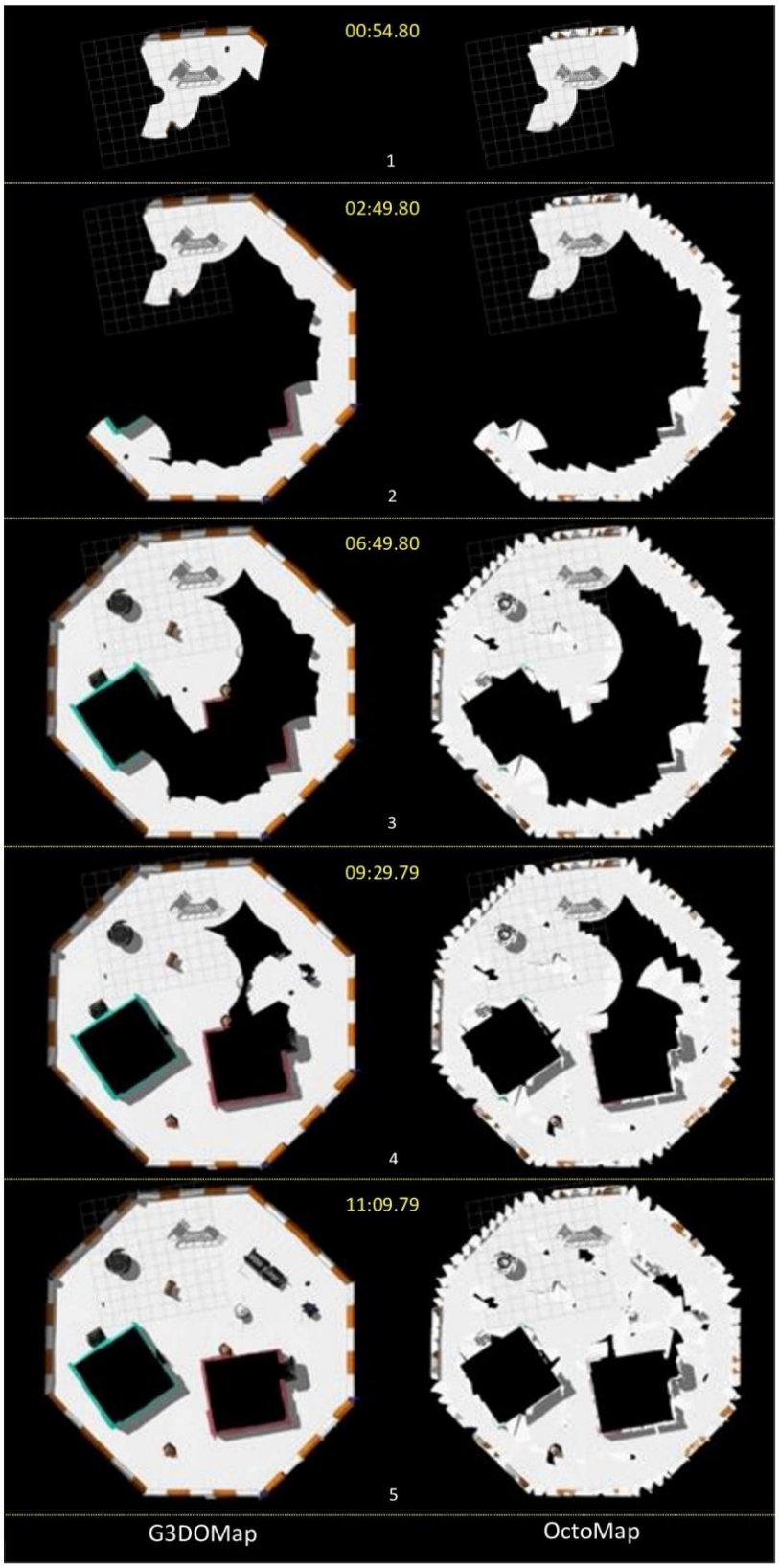


Figure 5.12 A snapshot series of the two mapping approaches when the robots are driven in the same path

Figure 5.14 displays a real-world map built with consecutive laser scans which are shifted from their actual position. The objects look distorted while they preserve the accurate colour information. The results highlight the requirement of a local map merging methodology where consecutive laser scans can be compared and merged based on their overlapping points. Since the colour information are almost accurate, it can be suggested to consider RBG values as a factor to merge point clouds locally.

It was also observed invalid object boundary colors in both mapping methods (Figure 5.15). The error is due to the sudden difference between endpoints in neighboring rays. However, unlike the other issues, this boundary identification [30] problem will not be further addressed in this thesis.

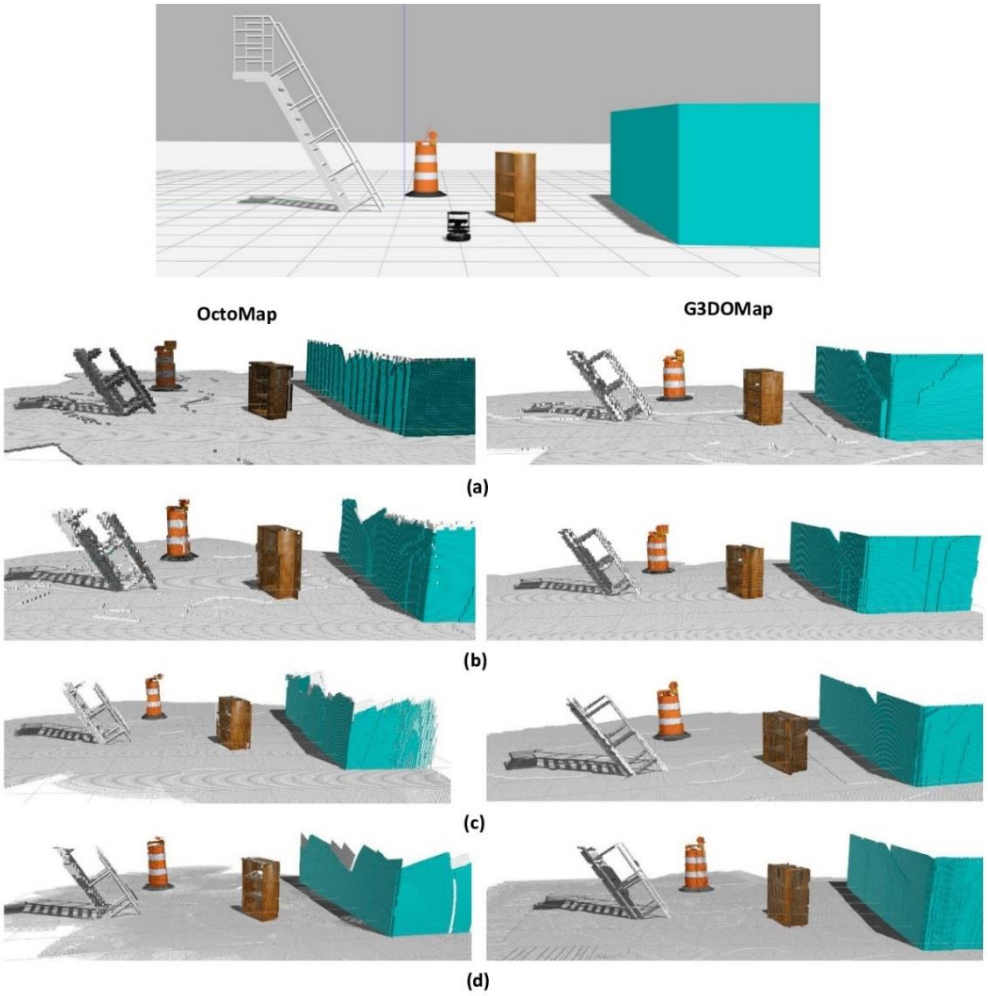


Figure 5.13: 3D occupancy grids of complex objects at different resolutions. (top) The simulated environment used for mapping. (a) 5 cm (b) 4 cm (c) 2 cm (d) 1 cm

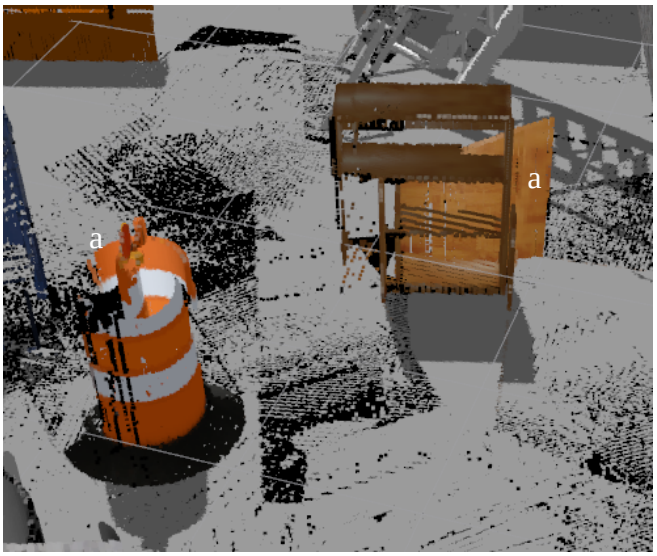
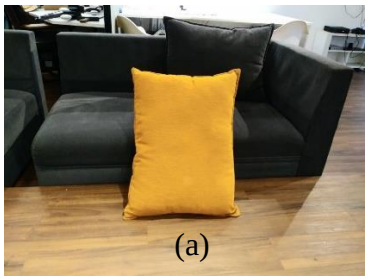


Figure 5.14: Object shift (a) observed in G3DOMap. The map resolution is 1 cm



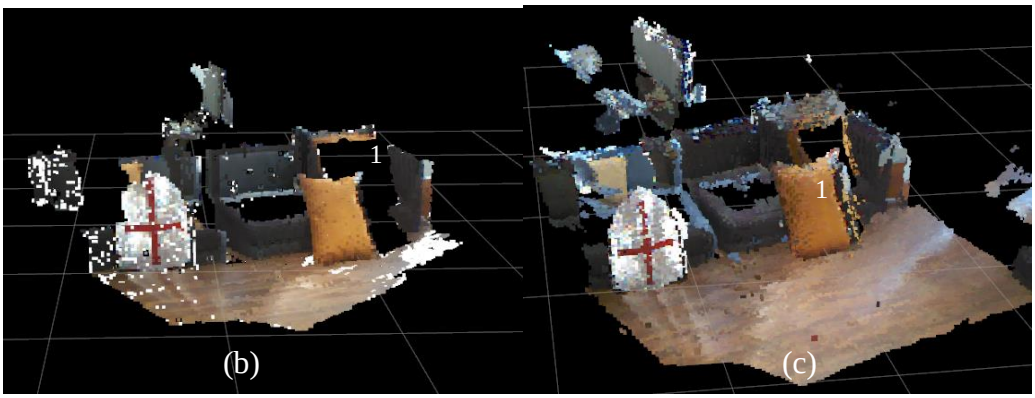


Figure 5.15: Invalid boundaries: (a) the real-world area of interest (b) OctoMap (c) G3DOMap. (1) indicates the invalid object boundaries identified by colors

5.12.3 Dynamic environment mapping comparison

In this experiment, another mobile robot was placed in the world, and it was moved across the mapping robot's visual range. Here the mapping robot was placed stationary to observe the response towards the dynamic object. Figure 5.16 indicates how

G3DOMap immediately removed the dynamic object from the map while OctoMap preserved the past voxels of the moving robot in the 3D map.

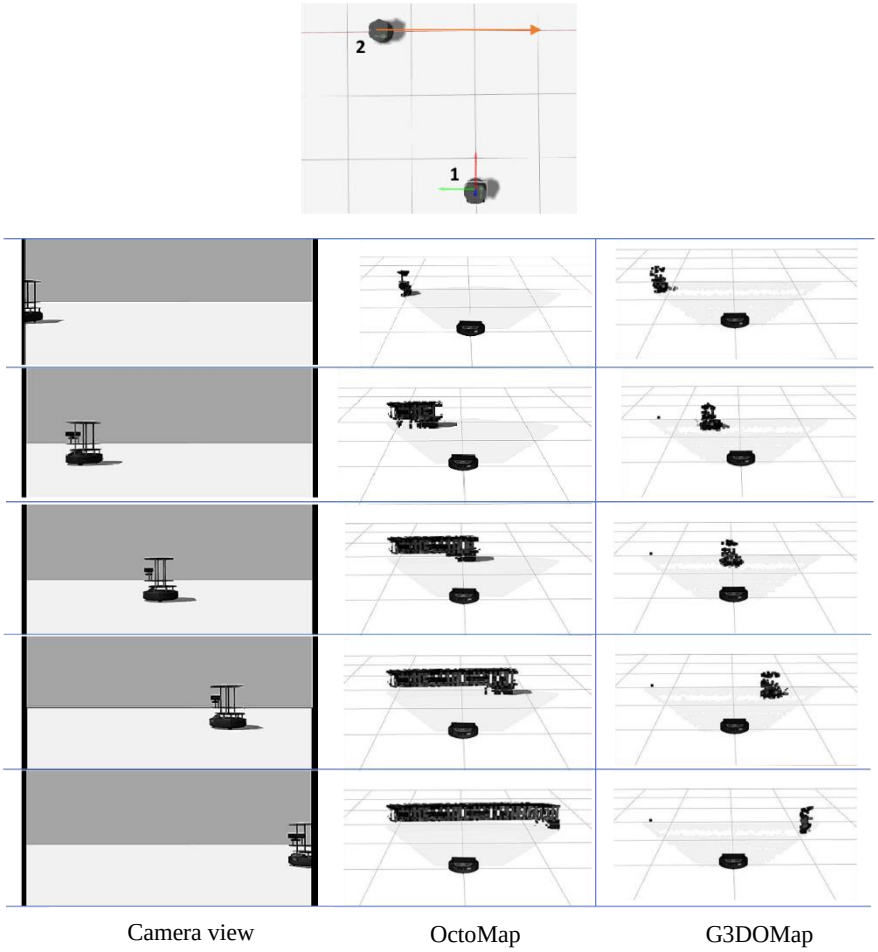


Figure 5.16: Dynamic environment responses of the two mapping techniques

5.12.4 Time comparison

Map building times for G3DOMap at different resolutions were measured and the results were compared against OctoMap. The results are indicated in Figure 5.17. The time measurements were taken for completely registering all the point clouds in the buffer.

5.13 Summary

This chapter presented the proposed high-performance computing-based 3D occupancy grid mapping system, G3DOMap, its system architecture, implementation

details and underlying data structures and algorithms. The system was compared with OctoMap qualitatively and quantitively. Now that the local maps of the robots are ready, the next step should be accurately merging those maps into a global frame. Chapter 6 presents the implementation details of direct map merging.

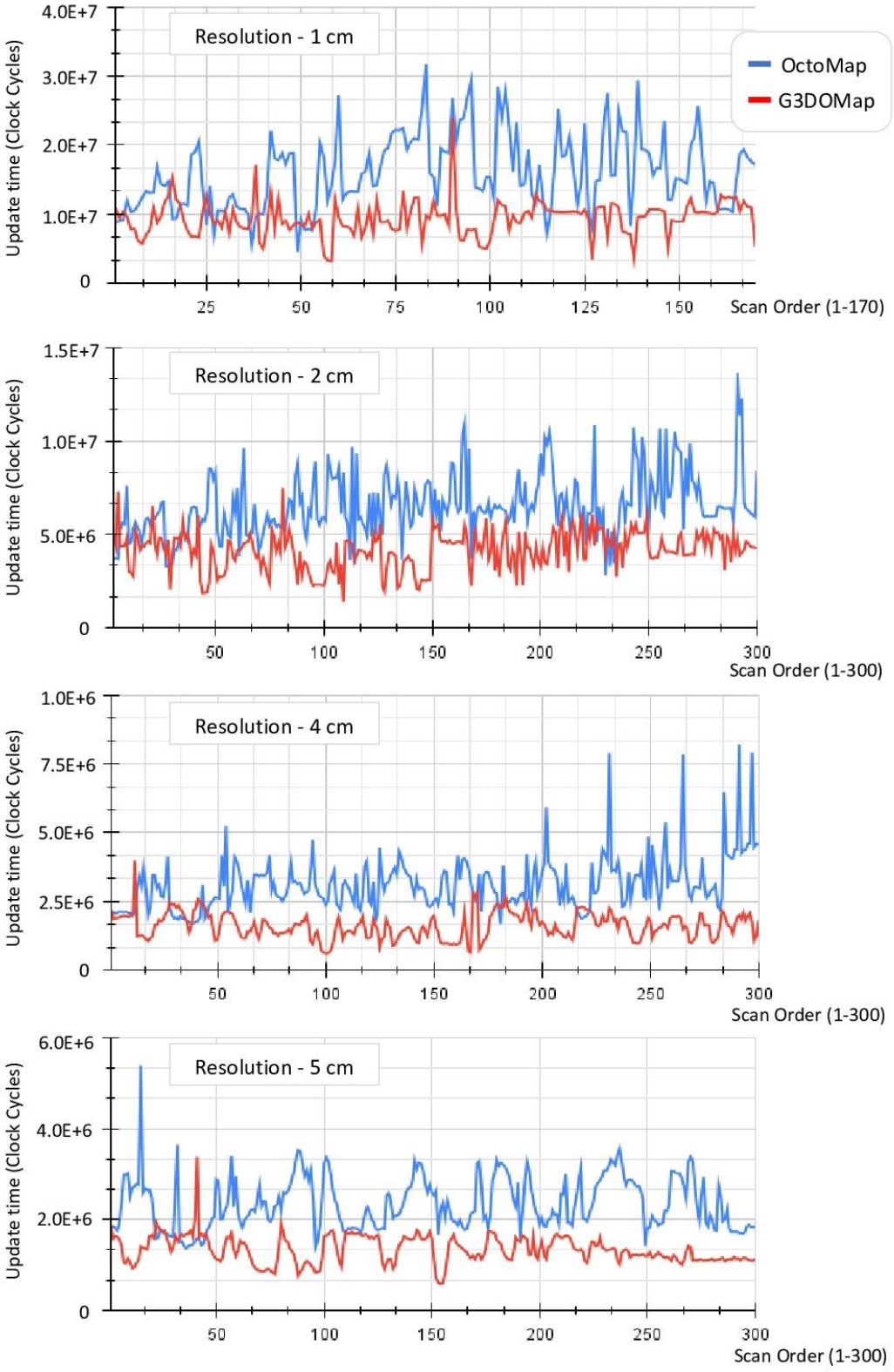


Figure 5.17: Point cloud registration times at different resolutions: G3DOMap Vs. OctoMap

6 3D MAP MERGING BY DIRECT METHOD

6.1 Introduction

In chapter 5, it was explained the novel process of building 3D occupancy grids with minimum latency. G3DOMap system was able to build 3D maps in real-time while keeping up with the robot's navigation speed and the camera frame rate.

The overall mapping environment can be covered faster by using a multi-robot system and merging those local maps into a global frame. This chapter discusses direct map merging when the initial poses of the robots are known.

It should be noted that direct map merging method makes several assumptions that could cause practical difficulties in real-life applications. However, as this research is focused on *high-performance* 3D map generation, direct map merging has been used as the base.

Direct map merging assumes that the robots' odometry data is absolutely accurate, visual sensors carry no errors, and the robots' initial relative poses are known. If motion data is accurate and the robot's initial pose is known, the current transformation of the new point cloud instance can be calculated. The new point cloud data can be either updated in a single central map or updated as localized maps and merged.

The direct map merging also can be the final step of a multi-robot 3D mapping system where pose estimation is already completed by a prior module.

The rest of the chapter will briefly discuss the system design and implementation procedure of direct map merging.

6.2 System design

Figure 6.1 illustrates the system architecture of the 3D map merging system. The multiple robots in the mapping system have their own local 3D occupancy maps aligned with their local coordinate frames. The base station can be a robot in the network or a dedicated stationary node. This base station subscribes to all 3D occupancy grids of the system, then expands them to separate octrees. Each robot pose with reference to the world coordinate frame is stored in the base station or determined by a pose estimation module.

At this stage of the architecture, the pose estimation module is considered as a black box.

Local map octrees are transformed to meet the world coordinates of the system and added into the merged octree. The merged tree is periodically compressed and the final occupancy grid map is published.

Direct map merging module is implemented using ROS framework, C++, and Octomap C++ library. The system is designed to publish the global map even if some robots are out of range. The map merger can run inside every robot, or one chosen robot.

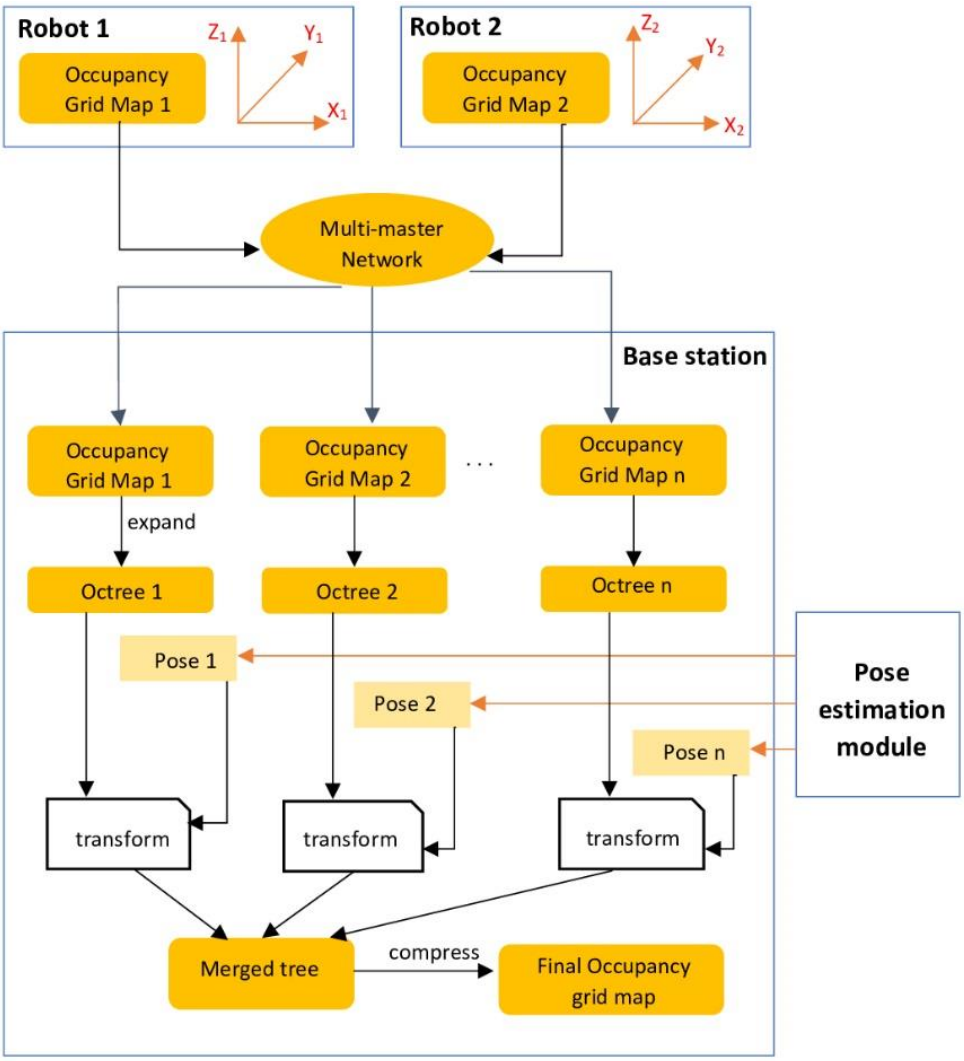


Figure 6.1: System architecture of 3D map merging

6.2.1 Results

Figure 6.2 shows the results of map merging with the known initial poses. The maps are generated on a real-world environment using Quanser Qbot2 platform. The processing was performed on a NVIDIA Jetson TX2 mobile processor. The maps were merged and visualized on a base node, which was connected to the same multi-master network.

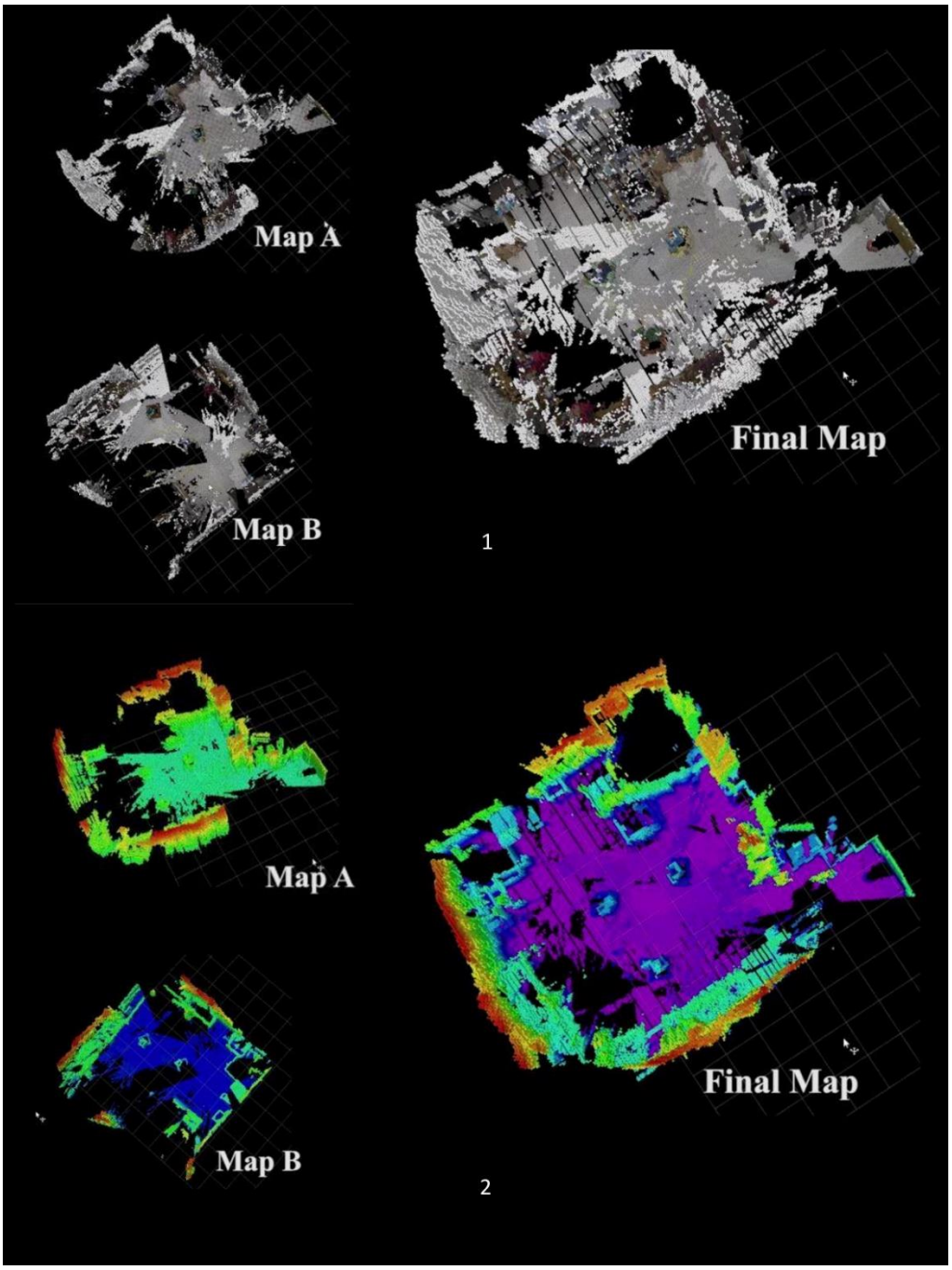


Figure 6.2: Maps merged with known initial poses. 1) real world colored (RGB) OctoMaps 2) height based colored OctoMaps: red being the highest, and purple being the lowest height

6.2.2 System errors

In the direct map merging system, there are two types of errors identified.

The first type of error is local shifts, indicated as (a) in Figure 6.3. They occur due to erroneous odometry sensor data, hence they are the hardest to correct. Local map

errors are propagated and added to the global maps when they are merged. Fully accurate wheel encoders and gyroscopes cannot be found, and these errors should be compensated using software methods.

The second error type is mapping inconsistencies introduced at map merging level. If the same object is seen at different perspective by multiple robots/views, the final map is affected by inconsistent representations. See (b), (c), (d) in Figure 6.3. Proper global map merging algorithms can minimize this error.

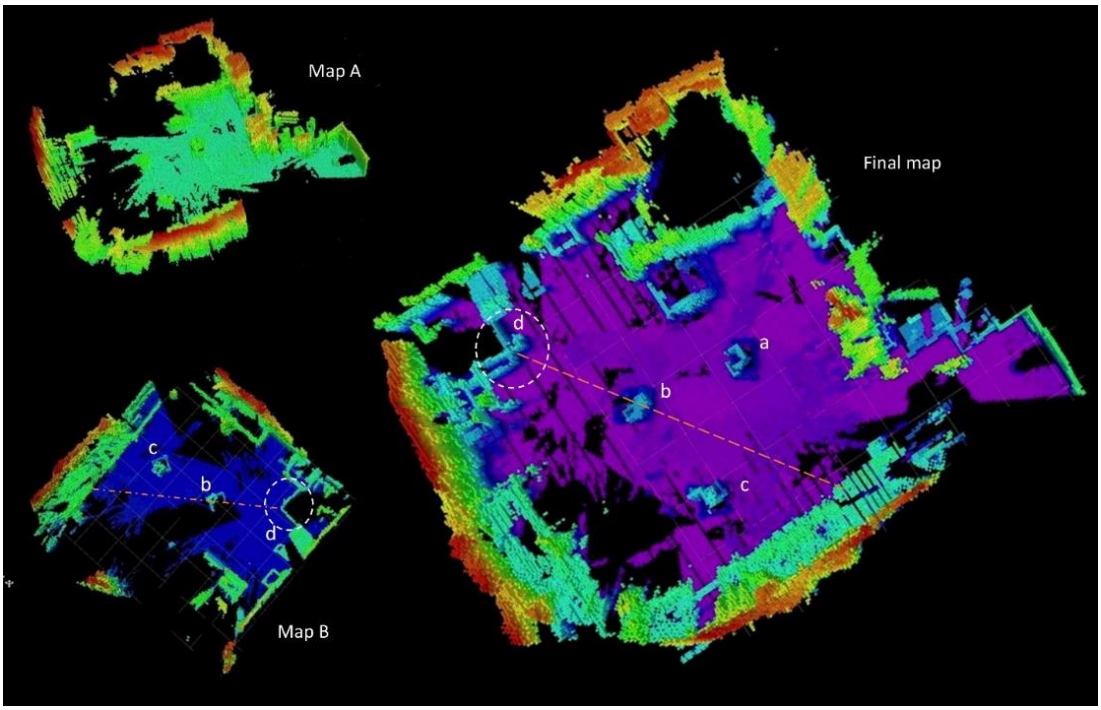


Figure 6.3: Identified errors in direct map merging

6.3 3D map merging using erroneous odometry sensor data

Analyzing the results in sections Visual comparison5.12.2 and 6.2.2, it is clear that any kind of map generating and map mering system can contain visual errors which need to be rectified before being added into the final map. The approach proposed in this section provides solutions for two types of errors. The object shifts and different object perspectives.

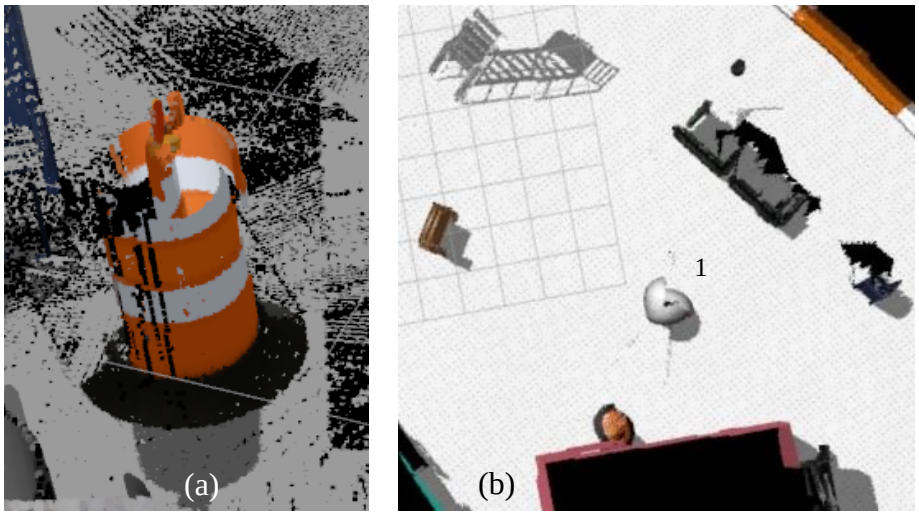


Figure 6.4: Errors observed in 3D maps: (a) object shift, (b) three mapping perspectives of sphere 1, caused by nonconsecutive frames

6.3.1 Object shifts

The mapped object appears stretched or deformed in a local map. The traffic cone in Figure 6.4 (left) has been mapped in one go. The robot was moved around it within the shortest possible time. Due to the small errors between consecutive odometry readings, the consecutive map frames are affected. An error in a single odometry frame is linear, but it is propagated throughout the system if not corrected. Not only that, these slight odometry errors are newly generated with every frame. Equation (30) presents the total odometry error at the end of n^{th} frame, E_n, where e_n is the linear error generated by the n^{th} frame.

$$E_n = e_1 + e_2 + e_3 + \cdots + e_n \tag{30}$$

6.3.2 Different object perspectives

As seen in the Figure 6.4 (right), the sphere has been represented as a combination of three different mapping perspectives. This commonly occurs in global maps after merging a few 3D maps. If the sphere was mapped using consecutive frames as in the traffic cone, the error would appear as a stretched object. In this scenario, the robot has arrived in the area three times and the object is mapped with three nonconsecutive sets of Kinect frames. There can be three reasons for different object perspective error.

Odometry error propagation, erroneous relative poses between multiple robots, and the representation of mapping angle in occupancy grids.

The third reason, occupancy grid representation changes due to mapping angle is displayed in Figure 6.5. Based on the voxel arrangement, the voxel center coordinates can be different in each map. Hence it can have a small impact on the merged map. But if the map alignment is accurate, and the map resolution is high, the basic object shape can be preserved.

6.3.3 Overcoming mapping errors

Looking at the nature of local mapping errors, they should be overcome by comparing the map frames before adding them into the final map. Once the errors are added to the map, it would be extremely hard to correct them automatically.

ICP algorithm is the proposed approach for rectifying erroneous odometry sensor data. Similar results have been achieved in the literature by using NDT (Normal Distribution Transform) [82] algorithm. It is frequently seen that researchers use NDT for rough alignments and ICP for fine alignments [83].

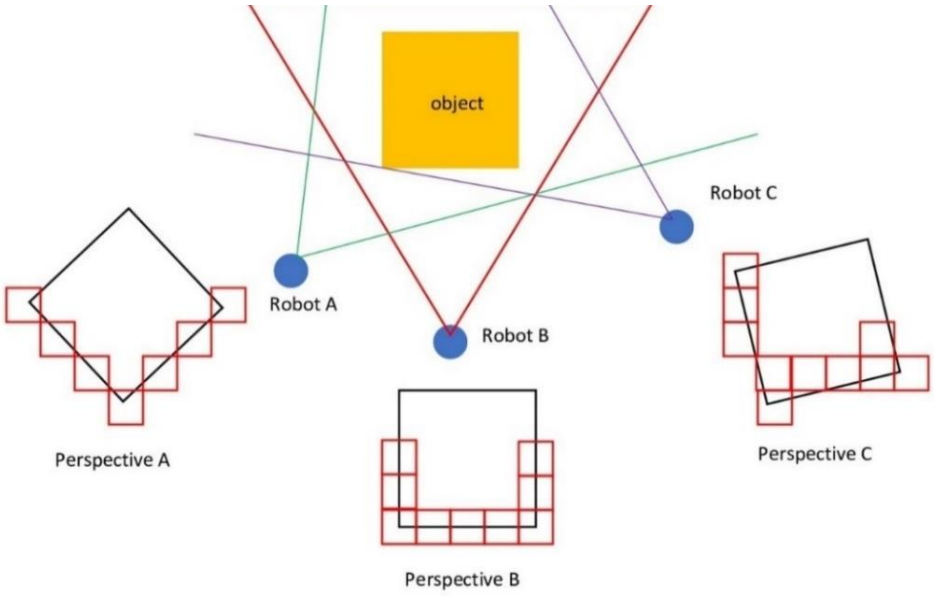


Figure 6.5: Different object perspectives in occupancy grid maps

6.4 Summary

This chapter briefly presented the design details of direct map merging. Unresolved odometry related mapping errors and different object perspective errors introduced by direct map merging were observed. It is proposed to use ICP algorithm to resolve odometry errors in local frames and consistency issues in global maps.

Global map merging requires a robust inter-robot communication network to exchange local maps. The upcoming chapter will discuss the inter-root communication system and the integration procedure of multimaster-fkie ROS package.

7 INTER ROBOT COMMUNICATION

7.1 Introduction

Chapter 6 discussed 3D map merging based on known initial pose information. In order to exchange the local maps for global map merging, the multi-robot system must have an active inter-robot communication network.

Maintaining interaction among autonomous robot units is challenging in the given scenario due to a few reasons.

- The extent of the environment is unknown
- The robots are designed to work independently, hence their relative positions at a given time can be out of the direct communication range
- The robot movements in the multi-robot system are uncoordinated
- When the robots meet each other after a long time, their 3D maps have grown large
- Robots may fail to operate in conditions such as low battery and accidents

Despite all the challenges mentioned above, there is one advantage in a multi-robot system. The number of robots in the network is limited so they can be easily identified by each other.

The METHODOLOGY chapter studied the inter-robot communication system requirements and existing ROS compatible protocols. The Multimaster-fkie [72] ROS package was selected as the inter-robot communication mechanism. This chapter will present how Multimaster-fkie package is integrated into the multi-robot system prototype.

7.2 Multimaster-fkie package

7.2.1 Introduction

Multimaster-fkie architecture is designed for exchanging information between ROS subsystems. The sub system can either be a single robot with multiple nodes or a local network of several robots. The information can be exchanged in the forms of topics,

services and actions.

Each subsystem should have a roscore setup, so it can act as a master. When topics, services and actions are registered on the multi-master system, a point-to-point socket is created to directly connect two or more nodes.

Multimaster_fkie has two ROS nodes: *master_dicovery* node and *master_sync* node.

Master_discovery node sends periodic multicast messages to the network. It makes the system aware of the own sub-system and keeps track of other sub-systems. If the master_discovery node identifies any change in the local sub-system, it notifies other masters in the network.

Master_sync node registers remote topics and services advertised by external master_discovery nodes to its roscore. Rather than synchronizing every component in the network, the user can define the topics and services to be synchronized. This can reduce the bandwidth usage.

7.2.2 Integration to the system

This section briefly describes how multimaster-fkie ROS package was integrated into the multi-robot 3D mapping system.

There is no limit to the number of robots in the multi-master network. When using multimaster-fkie, the robots must know the network configuration details of the other agents. The steps in Table 7.1 have been followed to successfully configure the multimaster-fkie network. All the agents are connected to the same network.

The same procedure can be followed to access data of complex ROS topics such as point clouds, maps, and navigation stack.

the subscription rate of multimaster-fkie to a large topic like OctoMap was sufficient

Table 7.1: Steps to configure Multimaster-fkie into the system

	Process	Details
Step 1	Configuring network IPs	• Configuring fixed IP addresses for all robots • Including all fixed IP addresses in /etc/hosts file with relevant host names • Can be tested using ping command
Step 2	Exporting ROS master URI	• Export self ROS master URI and the master port • Can use bash files for this
Step 3	Enabling multicast feature	• Enable multicasting in each robot • After pinging to the multicast group IP, the sender should receive responses from all robots
Step 4	Running Multimaster-fkie	• Start roscore in each robot • Run master-discovery node, together with multicast group IP • Launch master-sync nodes
Step 5	Publishing and subscribing data within the network	• Start other functional processes of the system to publish data into topics • All topics published within the network can be subscribed by any node.

7.3 Summary

This chapter included how multimaster-fkie, the existing multi-master ROS package was integrated into the system. The multi-master communication paradigm was selected from four available ROS-based communication methods because it allows event-triggered communication, distributed network architecture and fault tolerance.

At this point, GPU accelerated 3D map building, 3D map merging, and inter-robot communication systems are discussed. Chapter 8 is dedicated for the remaining sub system, autonomous exploration in unknown even-surfaced environments.

8 AUTONOMOUS EXPLORATION IN UNKNOWN EVEN-SURFACED ENVIRONMENTS

8.1 Introduction

Autonomous exploration is the last sub-system of the multi-robot system prototype. This subsystem aims to explore the environment and expand the knowledge using globally consistent submaps.

The hardware prototype used for proof of concept in this research is ground operated and wheeled. At this point, the navigation is assumed to be taken place in an even-surfaced environment. The system's primary navigation goal is to move the robots all over the 2D plane to achieve maximum coverage. The speed of exploration and area division among multiple robots are not the concerns at this stage. The focus remains on exploring the entire environment while generating the 3D maps.

As suggested in the methodology, a frontier-based exploration strategy is used for the even-surfaced environment exploration. Explore-lite ROS package has been selected as the path planner due to its greedy, grid-based strategy. This section presents how explore-lite is integrated into the overall system.

8.2 Explore-lite as the navigation planner

Explore-lite is a ROS based package developed for lightweight path planning. the lightweight comes from not generating its own 2D costmap and running on top of ROS navigation stack. Instead of creating an own costmap, it subscribes to nav_msgs/OccupancyGrid ROS messages to receive the basic map.

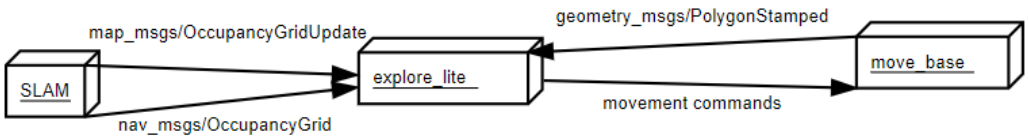


Figure 8.1: ROS message passing in explore-lite. Source: [72]

Explore-lite subscribes to map related messages published by any mapping node. It uses these maps to create a frontier map and send movement commands to ROS navigation package (ex: move_base).

Explore-lite is known as a greedy exploration algorithm. It identifies all the frontiers in the existing map and calculates the cost to navigate to each of them. After choosing the frontier with the least cost, the remaining frontier nodes are added into a stack.

Figure 8.2 shows a screenshot of running explore-lite in the indoor experimental setup.

8.3 Integration to the system and results

For the proof of concept, OctoMap has been used as the mapping framework.

OctoMap publishes 2D costmap of the navigation plane as nav_msgs/OccupancyGrid messages. The costmaps are derived from the 3D occupancy grid which is indicated in Figure 8.3 (left). An example 2D costmap published by G3DOMap is presented in Figure 8.3 (right). It is published directly from the G3DOMap's ground layer in the original resolution, which is excessive for a Qbot2.

Extract from the base prototype [1], ROS navigation package has been used as the move-base planner to execute movement commands. The system runs on NVIDIA Jetson TX2. A real-world indoor environment is used for mapping, and Quanser QBot2 is used as the moving robot. The series of snapshots in Figure 8.4 shows how the frontier-based exploration was carried on by the robot. It was observed that the system achieved full coverage of the room in an acceptable latency.

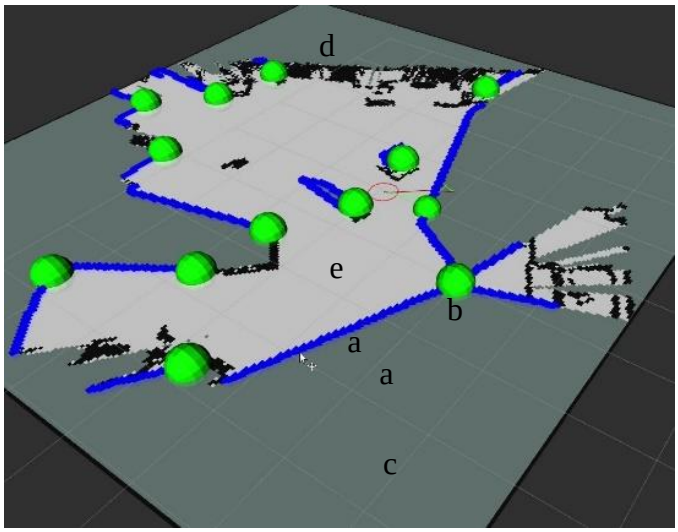
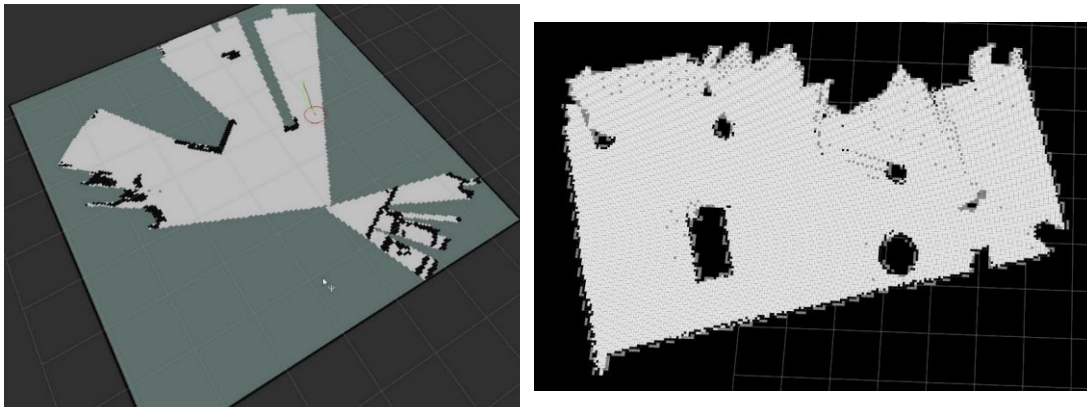


Figure 8.2: A snapshot from explore-lite navigation: (a) frontiers marked as blue lines. (b) goal points marked by green spheres. The sizes of the spheres vary based on the navigation cost. (c) Unknown environment, (d) known obstacles marked in black, (e) observed free environment



(a) (b)

Figure 8.3: CostMaps published for navigation (a) An example 2D costmap published by OctoMap. (b) An example 2D costmap generated by G3DOMap. The resolution here is too high for a Qbot2

8.4 Summary

The chapter discussed how frontier-based explore-lite ROS package was integrated into the overall system to achieve fast coverage.

This concludes all the methodology related chapters of the thesis. The next chapter will present the conclusion and discuss possible future work.

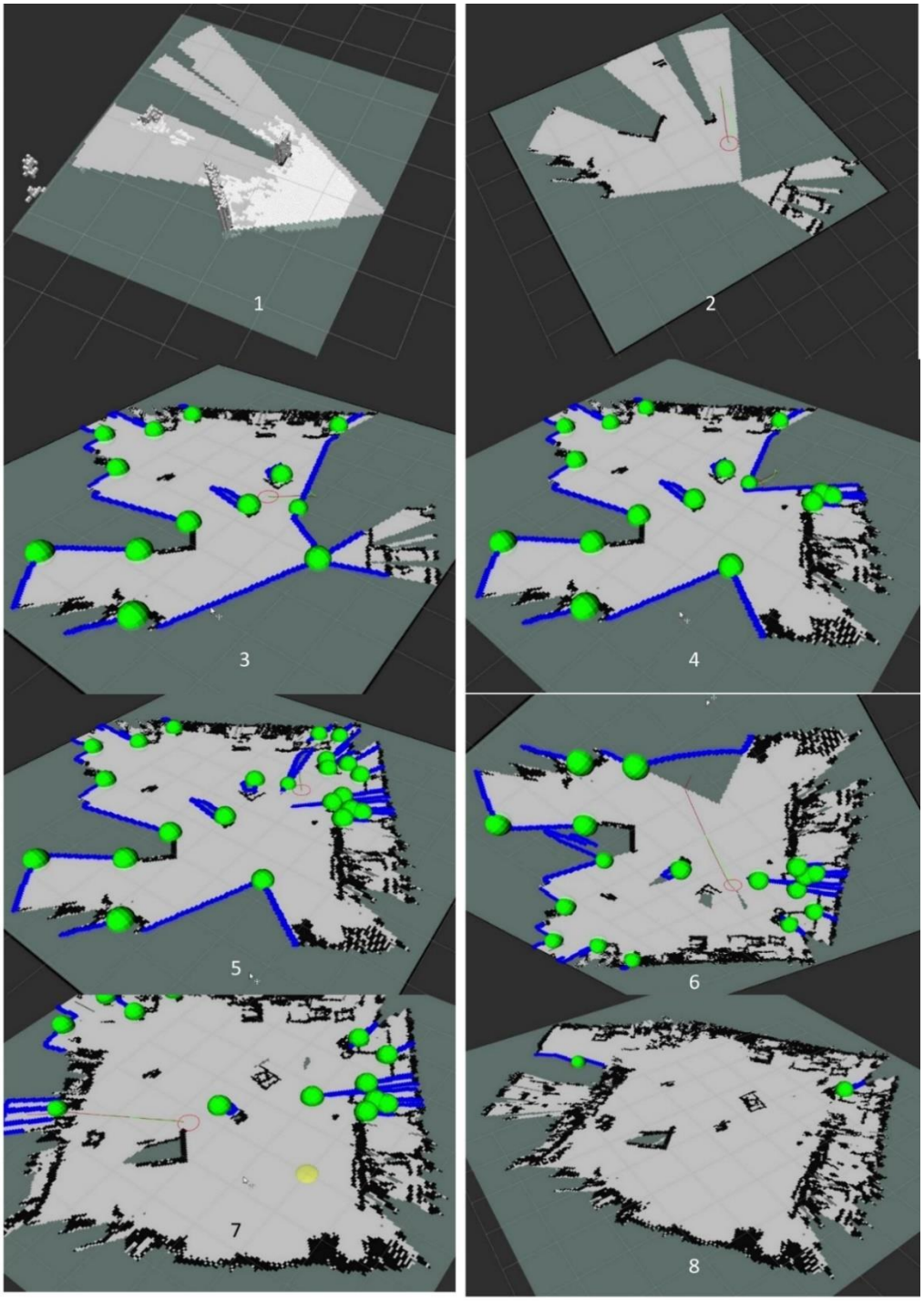


Figure 8.4: A series of snapshots of a single robot exploring an office room using explore-lite

9 CONCLUSION

9.1 Introduction

The work presented in this thesis is a combination of four major subsystems: GPU accelerated 3D map building, 3D map merging, inter-robot communication, and autonomous exploration. The 3D occupancy grid map acts as the central component of the system. Hence the focus of the thesis lies in addressing challenges of performance enhancement of 3D occupancy map building.

G3DOMap is the main contribution of this work. It is a fast, SIMD architecture-based 3D occupancy grid mapping system for mobile robots. Additionally, a real-world multi-robot 3D mapping and exploration system for unknown and even-surfaced environment was developed.

The system is platform independent. Any platform which provides RGBD point clouds and IMU data as inputs can be configured to use this system. Therefore, this system can be extended to use any type of environment using different robot types.

9.2 Summary of results

9.2.1 G3DOMap

Occupancy grid maps are a suitable mapping technique for autonomous mobile robot 3D mapping of unknown environments. Some reasons for this are discrete space representation, resolution management to meet memory requirements, derivation of navigation maps for robots of different sizes, compressibility, and sufficiency for both humans and machines.

OctoMap is the state-of-the-art mobile robot 3D mapping system that fulfills all the above requirements. But it displays poor time performance for building large or high-resolution maps. OctoMap's point cloud registration is developed as a single-threaded application, which is the bottleneck of the system leading to buffer overflows and missing color information. G3DOMap is developed as a solution to this problem. It addresses the 3D occupancy grid mapping process as a parallel computing approach.

CUDA algorithms are introduced for parallelized processes in G3DOMap: spatial point transformation, ray tracing, and 3D voxel generation. The voxels are represented

as nodes to be inserted or updated in the map, where node keys are 64-bit M-codes, and values are 32-bit RGBO-codes. The central map is located inside the CPU and is stored as a hash table. Replacing spatial octree with a hash table and using bit-level operations for node update have enhanced the time performance in CPU-based operations in G3DOMap. The node generation and map update process achieved over 2 times performance over the traditional octree data structure.

The features of G3DOMap have been compared with OctoMap based on quantitative measures (time) and qualitative measures (visual quality of maps). G3DOMap shows an average 1.92 times enhancement in point cloud registration time. The point cloud registration process spans from capturing RGBD frames to integrating new voxels into the map data structure. In visual aspects, G3DOMap shows better performance in map completion, full color registration, and dynamic object removal from the map.

The maps are complete because buffer overflows rarely occur due to high computational speed. GPU processes point clouds parallelly and match the image capturing rate of the depth camera. As the system registers voxel spatial information and color information in one go, G3DOMap does not display colorless(white) voxels in the colored map.

The reason for the faster response towards dynamic objects is; G3DOMap updates the occupancy using a computationally simpler algorithm than Elfes [8]. It considers occupancy as a discrete integer level value with preset upper and lower bounds, which is much suitable for a rapidly changing environment.

G3DOMap can be used in two forms. Static G3DOMap is faster and more applicable in static environments, manually controlled robots and object modeling applications. Dynamic G3DOMap is slower than the static mode, yet twice faster than OctoMap. Ray tracing at higher resolutions introduces a higher computational load. The system was able to successfully map a simulated environment at 1cm resolution in real time.

Even though the GPU has been utilized to enhance the 3D occupancy grid map generation, the performance results mentioned in Table 4.2 seems highly impossible.

There are few reasons for not achieving those theoretical numbers within a real system.

G3DOMap time performance can be further enhanced if the map was located in the GPU. Currently the application wastes a lot of computational time pushing the newly generated voxels from device to host (GPU to CPU). The data copying time has a significant impact on the overall process.

However, further time enhancement is blocked by ROS platform. ROS entirely runs on CPU, so the final map must be sent into the CPU before a new map publish. Unless an algorithmic way of efficiently doing this is suggested, the performance enhancement will stop at the point of central map update. This is not a platform enhancement but an algorithmic approach, hence may not be impossible to achieve.

9.2.2 3D map merging

For 3D map merging, a direct approach is taken assuming that only the direct odometry measures and Kinect point clouds are sufficient to build an accurate 3D map. The proof of concept is built using OctoMap, where the system merges two 3D occupancy grid maps obtained from wirelessly connected robots. Using direct odometry measures and using mechanically obtained initial poses led to two errors: object shifts, and deformations due to different robot perspectives. These are due to the absence of measures to rectify local and global mapping errors.

It is suggested to use ICP algorithm to reduce sensor-generated mapping errors.

9.2.3 Inter-robot communication

Multmaster-fkie communication system is used to interconnect the robots to share the maps. Multimaster-fkie fulfills all current system networking requirements. A robot can connect to the network non-continuously and non-periodically by notifying its presence. As the system is built with multiple master nodes, single point of failure is prevented. Multimaster-fkie provides the option to choose the ROS topics to be synchronized, so that it can reduce the bandwidth requirement. The package is also suitable for transmitting compact 3D occupancy grid map messages, which are comparatively large ROS messages. Large OctoMap messages have been synchronized between robots to establish the proof of concept.

9.2.4 Autonomous exploration in unknown even-surfaced environment

The system uses *explore-lite* third-party ROS package with the intention of maximizing the explored area while 3D mapping. It autonomously and greedily explores the environment and achieves a full coverage across navigable frontiers. The sub system was tested in a real-world, indoor environment.

9.3 Future work

For the future work of the research, a few implementation processes can be suggested.

The first improvement should be the implementation of a GPU based central map. Currently G3DOMap is not achieving the best time performance due to the high memory exchange between device and host. This overhead can be largely reduced by placing the central map inside the graphics unit and sending the built map to CPU on demand. Since the map is built as a hash table, the challenge would be to concurrently update nodes without race conditions.

The second implementation should be minimizing the odometry errors in point cloud registration. This can be achieved by using ICP algorithm and current odometry transformation as the initial alignment. Map merging time and local drift errors can also be minimized by creating smaller sub-map segments and combining them [11].

In the proof of concept, the robot's autonomous navigation is based on the cost map published by OctoMap. As a future step for the exploration, the system should derive the ground map from G3DOMap and publish the cost map considering the robot's physical dimensions.

REFERENCES

[1] K. T. D. S. De Silva, J. I. Chinthaka, B. P. A. Cooray, P. P. Kumara and S. J. Sooriyaarachchi, "Multi-robot 3D Exploration of Uneven Terrain, Bachelor Thesis," University Of Moratuwa, Sri Lanka, 2018.

[2] E. Howell, "Curiosity Rover on Mars: Facts and Information," 18 July 2018. [Online]. Available: https://www.space.com/17963-mars-curiosity.html. [Accessed 23 07 2021].

[3] L. Stahl, "Robots come to the rescue after Fukushima Daiichi nuclear disaster," 28 July 2019. [Online]. Available: https://www.cbsnews.com/news/robots-come-to-the-rescue-after-fukushima-daiichi-nuclear-disaster-60-minutes-2019-07-28/. [Accessed 23 07 2021].

[4] iStock, "Robotic Arm Taking A Cardboard Box In The Warehouse stock," [Online]. Available: https://www.istockphoto.com/photo/robotic-arm-taking-a-cardboard-box-in-the-warehouse-gm1273518964-375369603. [Accessed 23 07 2021].

[5] D. Leprince-Ringuet, "For self-driving cars, winter is coming," 05 March 2020. [Online]. Available: https://www.zdnet.com/article/for-self-driving-cars-winter-is-coming/. [Accessed 23 07 2021].

[6] R. Arora, "Autonomous Vehicles: The Mix of Opportunity and Uncertainty," 06 May 2019. [Online]. Available: https://www.fierceelectronics.com/components/autonomous-vehicles-mix-opportunity-and-uncertainty. [Accessed 23 07 2021].

[7] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss and W. Burgard, "A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard," *Auton. Robots*, p. 189–206, 2013.

[8] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46-57, 1989.

[9] J. Ryde and H. Hu, "3D mapping with multi-resolution occupied voxel lists.," *Auton Robot* , 2010.

[10] I. Dryanovski, W. Morris and J. Xiao, "Multi-Volume Occupancy Grids: an Efficient Probabilistic 3D Mapping Model for Micro Aerial Vehicles," *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1553-1559, 2010.

[11] P. Schmuck, S. A. Scherer and A. Zell, "Hybrid Metric-Topological 3D Occupancy Grid Maps for Large-scale Mapping," *IFAC-PapersOnLine* , no. 49, p. 230–235, 2016.

[12] J. Poppinga, M. Pfingsthorn, S. Schwertfeger, K. Pathak and A. Birk, "Optimized Octtree Datastructure and Access Methods for 3D Mapping," in *2007 IEEE International Workshop on Safety, Security and Rescue Robotics*, 2007.

[13] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *IEEE International Conference on Robotics & Automation (ICRA)*, St. Louis. New York, 1985.

[14] M. Yguel, O. Aycard and C. Laugier, "Update policy of dense maps: Efficient algorithms and sparse representation," in *International Conference Field and Service Robotics*, 2007.

[15] Y. Li and Y. Ruichek, "Building variable resolution occupancy grid map from stereoscopic system — A quadtree based approach," in *2013 IEEE Intelligent Vehicles Symposium*, 2013.

[16] J. Saarinen, H. Andreasson, T. Stoyanov and A. Lilienthal, "3D normal distributions transform occupancy maps: An efficient representation for

mapping in dynamic environments," *The International Journal of Robotics Research,* vol. 32, no. 14, pp. 1627-1644, 2013.

[17] D. Fridovich-Keil, E. Nelson and A. Zakhor, "AtomMap: A probabilistic amorphous 3D map representation for robotics and surface reconstruction," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[18] J. Hörner, Automatic Point Clouds Merging, Univerzita Karlova, Matematicko-fyzikální fakulta, 2018.

[19] A. Nuchter, H. Surmann, K. Lingemann, J. Hertzberg and S. Thrun, "6D SLAM with an application in autonomous mine mapping," in *IEEE International Conference on Robotics and Automation*, 2014.

[20] F. Rovira-Más, Q. Zhang and J. F. Reid, "Stereo vision three-dimensional terrain maps for precision agriculture," *Computers and Electronics in Agriculture,* vol. 60, no. 2, pp. 133-143, 2008.

[21] R. Valencia, E. H. Teniente, E. Trulls and J. Andrade-Cetto, "3D mapping for urban service robots," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[22] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," *Sensor Fusion IV: Control Paradigms and Data Structures*, 1992.

[23] D. Huber, O. Carmichael and M. Hebert, "3D map reconstruction from range data," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation*, 2000.

[24] S. Thrun, W. Burgard and D. Fox, "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation.*, 2000.

[25] C. Früh and A. Zakhor, "Constructing 3D City Models by Merging Aerial and Ground Views," *3D Reconstruction and Visualization*, Vols. November-Decmber 2003, pp. 52-61, 2003.

[26] A. Howard, D. F. Wolf and G. S. Sukhatme, "Towards 3D mapping in large urban environments," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[27] D. Wolf, A. Howard and G. S. Sukhatme, "Towards geometric 3D mapping of outdoor environments using mobile robots," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[28] J. Xiao, B. Adler, J. Zhang and H. Zhang, " Planar segment based three-dimensional point cloud registration in outdoor environments," *Journal of Field Robotics*, p. 1–31, 2013.

[29] K. Zhou, M. Gong, X. Huang and B. Guo, "Data-Parallel Octrees for Surface Reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 5, pp. 669-681, 2011.

[30] K. Pirker, M. Rüther, H. Bischof and G. Schweighofer, "Fast and accurate environment modeling using three-dimensional occupancy grids," in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 2011.

[31] C. Crassin, GigaVoxels: a Voxel-Based Rendering Pipeline for Efficient Exploration of large and detailed Scenes, UNIVERSITÉ DE GRENOBLE, 2011.

[32] M. Zeng, F. Zhao, J. Zheng and X. Liu, "Octree-based fusion for realtime 3D reconstruction," *Graphical Models*, 2012.

[33] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges and A. Fitzgibbon, "KinectFusion: Real-time

dense surface mapping and tracking," in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, 2011.

[34] M. Nießner, M. Zollhofer, S. Izadi and M. Stamminger, "Real-time 3D Reconstruction at Scale Using Voxel Hashing," *ACM Transactions on graphics* , vol. 32, no. 6, pp. 1-11, Nov. 2013.

[35] B. curless and M. Levoy, "A volumetric method for building complex models from range images," *Proc. Computer graphics and interactive techniques, ACM*, p. 303–312, 1996.

[36] S. Kleinschmidt and B. Wagner, "GPU-accelerated Multi-sensor 3D Mapping for Remote Control of Mobile Robots using Virtual Reality," in *Proceedings of the 13th International Conference on Informatics in Control, Automation and Robotics*, 2016.

[37] A. Dai, M. Nießner, M. Zollofer, S. Izadi and C. Theobalt, "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface re-integration," *ACM Transactions on Graphics 2017 (TOG)*, 2017.

[38] S. Agarwa, K. Mierle and e. al., "Ceres Solver," 2010. [Online]. Available: http://ceres-solver.org. [Accessed 24 07 2021].

[39] W. Dong, J. Shi, W. Tang, X. Wang and H. Zha, "An Efficient Volumetric Mesh Representation for Real-Time Scene Reconstruction Using Spatial Hashing," in *018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[40] W. Dong, J. Park, Y. Yang and M. Kaess, "GPU Accelerated Robust Scene Reconstruction," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[41] O. Kähler, V. A. Prisacariu, C. Y. Ren, X. Sun, P. Torr and D. Murray, "Very High Frame Rate Volumetric Integration of Depth Images on Mobile Devices,"

IEEE Transactions on Visualization and Computer Graphics, vol. 21, no. 11, pp. 1241-1250, 2015.

[42] J. Wang, "On sign-board based inter-robot communication in distributed robotic systems," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, 1994.

[43] B. Yamauchi, "Frontier-based exploration using multiple robots," in *Proceedings of the second international conference on Autonomous agents*, 1998.

[44] R. Alami, S. Fleury, M. Herrb, F. Ingrand and F. Robert, "Multi-robot cooperation in the MARTHA project," *IEEE Robotics & Automation Magazine,* vol. 5, no. 1, pp. 36-47, 1998.

[45] R. Simmons, D. Apfelbaum, W. Burgard, M. Fox, D. Moors, S. Thrun and H. Younes, "Coordination for multi-robot exploration and mapping," in *Proceedings of the AAAI National Conference on Artificial Intelligence*, Austin, TX, 2000.

[46] B. P. Gerkey and M. J. Matarić, "Principled Communication for Dynamic Multi-Robot Task Allocation," *Experimental Robotics VII. Lecture Notes in Control and Information Sciences,* vol. 271, pp. 353-362, 2001.

[47] A. R. Mosteo, L. Montano and M. G. Lagoudakis, "Multi-robot routing under limited communication range," in *2008 IEEE International Conference on Robotics and Automation*, 2008.

[48] R. Doriya, S. Mishra and S. Gupta, "A brief survey and analysis of multi-robot communication and coordination," in *International Conference on Computing, Communication & Automation*, 2015.

[49] J. Stephan, J. Fink, V. Kumar and A. Ribeiro, "Concurrent Control of Mobility and Communication in Multirobot Systems," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1248-1254, 2017.

[50] K. Cesare, R. Skeele, S.-H. Yoo, Y. Zhang and G. Hollinger, "Multi-UAV exploration with limited communication and battery," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[51] Y. Kantaros and M. M. Zavlanos, "Global Planning for Multi-Robot Communication Networks in Complex Environments," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1045-1061, 2016.

[52] M. Sinay, N. Agmon, O. Maksimov, S. Kraus and D. Peleg, "Maintaining communication in multi-robot tree coverage," in *IJCAI*, 2017.

[53] S. Kemna, J. G. Rogers, C. Nieto-Granda, S. Young and G. S. Sukhatme, "Multi-robot coordination through dynamic Voronoi partitioning for informative adaptive sampling in communication-constrained environments," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[54] J. Banfi, Multi-robot exploration of communication restricted environment, POLITECNICO DI MILANO, 2013.

[55] F. Amigoni, J. Banfi and N. Basilico, "Multirobot Exploration of Communication-Restricted Environments: A Survey," *IEEE Intelligent Systems*, vol. 32, no. 6, pp. 48-57, 2017.

[56] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97*, 1997.

[57] H. González-Baños and J. Latombe, "Navigation Strategies for Exploring Indoor Environments," *The International Journal of Robotics Research,* pp. 829-848, 2002.

[58] H. Surmann, A. Nüchter and J. Hertzberg, "An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments," *Robotics and Autonomous Systems,* vol. 43, no. 3-4, pp. 181-198, 2003.

[59] C. Zhu, R. Ding, M. Lin and Y. Wu, "A 3D Frontier-Based Exploration Tool for MAVs," in *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, 2015.

[60] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova and R. Siegwart, "Receding Horizon "Next-Best-View" Planner for 3D Exploration," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[61] P. G. C. N. Senarathne and D. Wang, "Towards autonomous 3D exploration using surface frontiers," in *016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2016.

[62] C. Papachristos, S. Khattak and K. Alexis, "Uncertainty-aware receding horizon exploration and mapping using aerial robots," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[63] H. Umari and S. Mukhopadhyay, "Autonomous robotic exploration based on multiple rapidly-exploring randomized trees," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[64] A. Dai, S. Papatheodorou, N. Funk, D. Tzoumanikas and S. Leutenegger, "Fast Frontier-based Information-driven Autonomous Exploration with an MAV," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[65] M. Selin, M. Tiger, D. Duberg, F. Heintz and P. Jensfelt, "Efficient Autonomous Exploration Planning of Large-Scale 3-D Environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1699-1706, 2019.

[66] Wikipedia, "Dijkstra's algorithm," [Online]. Available: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm. [Accessed 14 08 2021].

[67] ROS.org, "teb_local_planner," [Online]. Available: http://wiki.ros.org/teb_local_planner. [Accessed 14 18 2021].

[68] P. Sermanet, R. Hadsell, M. Scoffier, U. Muller and Y. LeCun, "Mapping and planning under uncertainty in mobile robots with long-range perception," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

[69] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker and A. J. Davison, "ElasticFusion: Dense SLAM Without A Pose Graph," in *Proceedings of Robotics: Science and Systems Conference (RSS)*, 2015.

[70] M. G. Ocando, N. Certad, S. Alvarado and Á. Terrones, "Autonomous 2D SLAM and 3D mapping of an environment using a single 2D LIDAR and ROS," in *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*, 2017.

[71] S. H. Juan and F. H. Cotarelo, "Multi-master ROS systems," Institut de Robòtica i Informàtica Industrial (IRI), 2015.

[72] J. Hörner, Map-merging for multi-robot system, Univerzita Karlova, Matematicko-fyzikální fakulta, 2016.

[73] J. Horner, "m-explore," gitHub , [Online]. Available: https://github.com/hrnr/m-explore. [Accessed 24 07 2021].

[74] "ros-drivers/freenect_stack," gitHub, [Online]. Available: https://github.com/ros-drivers/freenect_stack. [Accessed 24 07 2021].

[75] "Kobuki's Control System," ROS.org, [Online]. Available: http://wiki.ros.org/kobuki/Tutorials/Kobuki%27s%20Control%20System. [Accessed 24 07 2021].

[76] L. Cruz, D. Lucio and L. Velho, "Kinect and RGBD Images: Challenges and Applications," in *25th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials*, 2012.

[77] QUANSER QBot 2 for QUARC, Quanser Inc., 2018.

[78] NVIDIA, "Jetson TX2," [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/. [Accessed 24 07 2021].

[79] Geeks3D, "(GPU Computing) NVIDIA CUDA Compute Capability Comparative Table," 06 06 2010. [Online]. Available: https://www.geeks3d.com/20100606/gpu-computing-nvidia-cuda-compute-capability-comparative-table/. [Accessed 05 10 2021].

[80] Wikipedia, "Bresenham's line algorithm," [Online]. Available: https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm. [Accessed 24 07 2021].

[81] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," in *Proceedings of Eurographics*, Amsterdam, The Netherlands, 1987.

[82] P. Biber and W. Strasser, "The normal distributions transform: a new approach to laser scan matching," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, 2003.

[83] H. Men, B. Gebre and K. Pochiraju, "Color point cloud registration with 4D ICP algorithm," in *2011 IEEE International Conference on Robotics and Automation*, 2011.

[84] K. Pathak, N. Vaskevicius, J. Poppinga, M. Pfingsthorn, S. Schwertfeger and A. Birk, "Fast 3D mapping by matching planes extracted from range sensor point-clouds," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[85] S. May, D. Droeschel, S. Fuchs, D. Holz and A. Nüchter, "Robust 3D-mapping with time-of-flight cameras,," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[86] P. Henry, M. K. M, E. Herbst, X. Ren and D. Fox, "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647-663, 2012.

[87] F. Steinbrücker, J. Sturm and D. Cremers, "Volumetric 3D mapping in real-time on a CPU," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[88] J. Yang, H. Li, D. Campbell and Y. Jia, "Go-ICP: A Globally Optimal Solution to 3D ICP Point-Set Registration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 11, pp. 2241-2254, 2015.

[89] L. Ma, J. Zhu, L. Zhu, S. Du and J. Cui, "Merging grid maps of different resolutions by scaling registration," *Robotica*, vol. 34, no. 11, pp. 2516-2531, 2016.

[90] S. Saeedi, L. Paull, M. Trentini, M. Seto and H. Li, "Map merging for multiple robots using Hough peak matching," *Robotics and Autonomous Systems*, vol. 62, no. 10, pp. 1408-1424, 2014.

[91] J. Park, A. J. Sinclair, R. E. Sherrill, E. A. Doucette and J. W. Curtis, "Map Merging of Rotated, Corrupted, and Different Scale Maps using Rectangular Features," in *Proceedings of IEEE/ION PLANS 2016*, Savannah, GA, April 2016.

[92] C. A. V. Hernández and F. A. P. Ortiz, "A real-time map merging strategy for robust collaborative reconstruction of unknown environments," *Expert Systems with Applications,* vol. 145, 2020.

[93] E. Hołowko, J. Wojsz, R. Sitnik and M. Karaszewski, "automatic merging of multiview 3D point clouds," *Journal on Computing and Cultural Heritage ,* 2014.

[94] J. Jessup, S. N. Givigi and A. Beaulieu, "Robust and Efficient Multirobot 3-D Mapping Merging With Octree-Based Occupancy Grids," *IEEE Systems Journal,* vol. 11, no. 3, pp. 1723-1732, 2017.

[95] Y. Yue, D. Wang, P. G. C. N. Senarathne and D. Moratuwage, "A hybrid probabilistic and point set registration approach for fusion of 3D occupancy grid maps," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2016.

[96] T. M. Bonanni, B. D. Corte and G. Grisetti, "3-D Map Merging on Pose Graphs," *IEEE Robotics and Automation Letters,* vol. 2, no. 2, pp. 1031-1038, 2017.

[97] B. Williams, M. Cummins, J. Neira, P. Newman, I. Reid and J. Tardos, "An image-to-map loop closing method for monocular SLAM," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.

[98] A. Nuchter., K. Lingemann, J. Hertzberg and H. Surmann, "6D SLAM – 3D mapping outdoor environments," *J. Field Robot,* vol. 24, p. 699–722, August 2007.

[99] S. Choudhary, A. J. B. Trevor, H. I. Christensen and F. Dellaert, "SLAM with object discovery, modeling and mapping," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[100] C. Brand, M. J. Schuster, H. Hirschmüller and M. Suppa, "Submap matching for stereo-vision based indoor/outdoor SLAM," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[101] L. Gallagher and J. B. McDonald, "Collaborative Dense SLAM," *arXiv.org*, 2019.

[102] H. Zhang, X. Chen, J. Lu and J. Xiao, "Distributed and collaborative monocular simultaneouslocalization and mapping for multi-robot systems in large-scale environments," *International Journal of Advanced Robotic Systems*, vol. 15, no. 3, 2018.

[103] X. Chen, T. Läbe, A. Milioto, T. Röhling, O. Vysotska and A. H. J. B. C. Stachniss, "OverlapNet: Loop Closing for LiDAR-based SLAM," *arXiv.org*, 2021.

[104] A. Memon, . Wang and A. Hussain, "Loop closure detection using supervised and unsupervised deep neural networks for monocular SLAM systems," *Robotics and Autonomous Systems*, 2020.

[105] R. Aragues, C. Sagues and Y. Mezouar, "Feature-based map merging with dynamic consensus on information increments," *Auton Robot* , vol. 38, p. 243–259, 2015.

[106] S. Thrun and A. Btlcken, "Integrating grid-based and topological maps for mobile robot navigation," in *Proceedings AAAI-96* , Portland, OR, 1996.

[107] P. Henry, M. Krainin, E. Herbst, X. Ren and D. Fox, "RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments.," *Experimental Robotics. Springer Tracts in Advanced Robotics*, vol. 79, pp. 477-491, 2014.

[108] J. Park, Q.-Y. Zhou and V. Koltun, "Color point cloud registration revisited," in *ICCV*, 2017.

[109] Y. Yue, A framework of probabilistic 3D map fusion for collaborative robots. Doctoral thesis, Nanyang Technological University, Singapore, 2019.

[110] Wikipedia, "Graphics processing unit," [Online]. Available: https://en.wikipedia.org/wiki/Graphics_processing_unit. [Accessed 24 07 2021].

[111] "CPU vs GPU: What is GPU Computing?," Apps4Rent: premium application hosting , [Online]. Available: https://www.apps4rent.com/cpu-vs-gpu.html. [Accessed 24 07 2021].

[112] A. Cano, "A survey on graphic processing unit computing for large-scale data mining," *Data Min. Knowl. Discov,* vol. 8, no. 1, 2018.

[113] G. Vosselman and S. Dijkman, "3D building model reconstruction from point clouds and ground plans," *Int. Arch. Photogramm. Remote Sens,* vol. 34, no. 3, p. 37–43, 2001.

[114] C. Liang-Chien, T. Tee-Ann, K. Chih-Yi and R. Jiann-Yeou, "Shaping Polyhedral Buildings by the Fusion of Vector Maps and Lidar Point Clouds," *Photogrammetric Engineering & Remote Sensing,* vol. 9, pp. 1147-1157, 2008.

[115] P. Grussenmeyer, E. Alby, T. Landes, M. Koehl, S. Guillemin, J. F. Hullo, P. Assali and E. Smigiel, "Recording approach of heritage sites based on merging point clouds from high resolution photogrammetry and Terrestrial Laser Scanning.," in *Proceedings of the XXII ISPRS Congress* , Melbourne, Australia, 2012.

[116] R. Zlot and M. Bosse, "Three-dimensional mobile mapping of caves," *Journal of Cave and Karst Studies,* vol. 76, no. 3, pp. 191-206, 2014.

[117] P. Payeur, P. Hebert, D. Laurendeau and C. M. Gosselin, "Probabilistic octree modeling of a 3D dynamic environment," in *Proceedings of International Conference on Robotics and Automation*, 1997.

[118] P. Payeur, D. Laurendeau and C. M. Gosselin, "Range data merging for probabilistic octree modeling of 3D workspaces," in *IEEE International Conference on Robotics and Automation*, 1998.

[119] E. Einhorn, C. Schröter and H. Gross, "Finding the adequate resolution for grid mapping - Cell sizes locally adapting on-the-fly," in *2011 IEEE International Conference on Robotics and Automation*, 2011.

[120] L. Heng, G. H. Lee, F. Fraundorfer and M. Pollefeys, "Real-time photo-realistic 3D mapping for micro aerial vehicles," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[121] J. Jessup, S. N. Givigi and A. Beaulieu, "Robust and efficient multi-robot 3D mapping with octree based occupancy grids," in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2014.

[122] N. Michael and e. al, "Collaborative Mapping of an Earthquake Damaged Building via Ground and Aerial Robots.," *Field and Service Robotics. Springer Tracts in Advanced Robotics*, vol. 92, pp. 33-47, 2013.

[123] D. Chetverikov, D. Svirko, D. Stepanov and P. Krsek, "The Trimmed Iterative Closest Point algorithm," in *2002 International Conference on Pattern Recognition*, 2002.

[124] Z. Jiang, J. Zhu, Y. Li, J. Wang, Z. Li and H. Lu, "Simultaneous Merging Multiple Grid Maps Using the Robust Motion Averaging," *J Intell Robot Syst.*, vol. 94, p. 655–668, 2019.

[125] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image and Vision Computing*, vol. 10, no. 3, pp. 145-155, 1992.

[126] Y. Yue, P. G. C. N. Senarathne, C. Yang, J. Zhang, M. Wen and D. Wang, "Hierarchical Probabilistic Fusion Framework for Matching and Merging of 3-

D Occupancy Maps," *IEEE Sensors Journal,* vol. 18, no. 21, pp. 8933-8949, 2018.

[127] A. Segal, D. Haehnel and S. Thrun, "Generalized-ICP," in *Proceedings of Robotics: Science and Systems*, Seattle, USA, 2009.

[128] D. Holz, A. E. Ichim, F. Tombari, R. B. Rusu and S. Behnke, "Registration with the Point Cloud Library: A Modular Framework for Aligning in 3-D," *IEEE Robotics & Automation Magazine,* vol. 22, no. 4, pp. 110-124, 2015.

[129] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999.

[130] Y. Zhong, "Intrinsic shape signatures: A shape descriptor for 3D object recognition," in *ICCV Workshop on 3D Representation for Recognition (3dRR)*, 2009.

[131] B. Steder, R. Rusu, K. Konolige and W. Burgard, "NARF: 3D range image features for object recognition," in *IROS Workshop on Defining and Solving Realistic Perception*, 2010.

[132] A. Johnson and M. Hebert, "Using spin images for efficient object recognition in cluttered 3D scenes," in *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 1999.

[133] R. B. Rusu, N. Blodow and M. Beetz, "Fast point feature histograms (FPFH) for 3D registration," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[134] R. Hansch, T. Weber and O. Hellwich, "Comparison of 3d interest point detectors and descriptors for point cloud fusion," *ISPRS Journal of Photogrammetry, Remote Sensing and Spatial Information Sciences,* p. 57–64, 2014.

[135] "Efficient RANSAC for Point-Cloud Shape Detection," *Computer Graphics Forum*, vol. 26, no. 2, pp. 214-226, 2007.

[136] ROS.org, "multimaster_fkie," [Online]. Available: http://wiki.ros.org/multimaster_fkie. [Accessed 24 07 2021].

[137] Wikipedia, "Motion planning," [Online]. Available: https://en.wikipedia.org/wiki/Motion_planning. [Accessed 24 07 2021].

[138] P. Corke, "Chapter 5: Navigation," in *Robotics, Vision and Control: Second edition* , Springer , 2017.

[139] introlab, "RTAB-Map, Real-Time Appearance-Based Mapping," GitHub, [Online]. Available: http://introlab.github.io/rtabmap/. [Accessed 14 08 2021].