Dense Reconstruction from Visual SLAM with Probabilistic Multi-Sequence Merging

by

Hanxiang Zhang

Submitted in partial fulfilment of the requirements
for the degree of Master of Applied Science

at

Dalhousie University
Halifax, Nova Scotia
December 2023

Table of Contents

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This thesis presents a comprehensive visual SLAM system that extends the application of ORB-SLAM3. Using it as a template, a supplementary and optional function of 3D dense reconstruction is implemented for both RGB-D and stereo cameras. With conventional datasets, TUM, EuRoC, and KITTI as benchmarks, we confirm the validity of proposed system in both indoor and outdoor scenarios. Besides, the concept of Octree is integrated into our system to generate Octomap. A compact mapping can be achieved as such, verified by the fact that the size of each dense point cloud map is reduced to approximately one-fifth after the conversion. Furthermore, a multi-sequence merging method is included in our proposed system, formulating with a probabilistic-based optimizing algorithm and map accessing functions from the original system. Multi-sequence experiments evince that the tracking accuracy profits from the exploitation of a priori knowledge gathered through the preceding sequences.

LIST OF ABBREVIATIONS

SLAM	Simultaneous Localization and Mapping
UGV	Unmanned Ground Vehicle
UAV	Unmanned Aerial Vehicle
AUV	Autonomous Underwater Vehicle
LIDAR	Light Detection and Ranging
RADAR	Radio Detection and Ranging
SONAR	Sound Navigation and Ranging
IMU	Inertial Measurement Unit
GNSS	Global Navigation Satellite System
RANSAC	Random Sample Consensus
ICP	Iterative Closest Point
SIFT	Scale-Invariant Feature Transform
SURF	Speeded-Up Robust Features
ORB	Oriented FAST and rotated BRIEF
FAST	Features from Accelerated Segment Test
BRIEF	Binary Robust Independent Elementary Features

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my supervisor, Dr. Jason Gu, for his invaluable guidance, unwavering corroboration, and insightful feedback throughout my study.

I am very grateful to my peer Chang Xu. We worked as one.

I also appreciate my roommate Qiguang Chen for providing meticulous support to me through all means.

I want to thank my parents as well. Their silent support has been the driving force behind me.

Last but not least, I extend my appreciation to my committee, Dr. Ya-Jun Pan and Dr. Kamal El-Sankary, for taking time out of a busy schedule to attend my defense and offer their beneficial suggestions.

CHAPTER 1 INTRODUCTION

1.1 Background

As one of the most popular and intense research topics in the robotics field, the SLAM algorithm capacitates mobile robots to localize themselves while exploring and mapping an unknown environment. Being widely used in industry, SLAM algorithms can be implemented to different types of robots, depending on the purpose of application, including agriculture, healthcare, manufacturing, smart cities, etc. Studies have been conducted during the past decades [1], resulting to the proposition of numerous SLAM systems. Implementing heterogeneous sensors, applying various optimization techniques, and picturing with different mapping methods, all the SLAM algorithms are aiming to maintain the system robustness, enhance the tracking accuracy and achieve the real-time performance.

SLAM algorithms can read from sensors including, but not limited to, camera, lidar, radar, IMU, and GNSS. Table I below provides an example of categorizing some typical sensors used by SLAM algorithms based on its application.

Table I: Example of SLAM Applications with Typical Sensors

SLAM Application	**Typical Sensors**		
UGV	Camera	Lidar	IMU
UAV	Lidar	IMU	GPS
AUV	Sonar	Camera	IMU

Choosing appropriate sensors is of utmost importance for the SLAM application. In terms of UGVs, Lidar SLAM and Visual SLAM are considered together as the two mainstreams. They are named by the main sensor used in the algorithms: Lidar SLAM utilizes 2D or 3D lidars. Visual SLAM adopts monocular camera, stereo camera, and sometimes RGB-D camera – a hybrid sensor that also collects depth information. Figure 1 examples each of these sensors.

(a) LS01B1 Rotating 2D LIDAR [24]

(b) Ultra Puck 3D Lidar [25]

(c) C270 HD Monocular Camera [26]

(d) RealSense™ D405 RGB-D Camera [27]

Figure 1: Example of Different Lidar and Camera Sensors

Lidar-based algorithms are developed in earlier years and are considered as a mature technology nowadays. The highly reliable and accurate maps can be later used by path planning algorithms. The lidar-based algorithms are, nonetheless, too expensive as a solution when building products like a robot vacuum cleaner. It is common that the price of a lidar sensor, varying by its accuracy, can reach thousands of dollars, and even beyond.

During the past years, many approved Lidar SLAM algorithms have been published, including Gmapping [2], Hector-SLAM [3], LOAM [4], LIO-SAM [5], etc.

Of these two branches, the latter uses cameras as its main sensor to estimate the robot pose. It has not been wide-ranging discussed, in-depth studied and high-speed evolved until the late 2000s, due to limiting factors such as the undersupply of computing resources. Thanks to the development of computer technology, more computing resources are accessible to the researchers in recent years. Being able to handle the high workload from image processing, the vision-based algorithms become a trend of research.

Unlike the lidar-based algorithms that provide accurate point clouds without cumulative error, a vision-based SLAM algorithm has to extract the key features from each image captured by the camera, and match across multiple images to determine the relative changes on both pose and position of the camera. This usually gives extremely inaccurate results. The main contributing factors include the defect of manufacturing, noises of sensor, and cumulative errors. Therefore, additional optimization algorithms are required to produce credible results.

The early-stage Visual SLAM algorithms are mostly filter-based. MonoSLAM [6] is acknowledged as the first real-time monocular solution, making use of the extended Kalmen filter (EKF) algorithm [7]. However, implementing EKF causes consistency issue during linearization. [8] improves this with unscented Kalmen filter (UKF) [9], and UFastSLAM [10] overcomes this drawback by introducing Rao-Blackwell particle filter (RBPF) [11] to SLAM system.

Some other Visual SLAM systems rely on keyframes. PTAM [12], as a representative algorithm, shows an ingenious process using two parallel threads. It separates the tracking

and mapping tasks, obtaining comparable results at a lower computing cost; nonetheless, PTAM is constrained by the adequacy of feature matching. Direct registration algorithms avoid this issue by making direct operations to the intensity of pixels in lieu of feature extractions. DTAM [13] constructs a depth map with all pixels in an RGB image, promoting a more robust and accurate tracking when suffering from feature deficits. In contrast, the real-time operation of DTAM cannot be accomplished without considerable GPU resources. LSD-SLAM [14] alters the selection of pixels, adapting it for larger-scale scenarios. Instead of all the pixels, only those within high-gradient regions are selected for computation. However, as a direct method, which is formed on the gray-scale invariant assumption, its robustness and accuracy can be undermined by unmodelled behaviors like lens vignetting and drastic changes of illumination.

ORB-SLAM [15] is a monocular feature-based SLAM system that has been divided into three threads: tracking, local mapping and loop closing. It uses oriented FAST and rotated BRIEF (ORB) features [31] to achieve a real-time operation without GPU. Essential Graph is first introduced for speedier loop closing. The successor ORB-SLAM2 [16] extends the original work to be compatible with stereo camera and RGB-D camera. A place recognition module is established on DBoW2 [17], and being applied to the system for relocalization, reinitialization and loop detection. The accuracy is further improved by implementing the EPnP [18] algorithm. ORB-SLAM3 [19] is a state-of-the-art algorithm published recently. Monocular-inertial and stereo-inertial options are integrated with its previous work, providing this SLAM system with additional robustness and accuracy. The refinement of place recognition algorithm increases the recall rate significantly, preserving the system

from getting lost in low-textured scenes. It also uses a multi-map sub-system that underlies ORBSLAM-Atlas [20].

Despite the fact that ORB-SLAM3 is one of the most powerful and valuable visual SLAM systems, the output semi-dense map confines its range of application. On the other hand, this algorithm does not trim, nor release any past landmarks – meaning all the landmarks are stored cumulatively until the end of program. Tested with a large-scale dataset, the time efficiency suffers after processing hundreds of keyframes.

Novel research aiming to enhance the system accuracy are being continually proposed to this day. SOFT2 [37] makes significant improvement on solving the epipolar geometry and kinematics, bringing resilience to object depth uncertainty. CT-ICP [38], although working with the LiDAR configuration, demonstrates a state-of-the-art loop detection method, achieving minor pose error in the validation with benchmark datasets. [39] reports their work on obtaining more precise front-end estimations. OV^2SLAM [40] utilizes a multi-threaded architecture for a robust and precise system. [41] describes progress on motion estimations using an innovative technique to integrate information from multiple stereo camera configurations, named joint forward-backward visual odometry with multiple cameras and feedback mechanism. RT-SLAM [42] processes visual information with semantic algorithm to achieve accurate motion estimations. Dynam-SLAM [43] presents a stereo visual-inertial SLAM system to survive under dynamic environments by defining virtual landmarks. D3VIL-SLAM [44] applies fusion of cameras with both LiDAR and inertial measurements, and an enhancement on robot pose estimations is validated through the benchmark dataset.

1.2 Contributions

In this thesis, we propose an accurate visual SLAM system with dense point cloud reconstruction and probabilistic multi-sequence merging. The contributions are listed below:

A dense mapping approach is inspired by the RGB-D SLAM [21][22] and embedded to the original system as an optional feature. It reconstructs the maps in a more detailed level that enriches the robot scene understanding and perception. We offer this implementation with either stereo camera or RGB-D camera. Sensing through a stereo camera not only reduce the hardware complexity but also demonstrates enhanced performance in texturally rich environment leveraging its reliance on feature matching across plentiful stereo pairs. The utilization of an RGB-D camera is also kept since it is considered to remain superior functionality facing occlusions due to the nature of having a direct depth measurement.

Moreover, an Octomap [23] is integrated to the original system, aiming to enhance the mapping with finer texture and shading with improved memory efficiency and real-time updates. It expands the system to be used by further robotic applications, especially those demanding accurate 3D spatial understanding, such as mobile manipulation and navigation.

Last but not least, the proposed system includes a multi-sequence merging method that fetches the knowledge of one sequence with the next and conducts a probabilistic-based optimization to those redundant map points. Multiple benchmark datasets have been tested to validate its beneficial impact on facilitating precise estimations of the camera pose.

1.3 Organization

This section contains a sequence of the thesis chapters, and also a brief description of each, shown as follows:

Chapter 1: This chapter includes an introduction to SLAM, reveals the challenges that researchers have experienced and those are experiencing, and describes the motivation of my thesis.

Chapter 2: This chapter contains a literature review on related SLAM systems, and explains the principles used by them.

Chapter 3: This chapter exhibits the system architecture of my proposed SLAM system, and explicates the improvements made to the original SLAM system.

Chapter 4: This chapter evinces the feasibility of the proposed SLAM system by comparing it with previous work. Results obtained from multiple sets of experiments are analyzed and discussed to prove the enhancement on system performance.

Chapter 5: This chapter concludes the work of this thesis and advises the future orientation.

CHAPTER 2 LITERATURE REVIEW

RGB-D SLAM [21][22] and ORB-SLAM family are introduced in this section. The former allows to construct colored 3D models from camera. Having the same mapping method as our proposed system, it is included in this thesis to make a fair comparison, evincing the system performance. This is detailed in the fourth chapter. On the other hand, ORB-SLAM family consists of three generations. Evolving in the past decade, it has been acknowledged to be one of the most powerful visual SLAM algorithms. An introduction to all the generations in ORB-SLAM family is presented in this section since our system is built on the basis of its latest generation, ORB-SLAM3.

2.1 RGB-D SLAM

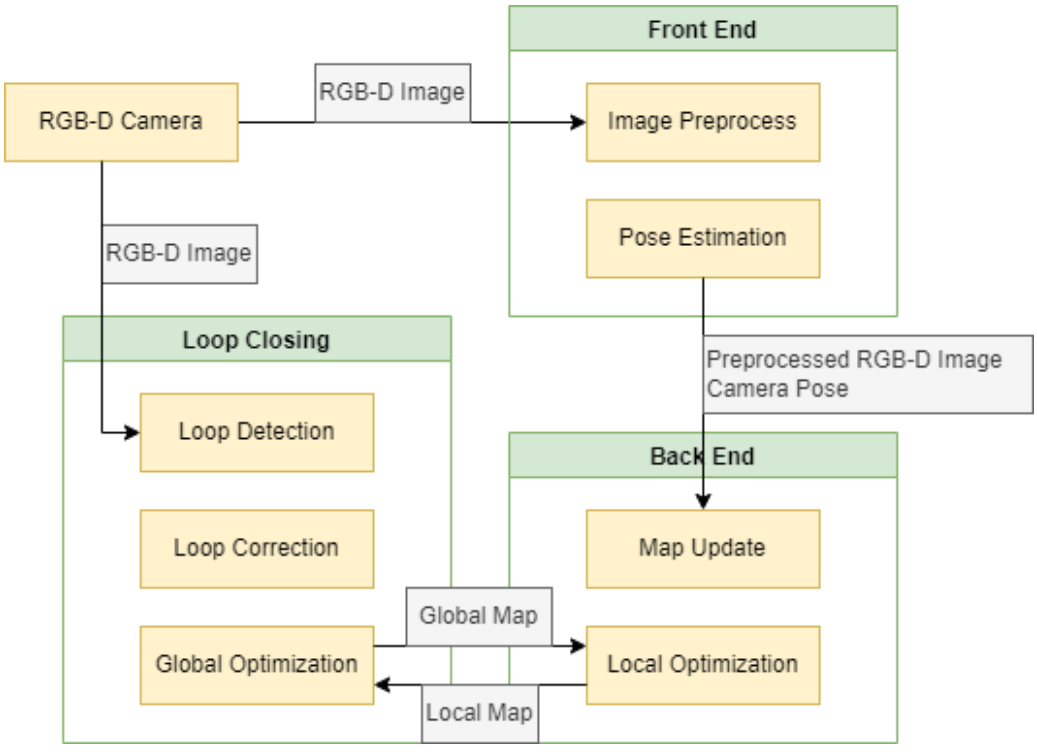


Figure 2: General Framework of SLAM System with RGB-D Camera

Figure 2 demonstrates the general framework for SLAM system with an RGB-D camera as the input sensor. The process is explained as following: The sensor data from an RGB-D camera works together with the global map in the front end. They are used to for the image preprocess and pose estimation. With the preprocessed image and the estimated camera pose, the back end of this system updates the local map and optimizes it. A loop closing pipeline containing loop detection, correction utilizes the local map to optimize and update the global map. This updated map is stored for the succeeding data.

RGB-D SLAM [21] and its second version named RGB-D SLAMv2 [22] were proposed by Felix Endres et al. This section focuses on reviewing the latest version since it covers most work of its ancestor as well as sharing a similar system structure. A schematic overview of RGB-D SLAMv2 is excerpted as in Figure 3 [22].

This system extracts SIFT [30], SURF [31] or ORB [32] features from the color image and matches them across adjacent images. The depth information directly obtained from camera helps to locate the sensor in 3D space. Random Sample Consensus (RANSAC) algorithm is used to robustly filter out the outliers among the camera pose estimations. Only those inliers are left to compute a refined transformation, generating an edge in the backend pose graph. On the other hand, Iterative Closest Point (ICP) algorithm aims to estimate the best transformations in point clouds generated from motions defined by the Lie parameterization $SE(3)$. The last step of this system provides an effective nonlinear optimization on the pose graph, where a minimization problem is solved on an error function. The following part in this section details this process.

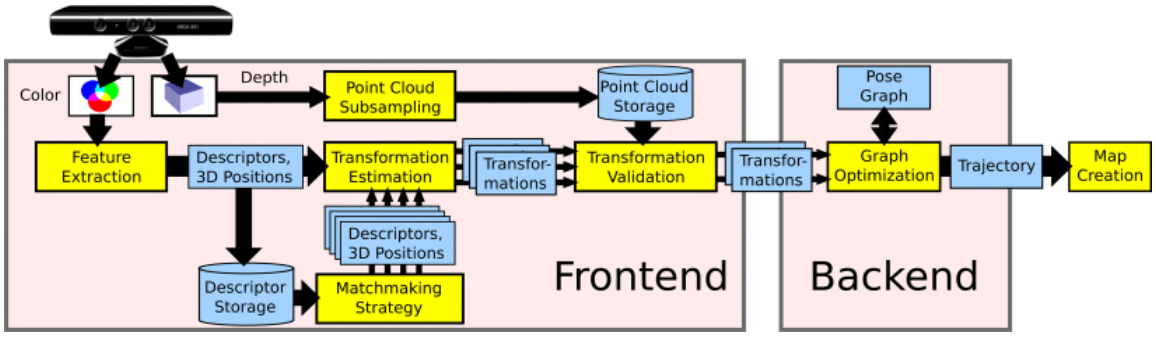


Figure 3: Schematic Overview of RGB-D SLAMv2 [22]

First, some commonly used feature extraction and matching algorithms are introduced, including SIFT, SURF and ORB. The concepts of RANSAC and ICP are explained afterwards. This section ends with the error function and optimizations that RGB-D SLAM uses.

SIFT stands for scale-invariant feature transform. The SIFT algorithm provides scale-invariant to the feature extraction process. This starts with blurring an input image. The image is minified to multiple octaves and blurred progressively, creating a scale space, also called a Gaussian pyramid. Representing with math, the blurred images $L(x, y; \sigma)$ can be calculated with the convolution of the original image $I(x, y)$ and a Gaussian blur operator G:

$$L(x, y; \sigma) = G(x, y; \sigma) * I(x, y) \tag{1}$$

where σ is the Gaussian blur factor.

The operator is defined as:

$$G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2}$$

A Difference of Gaussian (DoG) kernel is established afterwards by comparing the difference of adjacent images with different blur factors, for instance: σ and $k\sigma$. It is said to have a higher accuracy of finding keypoints in the input image. This is completed for all octaves in the Gaussian pyramid. Therefore, if there are n layers in an octave of the pyramid, the DoG will contain $n-1$ layers for each octave. Figure 4 [29] below shows a set of progressively blurred images, with blur factor $\sigma = 0.5, 1, 2, 4, 8, 16$, respectively.

Figure 4: Image with Progressively Blur Scale [29]

DoG is defined as:

$$D_{\sigma,k}(x,y) = L(x,y;\sigma) - L(x,y;k\sigma), \qquad k > 1 \tag{3}$$

Calculating the Laplacian of Gaussian (LoG) approximations is the following step. It is essential for having the scale invariant property. The idea is to compare each pixel in an

image with its 8 neighbors, as well as the 9 pixels in both the previous and next octave of different blur scale. If the current pixel is a local extremum compared to its 26 neighbors, it will be treated as a potential keypoint.

LoG can be calculated by:

$$\nabla^2 G_\sigma(x,y) \approx \frac{G_{k\sigma}(x,y) - G_\sigma(x,y)}{(k-1)\sigma^2} \tag{4}$$

The DoG representation in equation (3) is commonly used as an approximation of LoG in equation (5) due to its time-efficiency. The scale factor $k = 1.6$ is preferred.

However, the previous step gives more than necessary number of extrema. Those lies along an edge and those without enough contrast are removed. The latter is checked with the intensity of pixel and the earlier is performed by calculating its principal curvature with a 2×2 Hessian matrix $\boldsymbol{H}$. Representing with math, a potential keypoint is determined to be rejected and discarded if having:

$$\begin{cases} |D(x,y)| < 0.03 \\ \dfrac{\alpha}{\beta} < 10 \end{cases} \tag{5}$$

where

$$\alpha, \beta = \text{eigen}(\boldsymbol{H}), |\alpha| \geq |\beta|; \boldsymbol{H}(x,y) = \begin{bmatrix} D_{xx}(x,y) & D_{xy}(x,y) \\ D_{yx}(x,y) & D_{yy}(x,y) \end{bmatrix} \tag{6}$$

where D_{xy} is the second order difference quotient of DoG with respect to x first and y next. The other three terms are following the same rule. Equations below examples the calculations of the first and second order difference quotients:

$$D_x(x,y) = L(x+0.5,y) - L(x-0.5,y) \tag{7}$$

$$D_{xx}(x,y) = D_x(x+0.5,y) - D_x(x-0.5,y) = \\ L(x+1,y) + L(x-1,y) - 2L(x,y) \tag{8}$$

The two eigenvalues of the Hessian matrix of a feature point indicate the corresponding shape around this point: When α and β both have large magnitudes, a corner is detected. When α is far larger than β, it is recognized as an edge, which is one of the two cases that the potential keypoints should be rejected. The last case is for having small values of both α and β. In this case, it is a low-contrast region that should also be rejected.

The legitimate keypoints obtained from previous steps are proven to be stable and scale invariant. The succeeding step called orientation assignment provides SIFT algorithm rotation invariance. The gradient magnitude $m(x,y)$ and direction $\theta(x,y)$ are calculated for each point in an orientation collection region around the keypoint $L(x,y)$. They can be calculated by:

$$\begin{cases} m(x,y) = \sqrt{[L(x+1,y) - L(x-1,y)]^2 + [L(x,y+1) - L(x,y-1)]^2} \\ \theta(x,y) = \arctan\dfrac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)} \end{cases} \tag{9}$$

An orientation histogram is used as a compilation of each keypoint. The horizontal axis contains sectors that cover all 360 degrees. Configurations can be set manually depending on the accuracy required. For examples, 10 sectors (36 degrees each), and 8 sectors (45 degrees each) can be used. The vertical axis is for the cumulative intensities for all pixels in the orientation collection region. The highest peak in the histogram marks the orientation of the keypoint.

Figure 5 [30] visualizes the process of determining the orientation of a keypoint. The blue circle indicates the orientation collection region. As can be seen that each pixel around the

keypoint at the center are marked with a gradient with a black arrow, representing its magnitude and orientation. The right part of this figure shows a detailed gradient within each pixel.

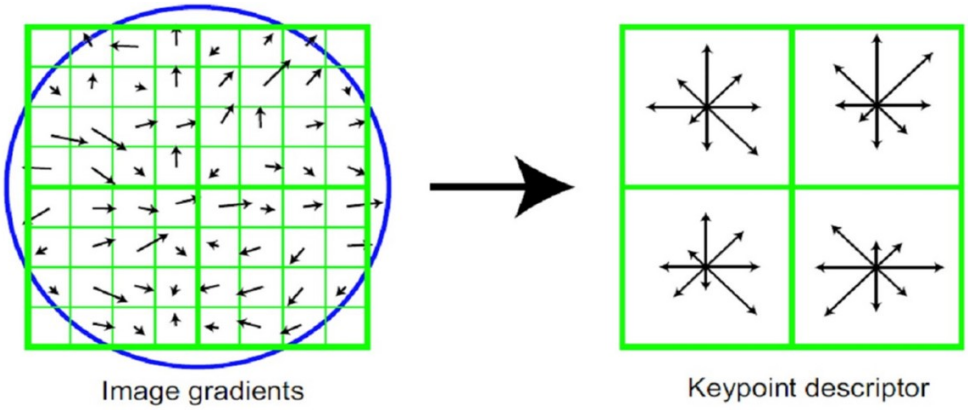


Figure 5: Process of Calculating Gradient Magnitude and Orientation of a Keypoint [30]

So far, the location, scale, and orientation of all keypoints have been determined. A distinctive and invariant descriptor is going to be assigned to each keypoint. During this step, 4×4 blocks are windowed around the keypoint, which can be further combined into 16×16 sub-blocks. For each block, gradient in 8 sectors are calculated with similar steps as previous. Therefore, a 128-dimentional ($4 \times 4 \times 8$) array can be used as the descriptor of a keyframe.

At the end, in order to match the same feature across different images, the Euclidean distance between keypoints in both images is calculated. The less distance they have, the better they are matched.

Another feature extraction algorithm is SURF (Speeded-Up Robust Features) [30], it is developed to increase to computing speed of SIFT. Instead of Gaussian filter for SIFT, SURF utilizes box filters, enabling it for real-time applications.

The integral image $I_\Sigma(\boldsymbol{x})$ at $\boldsymbol{x} = (x, y)^T$ shows the summation of all pixels inside a rectangular region formed by $\boldsymbol{x}$ and the origin. It can be calculated by:

$$I_\Sigma(\boldsymbol{x}) = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} I(i, j) \tag{10}$$

The main benefit of using integral image is to accelerate the computation of approximate second order Gaussian derivatives, independently of size. Furthermore, instead of using a LoG approximation as previously introduced in the SIFT algorithm, SURF calculates both the approximation for convolution and for the second order derivative in a more effective way. Given that:

$$\boldsymbol{H}(\boldsymbol{x}; \sigma) = \begin{bmatrix} L_{xx}(\boldsymbol{x}; \sigma) & L_{xy}(\boldsymbol{x}; \sigma) \\ L_{yx}(\boldsymbol{x}; \sigma) & L_{yy}(\boldsymbol{x}; \sigma) \end{bmatrix} \tag{11}$$

Figure 6 [31] compares the Gaussian second order partial derivatives in y-direction (top-left) and xy-direction (bottom-left) with the box filter approximations. A 9×9 box filter is used to estimate the Gaussian second order derivatives with blur factor $\sigma = 1.2$.

In this case, the determinant of Hessian matrix can be calculated by:

$$\det(\boldsymbol{H}_{approx}) = D_{xx} D_{yy} - (\varepsilon D_{xy})^2 \tag{12}$$

Where ε is a relative weight that manually balanced, determined by the Frobenius norms:

$$\varepsilon = \frac{|L_{xy}(1.2)|_F |D_{xx}(9)|_F}{|L_{xx}(1.2)|_F |D_{xy}(9)|_F} \approx 0.912 \tag{13}$$

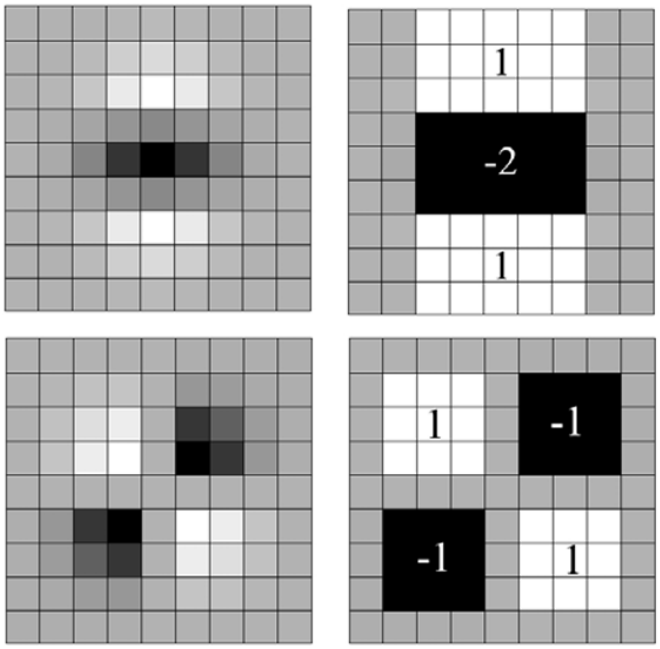


Figure 6: Gaussian Filter (left) vs. Box Filter (right) [31]

The scale space for SURF is also slightly different from SIFT: Instead of reducing the image size repeatedly, the size of box filter can be increased iteratively to analyze the scale space. For instance, a 9×9 box filter is exampled in the last step to form a layer of the pyramid. 15×15, 21×21, and 27×27 filters can be used to set the other layers. Note that when a bigger mask of the box filter is used, the blur scale σ should also be magnified by the same ratio. For example, the 27×27 filter should have $\sigma = \frac{27}{9} \times 1.2 = 3.6$. The advantage of this is that with the relative weight is remained constant since Frobenius norms are scale normalized. Therefore, only one calculation is required among all layers of the filter, which significantly boosts the computation of scale space.

The keypoint localization is similar to what SIFT does, the 26 neighbors of an interest point are used for the comparison to localize the keypoint. Those with weak contrast and on an edge are, again, eliminated to reduce the total number of keypoints.

In terms of the orientation, the Haar wavelet responses of a keypoint are calculated in both x and y direction, denoted d_x and d_y. Same as SIFT, an orientation collection region of 4×4 is used. The responses within this region are summed up to ensure the uniqueness of descriptors. Furthermore, considering the polarity change among intensities, the magnitude of d_x and d_y are also used to make the descriptors recognizable. Figure 7 [30] examples these four indicators Σd_x, Σd_y, $\Sigma|d_x|$ and $\Sigma|d_y|$ over three typical intensity patterns.

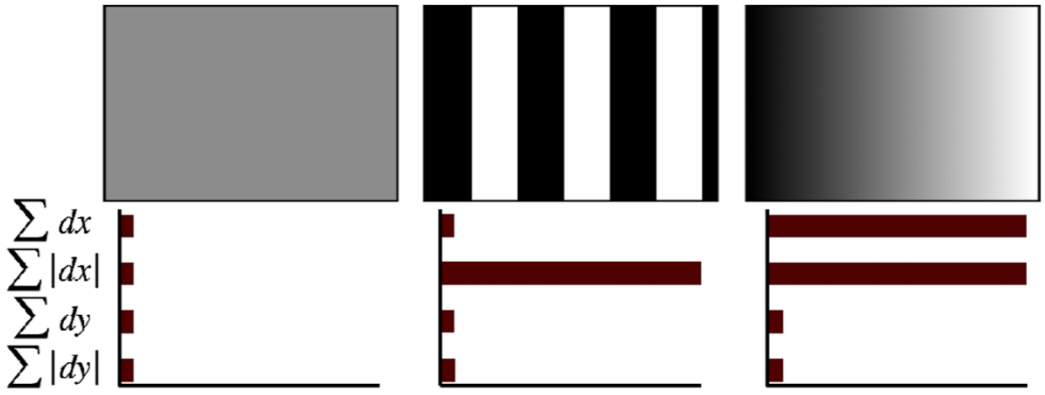


Figure 7: SURF Descriptors against Three Intensity Patterns [30]

To establish a descriptor for each keypoint, a 64-dimensional array ($4 \times 4 \times 4$) is generated. This also explains the higher efficiency that SURF holds compared to the SIFT algorithm.

For the feature matching, the Euclidean distance is computed, which is following the same rule as in the SIFT algorithm. Besides, SURF calculates the trace of Hessian matrix: When these two keypoints being compared have both trace of same sign, no matter positive or negative, these two features are considered to have their contrasts varying in the same direction.

ORB (Oriented FAST and rotated BRIEF) [32] algorithm can also be used to extract features from an input image and create unique descriptors for them, which can be further

used for applications such as image identification. This will be discussed in the later section of ORB-SLAM family. A comparison among all the three algorithms of feature extraction and matching will come in that section as well.

To filter out the outliers among the results computed directly from the camera measurements, the RANSAC algorithm is implemented in the RGB-D SLAM.

Unlike the canonical data fitting method least square, which takes all data points into account, RANSAC is famous for its ability to figure out the inliers and outliers. This is extremely important for the point cloud segmentation of a SLAM system. Although it is not mentioned in RGB-D SLAM, some modern SLAM systems extract objects from the input point clouds, utilizing the a priori knowledge such as the shape, color, and sometimes texture. For example, a desktop on the office desk is very likely to be a block shaped object.

The percentage of the numbers of inliers among all data points is defined as

$$W = \frac{num_{inlier}}{num_{inlier} + num_{outlier}} \tag{14}$$

Selecting n points from the point set, the probability of having at least one outlier in each time of k iteration is $(1 - W^n)^k$.

Thus, the probability of having a correct prediction of model, which means that all points selected are inliers, can be defined as

$$P = 1 - (1 - W^n)^k \tag{15}$$

The number of iterations can therefore be calculated by the following equation.

$$k = \frac{\log(1-P)}{\log(1-W^n)} \tag{16}$$

The values of P, W, and n are set based on experience. For example, having 300 data points selected with inlier rate of 98%, the expected numbers of iterations required to obtain a prediction with 99% correctness is around 2000.

However, RANSAC is limited for single model scenario. If more than one model can be found among data points, Hough transform, or PEARL should be considered.

The iterative closest point (ICP) method is used to estimate the motions by aligning consecutive point clouds and depth frames in RGB-D SLAM. Only point-to-point, i.e. the original version of ICP used in RGB-D SLAM is discussed in this section. The improved versions such as point-to-plane ICP and projective ICP will be included in the later section of ORB-SLAM.

The question is mathematically defined as: given point cloud $\boldsymbol{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \dots, \boldsymbol{p}_n\}$ and its consecutive point cloud $\boldsymbol{Q} = \{\boldsymbol{q}_1, \boldsymbol{q}_2, \dots, \boldsymbol{q}_n\}$ and solve for the camera rotation matrix $\boldsymbol{R}$ and translation matrix $\boldsymbol{t}$ in the following equation.

$$\boldsymbol{q}_i = \boldsymbol{R}\boldsymbol{p}_i + \boldsymbol{t} \tag{17}$$

This equation does not hold all the time due to sensor noises and possible errors in data association process. In this case, this is an optimization problem with object function:

$$\frac{1}{2}\sum_{i=1}^{n} \|\boldsymbol{q}_i - \boldsymbol{R}\boldsymbol{p}_i - \boldsymbol{t}\|^2 \tag{18}$$

A common way of solving this problem is singular value decomposition (SVD).

Giving center of mass for each point clouds, $\boldsymbol{\mu}_p = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{p}_i$ and $\boldsymbol{\mu}_q = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{q}_i$, the displacement of each data point to the center of mass is defined as $\boldsymbol{p}_i' = \boldsymbol{p}_i - \boldsymbol{\mu}_p$ and $\boldsymbol{q}_i' = \boldsymbol{q}_i - \boldsymbol{\mu}_q$, the object function in equation (18) can be rewritten as

$$\frac{1}{2}\sum_{i=1}^{n}\left(\|\boldsymbol{q}_i{}' - \boldsymbol{R}\boldsymbol{p}_i{}'\|^2 + \left\|\boldsymbol{\mu}_q - \boldsymbol{R}\boldsymbol{\mu}_p - \boldsymbol{t}\right\|^2\right) \tag{19}$$

The optimized solution is solved as

$$\boldsymbol{R}^* = \underset{\boldsymbol{R}}{\operatorname{argmin}} \sum_{i=1}^{n} -\boldsymbol{q}_i'^T \boldsymbol{R}\boldsymbol{p}_i{}' \tag{20}$$

$$\boldsymbol{t}^* = \boldsymbol{\mu}_q - \boldsymbol{R}\boldsymbol{\mu}_p \tag{21}$$

To simplify the rotation matrix, let $\boldsymbol{W} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{q}_i'\boldsymbol{p}_i'^T$.

From SVD, for a full rank matrix $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, where $\boldsymbol{\Sigma}$ is a square diagonal matrix, the combination of $\boldsymbol{U}$ and $\boldsymbol{V}$ matrices exists and is unique. Therefore, the optimized rotation matrix in equation (20) can be rewritten as

$$\boldsymbol{R}^* = \boldsymbol{U}\boldsymbol{V}^T \tag{22}$$

2.2 ORB-SLAM Family

ORB-SLAM family includes three generations and has now become one of the most accurate real-time visual SLAM systems without the adoption of deep learning methods. Inspired by PTAM [12], the first generation ORB-SLAM [15] enhanced efficient real-time operation in large-scale environment. It is the first SLAM system utilizes ORB features for extraction and matching. Dividing the SLAM task into tracking, local mapping, global relocalization and loop closing, ORB-SLAM enables efficient computation without GPU acceleration. Besides, the system robustness is significantly enhanced by its unique initialization process, map creation method, and keyframe selection policy.

A schematic overview of ORB-SLAM is included below as Figure 8.

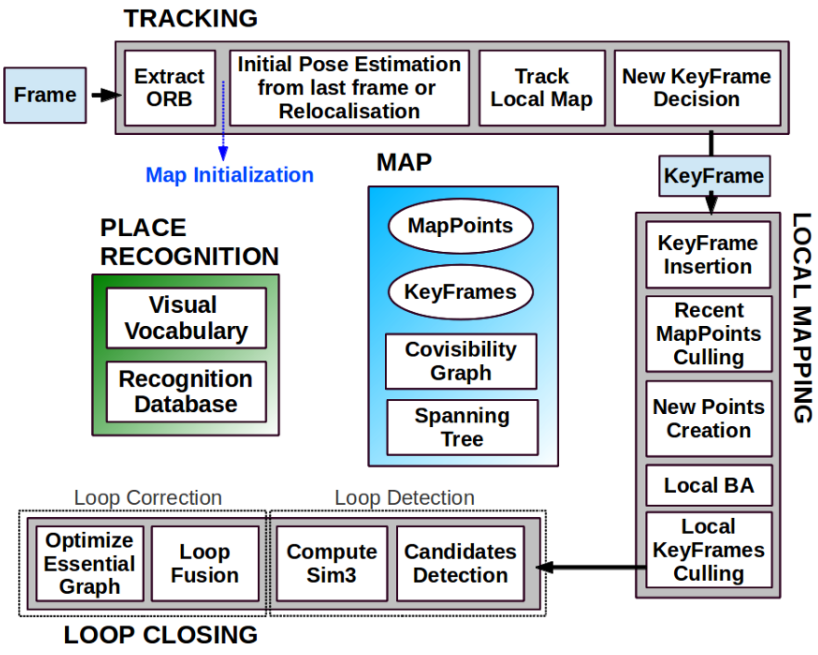


Figure 8: Schematic Overview of ORB-SLAM [15]

As a beginning version, ORB-SLAM also has its limitations. First, implemented with only monocular camera configuration, the depth estimations could be challenging compared to

the other systems using RGB-D or stereo configurations. Besides, the loop closing method at that time can still become struggling under certain conditions. This is said to happen more frequently in complex and repetitive environments. Lastly, the computational cost is relatively high due to the nature of ORB features.

Considering of all the deficiencies, ORB-SLAM2 [16] was proposed by the same group of researchers after two years. Comparing to their previous work, improvements are made from multiple aspects. First of all, ROS platform becomes optional, the SLAM system can be performed and visualized by Pangolin package instead. Secondly, not only monocular but also stereo and RGB-D cameras are supported in its framework. And as the result, the initialization process is redesigned as preprocessing since the depth information can be either directly obtained from an RGB-D camera or calculated through disparities from a stereo camera. The scale information of objects, in this case, is calculated from rigid transformation instead of similarities therefore being more accurate. This also fixes failures from the drifting as well as during rotations, caused by the nature of monocular cameras. Moreover, a full bundle adjustment is implemented to ORB-SLAM2 after its loop detection for enhanced mapping accuracy. Lastly, a localization mode is implemented to the SLAM system. Other than the normal operation which creates a map while localizing the robot simultaneously, it is capable for operations in a known environment where a map has already been generated and can be loaded as input of the system. It is introduced that the local mapping and loop closing threads of the system are suspended under this situation; the relocalization thread is processed alone. The tracking of robot acts similar as a visual odometry system at this moment.

A schematic overview of ORB-SLAM2 is included below as Figure 9.

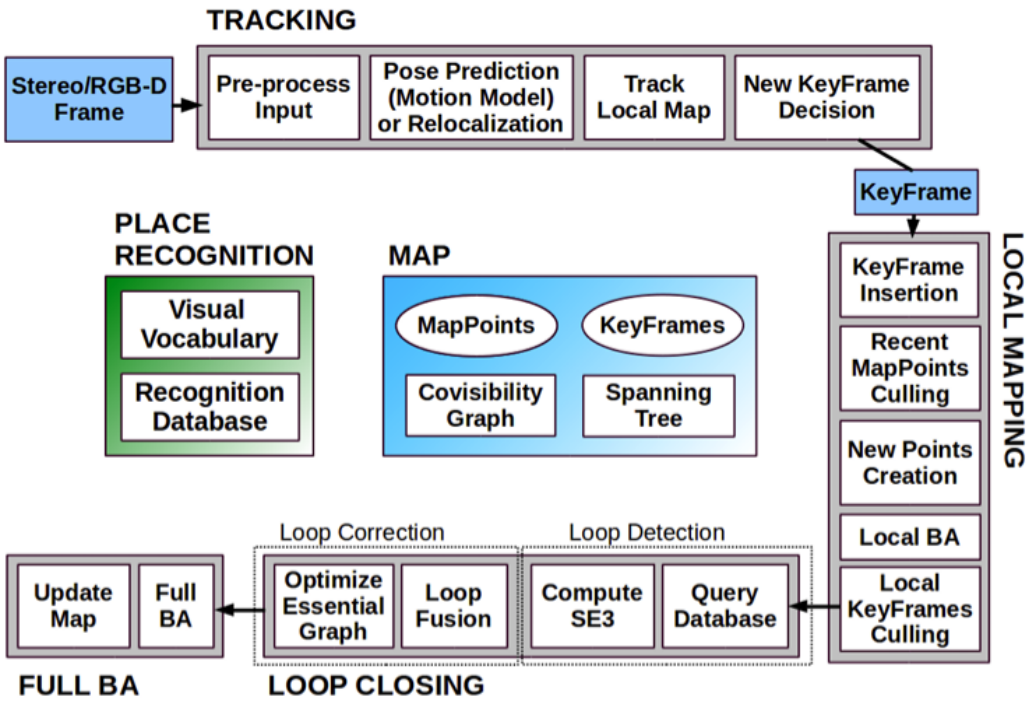


Figure 9: Schematic Overview of ORB-SLAM2 [16]

ORB-SLAM2 also has limitations compared to the other SLAM algorithms during the same period. One of the principal concerns raised by researchers is the lack of sensor fusion. There are multiple algorithms proposed at that time adopting information from multiple sensors. Lidar, GPS, and IMU sensors are frequently used as compensations for the camera in challenging situations to improve the system robustness in complex environment.

ORB-SLAM3 [19], the latest generation of this family, introduces multiple improvements based on the inadequacy explained earlier. Firstly, it supports IMU, the inertial sensor as an optional input to enhance the robustness. It is said that the system accuracy is also improved relying on Maximum-a-Posteriori (MAP) estimations. Besides, an advanced place recognition algorithm is proposed in ORB-SLAM3 to provide better system accuracy

with a slightly increased computational cost. Furthermore, integrating ORB-SLAM Atlas [20], ORB-SLAM3 has become a comprehensive multi-map SLAM system.

A schematic overview of ORB-SLAM3 is included below as Figure 10.

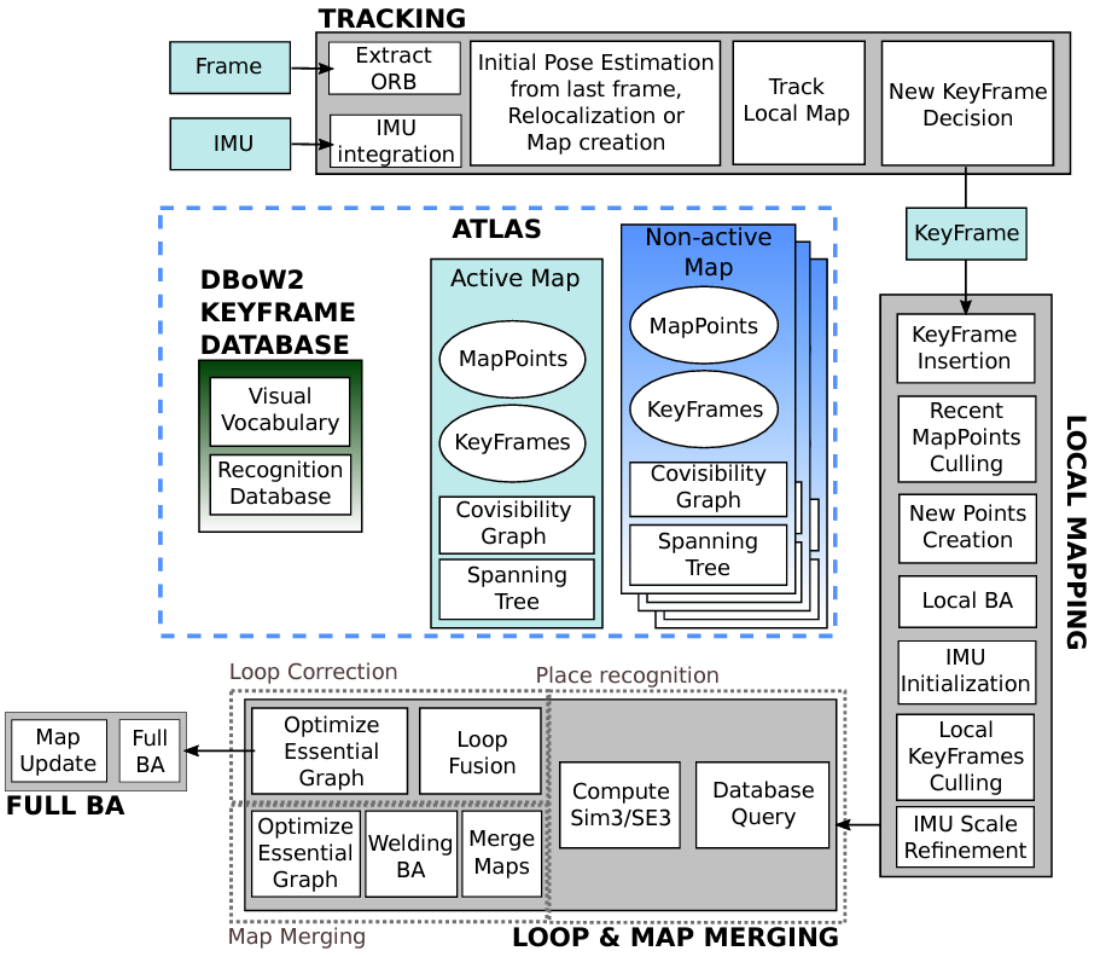


Figure 10: Schematic Overview of ORB-SLAM3 [19]

The following part in this section will explain how ORB features can be extracted and matched. The pros and cons of ORB feature alongside the two other feature extraction and matching methods discussed in the previous section will be summarized in this part as well.

ORB (Oriented FAST and rotated BRIEF) [32] is a combination and upgrade of a feature extraction algorithm FAST (Features from Accelerated Segment Test) and a feature descriptor BRIEF (Binary Robust Independent Elementary Features).

FAST takes 16 pixels around the interested pixel, also called a patch of interest: one pixel that are 2 pixels away along each of the intercardinal directions (a total of 4), and one pixel that is approximately 3 pixels distance along each of the cardinal directions (a total of 4), and the two pixels next to the ones in the cardinal directions (a total of 8). These 16 pixels form a circle around the interested point, also known as a Bresenham's circle of radius 3. If more than half of these pixels have intensities l_i ($i \leq 16, i \in \mathbb{Z}^+$) are beyond a threshold ($l_i \geq l_p + h$ or $l_i \leq l_p - h$, where h is the preset threshold), the interest point of intensity l_p is considered as a keypoint.

FAST further improves this step by only comparing the four pixels in the cardinal directions. It is discovered that only if two or more pixels among these four satisfy the previously mentioned criteria, that the interest point is possible to be a keypoint. This reduces the process time significantly, to approximately one fourth.

To detect those interest points in adjacent locations and determine which one suits better, a non-maximal suppression is applied. It is explained that a score function will be computed in this case:

$$V = \sum_{i=1}^{i \leq 16} \left| l_p - l_i \right| \tag{23}$$

The interest point with lesser score is eliminated.

Different from SIFT and SURF algorithms, FAST detects the regions with rapid contrast change, which typically indicates an edge. The green marks in Figure 11 below represents part of the features detected. It can be seen that the keypoints are aligned with the joints of pipes, a ladder, and the frames of baffle boards.

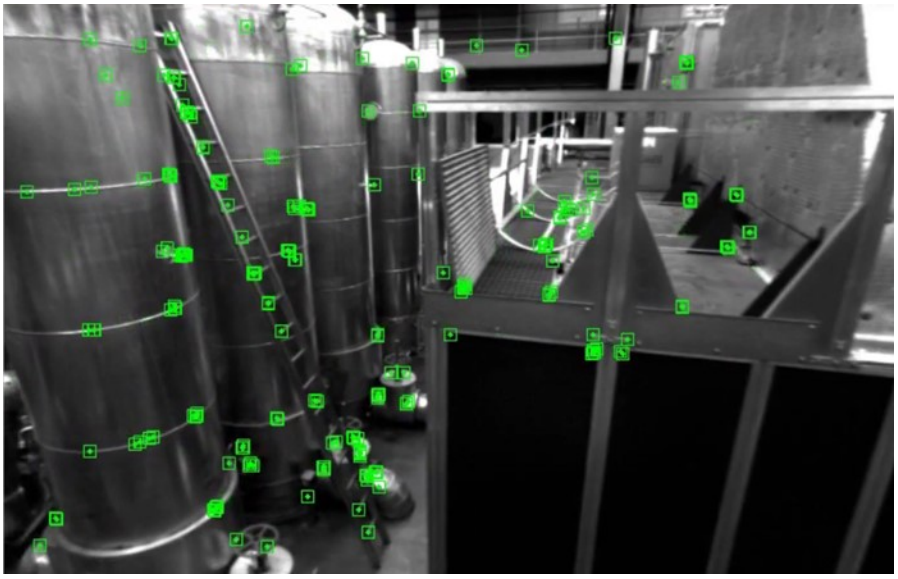


Figure 11: Features Extracted with FAST Algorithm

ORB further apply an orientation assignment to FAST, named oriented FAST. It assigns a direction for each keypoint, depending on the intensities of neighbor pixels. A vector is established pointing from the geometric center, which is the location of the keypoint itself, to the intensity center of its neighbors. The location of the intensity center as well as this orientation can be calculated with the moment of this patch.

In terms of the BREIF descriptor, it starts from blurring the image, using the Gaussian kernel to prevent the descriptor being sensitive to noises, same as SIFT. After that, a pair of pixels is selected randomly among these neighbors. The first point of the pairs is selected from a Gaussian distribution centered around the keypoint with a standard deviation of σ.

The second point is selected within a standard deviation of $\sigma/2$ centered around the first point. It has been proven that this will increase the feature matching rate. The intensities of these two points are compared to determine a binary value, which is used as one digit of the descriptor. Representing with math:

$$\tau(\boldsymbol{p}; \boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 : \boldsymbol{p}(\boldsymbol{x}) \leq p(\boldsymbol{y}) \\ 0 : \boldsymbol{p}(\boldsymbol{x}) > p(\boldsymbol{y}) \end{cases} \tag{24}$$

where $\boldsymbol{p}(\boldsymbol{x})$ and $\boldsymbol{p}(\boldsymbol{y})$ are the intensities of the first and second point, respectively. $\boldsymbol{x}$ and $\boldsymbol{y}$ are the pixels selected, defined by the image coordinates $\boldsymbol{x} = (u, v)^T$.

This intensity test is performed repeatedly for n pairs of random points, typically $n = 128, 256\ or\ 512$, so that these binary values form up a unique bitstring $f(n)$of the set length, that is the BRIEF descriptor. Representing with math:

$$f(n) = \sum_{1<i<n} 2^{n-1}\, \tau(\boldsymbol{p}; \boldsymbol{x_i}, \boldsymbol{y_i}) \tag{25}$$

ORB algorithm also endows the BRIEF descriptor with rotation invariance, named rotated BRIEF. Instead of rotating the entire patch around each keypoint, only the selected BRIEF pairs are rotated, so that the computation cost is reduced. The location of selected pairs can be represented by:

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{x_1} & \cdots & \boldsymbol{x_n} \\ \boldsymbol{y_1} & \cdots & \boldsymbol{y_n} \end{bmatrix} \tag{26}$$

The location of rotation invariant pairs can be represented by:

$$\boldsymbol{S_\theta} = \boldsymbol{R_\theta S} \tag{27}$$

where θ comes from the orientation assignments, indicating the direction from geometric center to the intensity center of the patch. $\boldsymbol{R_\theta}$ is the rotation matrix relating the original direction with the orientation of intensity center.

By this way, the location of the detected pairs can always be transformed into a normalized direction before comparing with the location of another set of random pairs that are in the same direction. Thus, the rotation invariance of the descriptor can be ensured.

Table II below exhibits a comparison of the performances among these three algorithms mentioned in this section.

Table II: Comparison of Three Feature Extraction and Matching Algorithms

Algorithm Name	**SIFT**	**SURF**	**ORB**
Computation Efficiency	+	++	+++
Rotation Robustness	+++	++	++
Scale Robustness	+++	++	+

It can be concluded that when the application demands high real-time performance, the ORB algorithm is preferred. However, it only provides partial scale invariance, indicating that the images are preferred to be taken from the right in front of the objects in order to achieve its best performance. The SURF algorithm not only solves this scale invariance problem but also makes progress on the rotation robustness. Besides, as an improved version of the SIFT algorithm, SURF computes the features in a more time efficiency way, contributed by its operational simplicity explained earlier. Although SIFT is the most time-

consuming algorithm, it has the best rotation and scale robustness. It is better to be applied to those demand high accuracy but without the need of real-time operation.

ORB-SLAM3 also has state-of-the-art optimization methods, including pose graph optimization, bundle adjustment, and loop closure optimization. One of the most critical keys to build an accurate SLAM system is the bundle adjustment. Therefore, a brief summary of the bundle adjustment method is included below.

Bundle adjustment aims to optimize the camera pose estimations and the coordinates of keypoints. The core issue is to solve a problem minimizing the reprojection error, which can also be considered as a nonlinear least square problem, defined as:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} \left[u_{ij} - \pi(C_j - X_i)\right]^2 \tag{28}$$

where u_{ij} is the pixel coordinate of observed ith point X_i by jth camera C_j, $\pi(\cdot)$ is the reprojection function, which is nonlinear.

There are multiple ways to solve this problem. The old-fashioned ways are based on gradient descent and Newton's method. However, gradient descent method cannot achieve fast converging; Newton's requires to calculate the computing expensive Hessian matrix and cannot ensure a steady descent of gradient. Gauss-Newton improves the solving to some extent. The calculation of Hessian matrix is avoided but the descent of gradient can still not be guaranteed. A powerful tool commonly used today is called Levenberg-Marquardt method, which is developed by combining gradient descent with Gauss-Newton. This method ensures a fast convergence with descending gradient by tuning a parameter λ.

CHAPTER 3 SYSTEM ARCHITECTURE

The proposed system is built on ORB-SLAM3 [19]. Figure 12 dissects the main system components, that a new working thread named *Dense Reconstruction* is created beyond the original system, circled by the red dashed box.

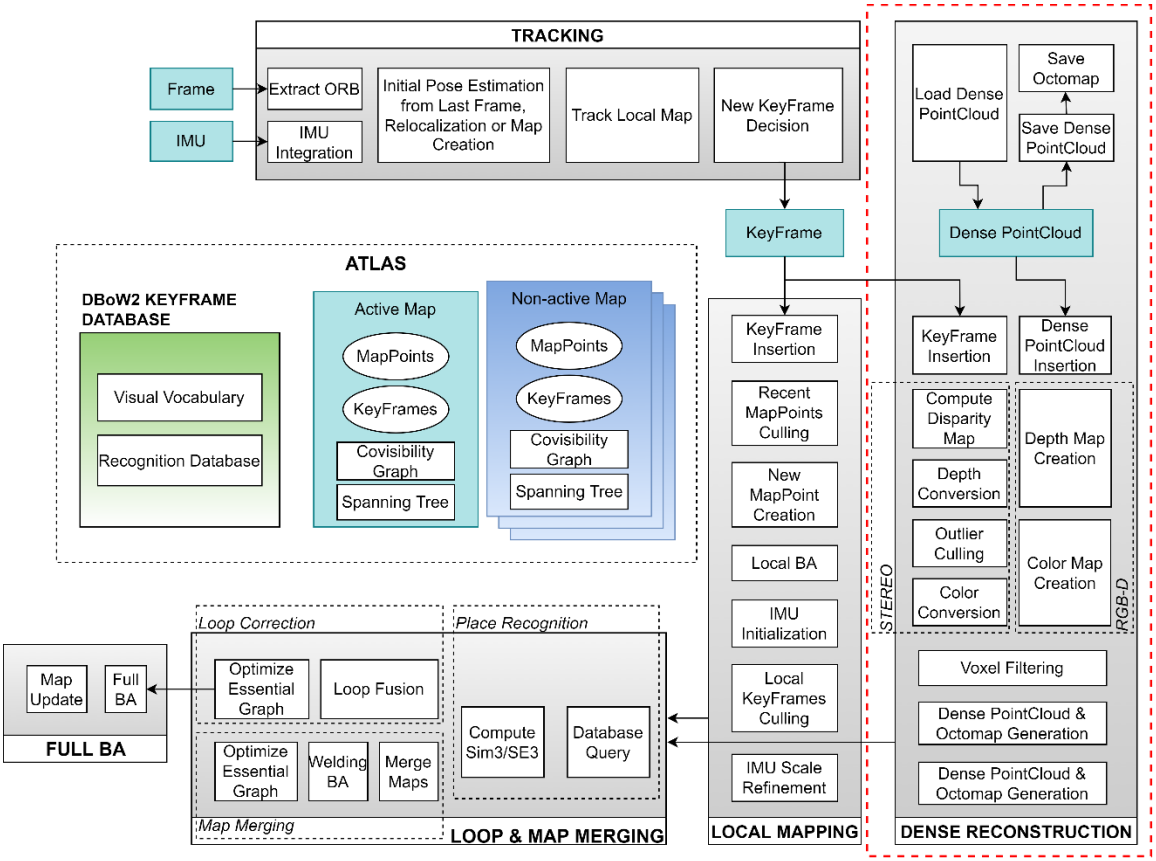


Figure 12: Architecture of the Proposed SLAM System

Dense Reconstruction thread takes two inputs. The first one is keyframes, decided by the tracking thread. This frame can be a combination of color image and depth image, provided by an RGB-D camera; or it can be two images from both views of a stereo camera. The following working process differs depending on the type of camera.

The second input is optional. Leveraging the Atlas subsystem and configure by user, when enabling the multi-sequence merging mode, a dense point cloud from the previous sequence(s) is expected to be input, carrying the keyframes, map points, as well as their connections.

The following stage generates a depth map and a color map for each keyframe. Our purpose is to obtain the red, green, and blue intensity of each pixel along with its distance to camera; therefore, the pixel can be converted to a map point in the dense point cloud. This is straightforward when using an RGB-D camera as all the information can be read directly from the image captured.

For a stereo camera, yet, we have to take additional considerations. To obtain the depth from corresponding left and right views of a keyframe, we first compute a disparity map. However, it is quickly discovered that not all pixels, even in keyframes, brings useful information. For example, some pixels can be too obscure to give reliable calculations, and some can have beyond reasonable distances. These outliers introduce perceptible errors when converting their disparities to depths. These pixels are mostly located on the periphery of the image; hence, a region of interest is chosen, and only those pixels within are used to compute the disparity.

In terms of creating a color map from stereo images, limited by the size of dataset, many stereo images are provided grayscale. This is not sufficient enough to generate a recognizable 3D scene. The demand of better visualization necessitates a color assignment based on depth.

Since the system is designed to process multiple sequences of the same scene, it is revealed that the same feature can be repeatedly detected, during one dataset and, specifically, cross-

datasets. The same map point can, therefore, be assigned to the point cloud for multiple times. However, a later estimation does not mean it is inevitably more accurate. For example, a map point corresponding to a center pixel of the image was detected at a distance of 1 meter from the camera and was then added to the point cloud. After 20 seconds, the same point is detected again, but now from 5 meters away, and the corresponding pixel is located at the periphery of the interest region in the image. We are cognizant of the fact that the latter presence, in this case, should be treated as less accurate, in accordance with the epipolar geometry.

With regard to this, we register a probability to each map point in the dense point cloud as its confidence. It is initialized when a map point is being added to the dense point cloud. A modified sigmoid function is used to assign this probability. The less depth a map point has, the higher confidence on its accuracy we can have. This probability is updated only when two map points are merged with the following rule: when a map point is being added to the dense map, if another map point already exists within a preset spatial distance, then they are treated as the same point. Accordingly, the pending map point combines with the existing one, and being replaced by a new map point at a location calculated from their probabilities of confidence. This prevents those inaccurate latter estimations from ruining the previous trust-worthy point cloud map.

To demonstrate the real-time reconstruction, we further process the dense point cloud to a voxel grid filter before visualization. The redundant points in the dense point cloud are removed. The point cloud is sparsified at a user-configurable resolution such that only desired number of points are preserved in the map, thereby reducing the CPU workload. The dense point cloud is also converted to an Octomap, which represents the scene with a

compact 3D model. Both the dense point cloud map and the Octomap are saved before exiting for potential uses afterwards.

The following sections in this chapter will detail the improvements mentioned above.

3.1 Camera Model

As discussed in the literature review, ORB-SLAM3 efficiently identifies the ORB features and accurately produces a semi-dense map in real-time. We further utilize these keyframes determined by the original system to provide an additional function of constructing dense point cloud maps with either RGB-D or stereo images. Furthermore, this dense point cloud is converted into an Octomap for possible subsequent operations. A pinhole camera model is assumed during the reconstruction for simplicity.

For a point $P(X_w, Y_w, Z_w)$ in the world coordinate system, its projection onto the image coordinate system, denoted as (u, v), can be calculated with a homogeneous expression:

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{K} \begin{bmatrix} \boldsymbol{R}_{3\times3} & \boldsymbol{t}_{3\times1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{29}$$

where z_c is the distance of the point from image; $\boldsymbol{R}$ and $\boldsymbol{t}$ are the rotation and translation matrices that form up the camera extrinsic. $\boldsymbol{K}$ is the camera intrinsic matrix defined with the focal length, f_x, f_y, the principal point offset c_x, c_y, horizontally and vertically, and the axis skew coefficient s. These camera parameters are directly input to the system from a configuration file.

Reconstruction of map points, which is detailed below, reverses the calculation of equation (29). The processes using RGB-D and stereo camera, however, differ significantly since

the depth z_c cannot be obtained directly from stereo images. For this reason, they are discussed separately in their respective sections.

3.2 Dense Reconstruction from RGB-D Camera

Generating a point cloud from RGB-D camera is relatively simple. This process is formally presented in Algorithm 1.

Algorithm 1: RGB-D Point Cloud Generation

Input: Focal length f_x, f_y, Camera center c_x, c_y
Set of frames with poses $\mathbb{S}_f = \{F^i | \mathbf{T}_w^c \mapsto F\}$,
$i \in \mathbb{Z}^+$
Set of images $\mathbb{S}_t = \{\mathbf{M}_t{}^i\}, t \in \{'c', 'd'\}$
$\mathbf{M}_c \in [0, 255]^{u \times 3v}$; $\mathbf{M}_d \in \mathbb{R}^{u \times v}$
Two empty queues Q_c and Q_d
Output: Set of map points $\mathbb{S}_P = \bigcup_{i_f \in \mathbb{Z}^+} (\mathbf{T}_w^c \mathbb{P}_c)^{i_f}$
Set of points in a keyframe $\mathbb{P}_c = \{\mathbf{P}^{i_p}\}, i_p \in \mathbb{Z}^+$
Map point $\mathbf{P} = [x, y, z, r, g, b]$

```
do in parallel
    for ∀F ∈ 𝕊_f do
        if F^n is decided to be a keyframe then
            Q_c ← M_c^n, Q_d ← M_d^n
do in parallel
    i_f ← 0
    while Q_c ≠ ∅ do
        M_c^now ← Q_c.pop_front
        M_d^now ← Q_d.pop_front
        i_p ← 0
        for row ← 1 to u do
            for col ← 1 to v do
                z ← M_d^now[row][col]
                x ← (row − c_x) × z/f_x
                y ← (col − c_y) × z/f_y
                r ← M_c^now[row][3 × col]
                g ← M_c^now[row][3 × col − 1]
                b ← M_c^now[row][3 × col − 2]
                ℙ_c^{i_f} ← P^{i_p} ← [x, y, z, r, g, b]
                i_p = i_p + 1
        𝕊_ℙ ← T_w^{c now} ℙ_c^{i_f}
        i_f = i_f + 1
```

Figure 13: Algorithm 1 for RGB-D Point Cloud Generation

The set of input frames are bijective with either the set of color images or the depth images. Therefore, the frame ID, n, can be used to access all these three sets, $\mathbb{S}_f$, $\mathbb{S}_c$ and $\mathbb{S}_d$. Therefore, if a frame is determined as keyframe in the Tracking thread, a correlated pair of color image and depth image is added to queue. The decision of keyframe is made, according to certain rules that have been introduced in the original system. On the other hand, the Dense Reconstruction thread pops out from the front of the queue with a first come first serve manner. The world position for each pixel is calculated with the pinhole model, illustrated in Figure 14.

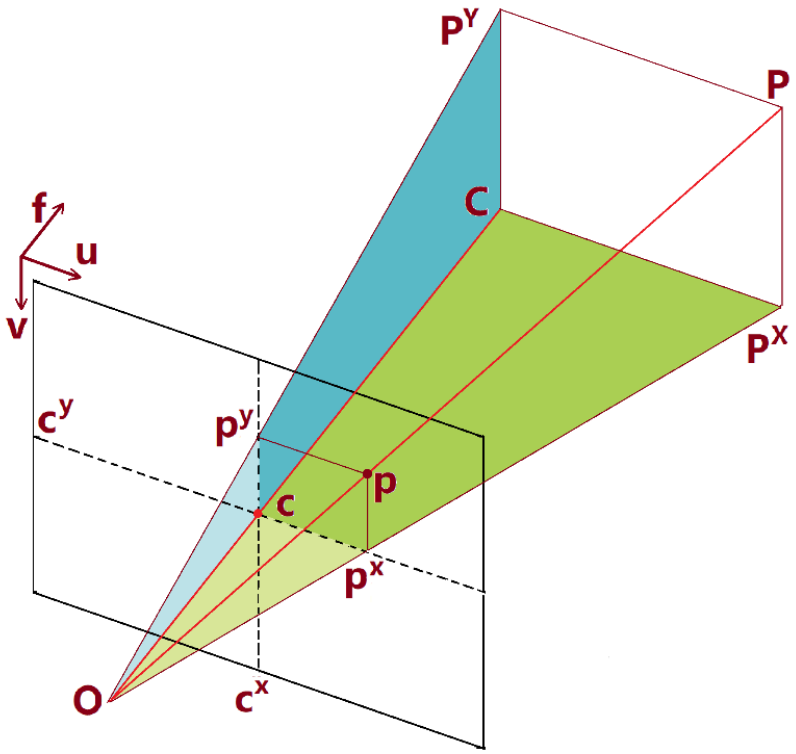


Figure 14: Illustrative Diagram of RGB-D Camera Model

Assume a point $P(P^X, P^Y, P^Z)$ — in the camera coordinate system, with origin at the focal point O — is projected to the image plane at $p(p^x, p^y, p^z)$, where P^Z is its distance to the camera, from C to O. Using an RGB-D camera, this can be directly measured, so being

used as *line 13* in Algorithm 1. p^z is also a known value, which is the distance from c to O. It measures from the image plane to the focal point, which is the focal length f. The x and y coordinate of this point can be calculated separately, each with a pair of similar triangles. Horizontally, the first triangle is formed by O, c and the point where p^x is marked in the figure; and the second triangle is within O, C and the point where P^X is marked. They are filled with light and dark blue, respectively. Likewise, another pair of triangles is formed vertically. These pair of triangles are in green. From these similarities, the equations in *line 14* and *15* are established.

Concurrently, color in blue, green, and red intensities are extracted from $\boldsymbol{M}_c$ and assigned to each point. This process iterates whenever the image queue is not empty and ends by the end of program when thread is killed. *Line 19* stores the converted map points of a single keyframe to $\mathbb{P}_c$, that is in camera coordinate system. *Line 21* converts these points to world coordinate system with a transformation matrix $\boldsymbol{T}_w^c$ calculated and optimized by the original system, then updates the dense point cloud $\mathbb{S}_P$ with all nonexistent points.

These map points are recorded in the form of voxels. For a 640 by 480 image, more than 300,000 voxels are generated, causing unnecessary computational burden if all of them are stored into the dense map. A reasonable solution to this issue is to use a voxel filter, which sieves the close voxels and replaces them with only one voxel. This filter can operate at a preset resolution. In order to ensure real-time performance most of the time in the experiments, more than 90% of the voxels are removed. They are stored into a *.pcd* file for reuse purpose as well as the Octomap conversion, which will be discussed in the latter section.

3.3 Dense Reconstruction from Stereo Camera

Generating a point cloud from stereo camera requires extra steps compared to the previous discussed RGB-D reconstruction. This process is formally presented in Algorithm 2.

Algorithm 2: Stereo Point Cloud Generation

Input: Focal length f_x, f_y, Camera center c_x, c_y
Camera baseline b, Calibration matrices $\mathbf{M}_{cl}$ $\mathbf{M}_{cr}$
Set of frames with poses $\mathbb{S}_f = \{F^i | \mathbf{T}_w^c \mapsto F\}$,
$i \in \mathbb{Z}^+$
Set of images $\mathbb{S}_t = \{\mathbf{M}_t{}^i\}, t \in \{'l', 'r'\}$
$\mathbf{M}_t \in \mathbb{R}^{u \times v}$; Two empty queues Q_l and Q_r
Output: Set of map points $\mathbb{S}_P = \bigcup_{i_f \in \mathbb{Z}^+} (\mathbf{T}_w^c \mathbb{P}_c)^{i_f}$
Set of points in a keyframe $\mathbb{P}_c = \{\mathbf{P}^{i_p}\}, i_p \in \mathbb{Z}^+$
Map point $\mathbf{P} = [x, y, z, r, g, b]$

do in parallel
  for $\forall F \in S_f$ **do**
    if F^n *is decided to be a keyframe* **then**
      $[\mathbf{M}_l{}^n, \mathbf{M}_r{}^n, ROI^n] \leftarrow$ **call** *Rectification*$(\mathbf{M}_{cl}, \mathbf{M}_{cr})$
      $Q_l \leftarrow \mathbf{M}_l{}^n$, $Q_r \leftarrow \mathbf{M}_r{}^n$
do in parallel
  $i_f \leftarrow 0$
  while $Q_l \neq \varnothing$ **do**
    $\mathbf{M}_l{}^{now} \leftarrow Q_l.pop\_front$
    $\mathbf{M}_r{}^{now} \leftarrow Q_r.pop\_front$
    $\mathbf{d} \leftarrow$ **call** *Disparity*$(\mathbf{M}_l{}^{now}, \mathbf{M}_r{}^{now})$
    color $\leftarrow$ **call** *Colorization*$(\mathbf{d})$
    $[umin, umax, vmin, vmax] \leftarrow ROI^{now}$
    $i_p \leftarrow 0$
    for $row \leftarrow umin$ **to** $umax$ **do**
      for $col \leftarrow vmin$ **to** $vmax$ **do**
        $z \leftarrow b \times f_x / \mathbf{d}[row][col]$
        $x \leftarrow (row - c_x) \times z / f_x$
        $y \leftarrow (col - c_y) \times z / f_y$
        $r \leftarrow$ **color**$[row][3 \times col]$
        $g \leftarrow$ **color**$[row][3 \times col - 1]$
        $b \leftarrow$ **color**$[row][3 \times col - 2]$
        $\mathbb{P}_c{}^{i_f} \leftarrow \mathbf{P}^{i_p} \leftarrow [x, y, z, r, g, b]$
        $i_p = i_p + 1$
    $\mathbb{S}_{\mathbb{P}} \leftarrow \mathbf{T}_w^{c\ now} \mathbb{P}_c{}^{i_f}$
    $i_f = i_f + 1$

Figure 15: Algorithm 2 for Stereo Point Cloud Generation

Unlike the RGB-D camera which can obtain the depth directly, the concept of disparity, which is introduced in epipolar geometry, is required to calculate depth from a stereo camera. However, we need to rectify both views in advance. In this step, functions in the *OpenCV* [28] library are used to undistort, calibrate, and crop the image: the epipolar is moved to infinity, the same pixels are aligned horizontally as much as possible, and invalid regions are removed. This is done in *line* 4 of Algorithm 2, where ROI stands for the region of interest, which consists of the upper and lower bounds of the image in both horizontal and vertical directions.

Disparity indicates the distance of a point P between the left and right views, in the unit of pixels. As shown in Figure 16, the point P is projected onto the left image plane at p_L, having horizontal coordinate of p_L^x; and for the right image plane, p_R^x. As aforementioned, only horizontal disparity $p_L^x - p_R^x$ needs to be considered after calibration. In *line 11* of the algorithm, disparities are calculated for all pixels with the calibration matrices $\boldsymbol{M}_{cl}$ and $\boldsymbol{M}_{cr}$. They are stored in consistent with the input image in $\boldsymbol{d}$, which is a matrix of the same size as image.

The next step is to convert the disparity to depth with geometric similarity. Note that after the calibration, the two "curves" $O_L O_R$ and $p_L^x p_R^x$ in blue are line segments, they seem to be curves due to one-point perspective. The length of baseline is defined as b; the distance between the two points marked p_L^x and p_R^x is $b - p_L^x c_L + p_R^x c_R$, since O_L and O_R are projected at the center of each image. The two triangles formed by the point marked P_L^X and each of the two segments are similar. Thus, the equation in *line 17* can be applied to find the length of the blue line segment perpendicular to baseline, which is the depth of point P.

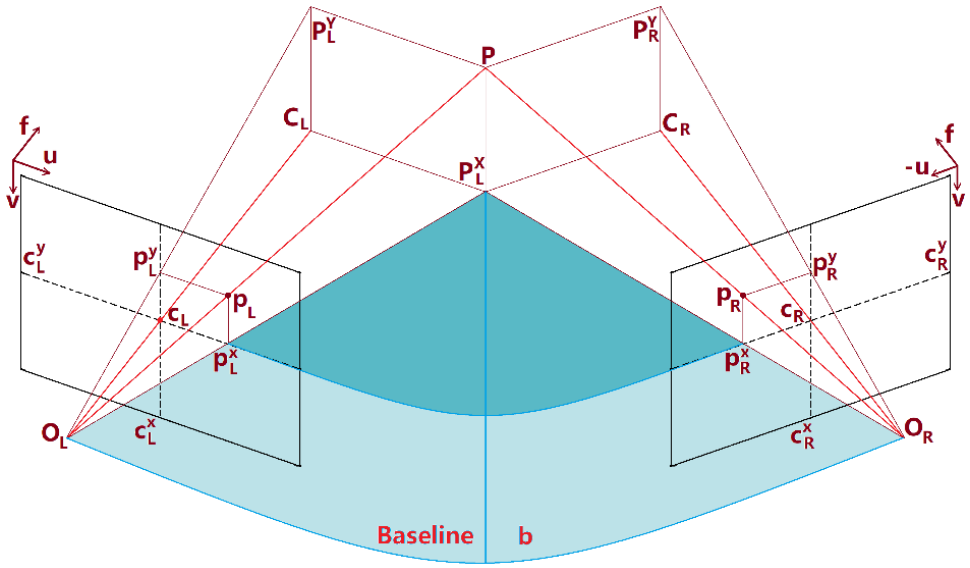


Figure 16: Illustrative Diagram of Stereo Camera Model

As for colors, the grayscale images have same intensities for all three channels: red, green, and blue. For better visualization, *line 12* assigns all pixels with colors based on their depths. The colorization is coded as an optional function intentionally. The alternative option assigns all map points with a single solid color, defined by user before the program starts. Having this option benefits the visualization when running multiple sequences in the same scene.

3.4 Octomap from Dense Point Cloud

The previous section outlines the process of generating a dense point cloud map. This section introduces an alternative mapping method, Octomap, its necessity and implementation.

One major limit for the dense point cloud map is that too much space is consumed after mapping for a sizable dataset, even after those redundant points being filtered. Depending

on the application, these dense maps can sometimes be needlessly detailed in textures and shadings. On contrary, Octomap have its storage cost prominently reduced by using Octree, a tree structure, which partitions the space into multiple cubic voxels. Each of them, being considered as a parent node, can be further divided into eight child nodes, halving in all three directions.

This step is repeated for multiple times until a satisfactory resolution is reached. The tiniest voxels after division, called leaf nodes, are assigned with float numbers to represent the probabilities of their positions being occupied. This probability is initialized with 0.5 and always in the $[0,1]$ interval. The larger this number is, a higher degree of certainty we have that this voxel has already been occupied. To visualize the map, all unoccupied and undetermined leaf nodes are rendered in fully transparent while those occupied are rendered with colors to form the 3D model. Different from the dense point cloud, Octomap has a compact structure, so that it can be directly used by mobile manipulation, navigation, and other robotic applications.

Furthermore, this tree structure guarantees an outstanding computing efficiency and demands less space for storing. To be more specific, all eight child nodes under a parent node are pruned if they are assigned with same states, i.e., "unoccupied", "undetermined" or "occupied", since the parent node itself is sufficient enough to describe this volumetric space. The map structure is greatly simplified in such manner.

Introduced by the author in [23], $P(n|z_{1:t})$, the probability that a leaf node n has been occupied at time t, with the sensor measurements $z_{1:t}$ given, is updated by:

$$P(n|z_{1:t}) = \left[1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \frac{1 - P(n|z_{1:t-1})}{P(n|z_{1:t-1})} \frac{P(n)}{1 - P(n)}\right]^{-1} \tag{30}$$

The $logit$ function maps the probabilities from $P(n) \in (0,1)$ to $\alpha(n) \in R$. When a probability is above 0.5, it is mapped to a positive number, and vice versa. The transformation is defined as:

$$\alpha = logit(P) = \ln\left(\frac{P}{1-P}\right) \tag{31}$$

The inverse function of $logit$ is $sigmoid$, defined as:

$$P = sigmoid(\alpha) = logit^{-1}(\alpha) = \frac{1}{1+e^{-\alpha}} \tag{32}$$

To avoid the calculations with astronomical numbers, upper and lower limits, l_{max} and l_{min}, for the $logit$ transformation are defined. Applying equation (32) to equation (31) results:

$$L(n|z_{1:t}) = \max(\min(L(n|z_{1:t-1}) + L(n|z_t), l_{max}), l_{min}) \tag{33}$$

where $L(n|z) = \ln\left(\frac{P(n|z)}{1-P(n|z)}\right)$.

This equation dissects the update of probability at leaf node n after receiving a new observation z_t, showing the fact that the update is based on accumulations. The current probability is mapped with equation (31) to make it accumulative, before adding it to the existing previous probabilities. To concludes the update, the probability is mapped back to the $[0,1]$ interval with equation (32). For instance, if a point has been regularly measured to be unoccupied, its estimations are correspondingly to be negative. The initial estimation at zero, mapped from a probability of 0.5, is added with these negative estimations and becomes more negative before the minimum limit is reached. This negative number with

huge magnitude, at the end, maps to a near-zero probability, indicating this voxel is very likely to be unoccupied.

This approach is implemented to the proposed system utilizing some predefined functions in the Octomap library that calculates and generates the Octomap with the dense point cloud.

3.5 Multi-Sequence Merging

The demands for additional computing resources have been recognized when constructing a 3D dense point cloud. This cannot be avoided due to the nature of dense reconstruction, computing pixel-wise always takes longer. To minimizing the negative impact, we would like to have the function of reutilizing the map and point cloud. As a result of this, we are able to fragment a huge sequence; Regaining the full view by merging multiple tinier sequences have become possible. This function is designed as optional since not in all cases that it is required.

The map saving and loading functions in the proposed system is refined so that all data can be accessed across datasets. With this approach, it is able to run multiple partially overlapping sequences in the same scene and provide probabilistic updates to map points in the dense point cloud, therefore, constructing a reliable map to enhance the tracking accuracy. In the previous section, we briefly mentioned that when a map point is added to the dense point cloud, its position is corrected according to a probability of confidence. This section explains this approach exhaustively.

As mentioned earlier, one of the key steps when generating the dense point cloud map is to obtain the depths of all map points. These depths are then transformed into world positions by multiplications with the transformation matrices. Regardless of the RGB-D camera that can directly obtain the depth, when using a stereo camera, the calculated depths are obviously not fully accurate since they are based primarily on the camera model and the epipolar geometry, both of which have made assumptions for simplicity.

For stereopsis vision, each increment of the disparity suggests a closer corresponding position in space. However, it is discovered from *line 17* of Algorithm 2 that the relationship between disparity and depth of a map point is not linear but inversely proportional, implying that the possible values of depth are denser for those points closer to the camera. This fact provokes the idea that measurements should not be treated as equals. The sparse depth values, for those map points at far, can lead to increased measurement error. Observations closer to the camera should thus be considered more credible.

The closest and farthest depth, d_{min} and d_{max}, establish an acceptable range for map points. Any map point within this acceptable range will be assigned with a probability, P, before joining the dense point cloud map. This probability represents the level of confidence we have to its sensor measurement. From a practical point of view, assigning full confidence to a map point at the minimum distance is witless, resulting it no longer updatable. Conversely, setting zero trust to a point at the maximum distance is also imprudent, exerting insufficient effect on the map update. Since the ideal values of d_{min} and d_{max} differ from datasets, they are predefined in a configuration file. The highest P_{max} and lowest P_{min} probabilities a map point can possess when locating at distance d_{min} and d_{max}, respectively, are also set in accordance with the surrounding conditions. A modified

version of the $sigmoid$ function is used to register probability P to a map point c with its depth d, as follows:

$$P(d) = 1 - \frac{1}{1 + \mathrm{e}^{-k(d-d_0)}} \tag{34}$$

where $k = 2\ln(P_{max}^{-1} - 1)/(d_{min} - d_{max})$; $d_0 = (d_{min} + d_{max})/2$; $d_{min} \leq d \leq d_{max}$.

This function retains the characteristic of the original function, having descending gradients when reaching its saturation. Moreover, it is ensured that the points (d_{min}, P_{max}) and (d_{max}, P_{min}) are passed through. Due to the symmetry property of $sigmoid$ function, this function has the properties: $P_{min} + P_{max} = 1$ and $P(d_0) = 0.5$.

Since the same map point can be detected time after time, whether in the same dataset or across different datasets, a probabilistic-based optimization is made to estimate its position when it is added to the dense point cloud map repeatedly. The probabilities of confidence for both times are calculated by equation (34) and used jointly to find a weighted average.

For a newly added map point c_1, the spatial distance from its closest point existing in the point cloud, c_2, is calculated by two-norm. If this is less than a certain value, that is converted from the preset resolution, these two map points will be considered as the same point, and be replaced with a new map point, c_3. We define the update as:

$$S(c_3) = \frac{P(d_1)}{P(d_1) + P(d_2)} S(c_1) + \frac{P(d_2)}{P(d_1) + P(d_2)} S(c_2) \tag{35}$$

where $S(c_i) = [x_i, y_i, z_i, d_i]$, representing the Cartesian coordinates of map point c_i in the world reference frame and its distance from the camera.

The probability, P_3, is then calculated by equation (35). As such, c_3 becomes the latest map point that another searching and replacing process is applied to. This is iterated as many times as necessary until no map point can be found within the minimum spacing. Figure 17 exemplify the update in 2D.

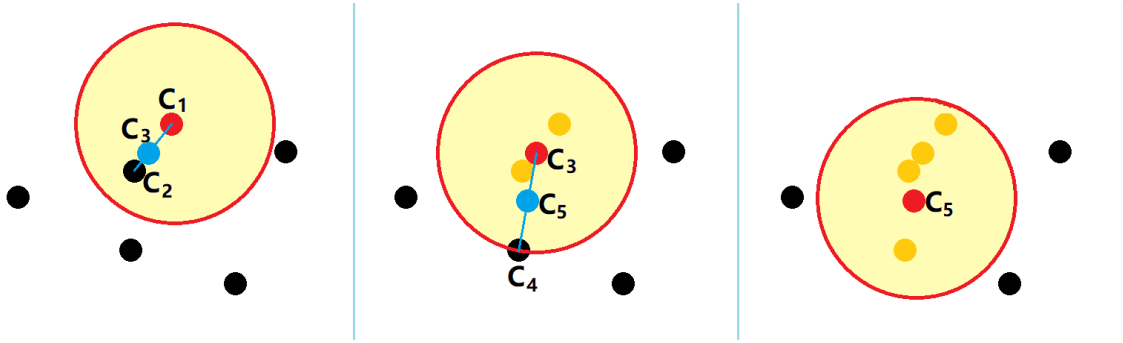


Figure 17: Example of Point Cloud Updating

Each black dot represents a map point that already exists in the dense point cloud. The red dot indicates the newest inserted map point and the red circle filled with light yellow describes the searching space around a map point. The blue dot shows a new position for the map point which is estimated by combining the two previous probabilities. Those map points that have been removed from the map are drawn in orange.

For the instance in this figure, two rounds of selection, searching and replacing procedure have occurred. At the end, the six initially map points in the point cloud is eliminated to four, with an accepted distance between any two of them.

In conclusion, using the update rule introduced in this section, we are able to achieve more accurate map point estimations; meanwhile, the redundant map points are banished from the point cloud to reduce the computation workload.

CHAPTER 4 RESULTS AND DISCUSSION

We verify the performance of our proposed system with the following two sets of evaluations:

• 3D dense reconstruction: with benchmark dataset TUM [33] for RGB-D camera; and indoor dataset EuRoC [34], outdoor dataset KITTI [35] for stereo camera configuration.

• Multi-sequence merging in all datasets.

All experimental results are produced in the operating environment of Intel Core i7-7500U at 2.7GHz, 1×8 GB RAM, and without GPU acceleration.

The length of each sequence in all three datasets are provided in Table III below.

Table III: Dataset Characteristics

a. TUM dataset (Length in m, duration in s, camera $30Hz$)

Sequence	*FR1_desk*	*FR1_room*	*FR2_desk*	*FR2_no_loop*	*FR2_with_loop*	*FR3_office*
Length	9.263	15.989	18.880	26.086	39.111	21.455
Duration	23.40	48.90	99.36	112.37	173.19	87.09

b. EuRoC dataset (Length in m, duration in s, camera $2 \times 20Hz$)

Sequence	*MH_01*	*MH_02*	*MH_03*	*MH_04*
Length	80.6	73.5	130.9	91.7
Duration	182	150	132	99
Sequence	*MH_05*	*V1_01*	*V1_02*	*V1_03*
Length	97.6	58.6	75.9	79.0
Duration	111	144	83.5	105

c. KITTI dataset (Total length 39.2km, 41k frames, in 22 sequences, camera $2 \times 10Hz$)

Sequence	*00*	*01*	*02*	*03*	*04*	*05*	*06*	*07*
Frames	4541	1100	4661	801	271	2761	1101	1101

To validate the accuracy of our proposed system, we compute the root mean square value of absolute trajectory error, which is a widely used criteria and defined as:

For an estimated trajectory $\widehat{\boldsymbol{X}} = \{\hat{x}_1 \dots \hat{x}_n\}$ and its corresponding ground truth $\boldsymbol{X}$,

$$ATE_{RMSE}(\widehat{\boldsymbol{X}}, \boldsymbol{X}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left\|T^{\widehat{X}_i} - T^{X_i}\right\|^2} \tag{36}$$

where inside the sigma is the Euclidean distance between the two poses from estimation and ground truth at time stamp i.

4.1 RGB-D Dense Reconstruction

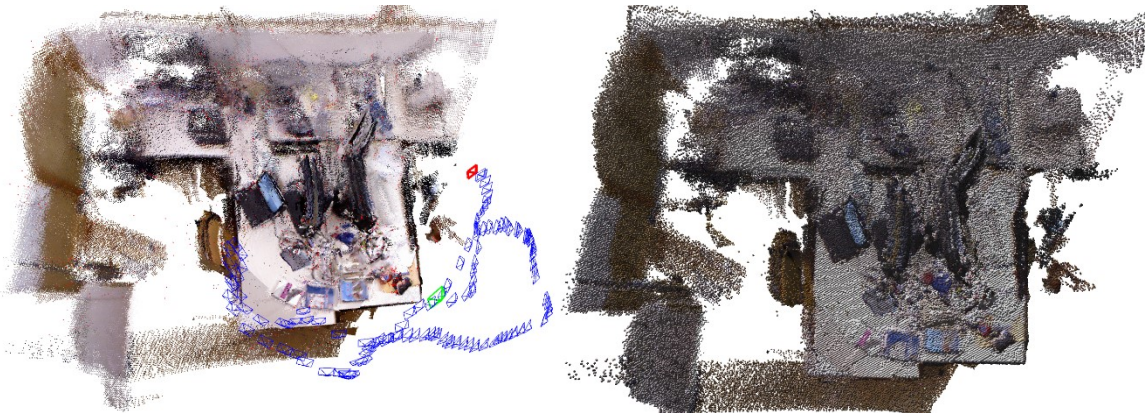
Figure 18 exhibits the dense point cloud maps with their corresponding Octomap generated from sequences in the TUM dataset using RGB-D camera configuration and single sequence operation.

These results demonstrate the validity of our system with RGB-D images. The red and green markers in the point cloud represent the starting and ending locations while those blue markers are the keyframes determined by the original ORB-SLAM3 system.

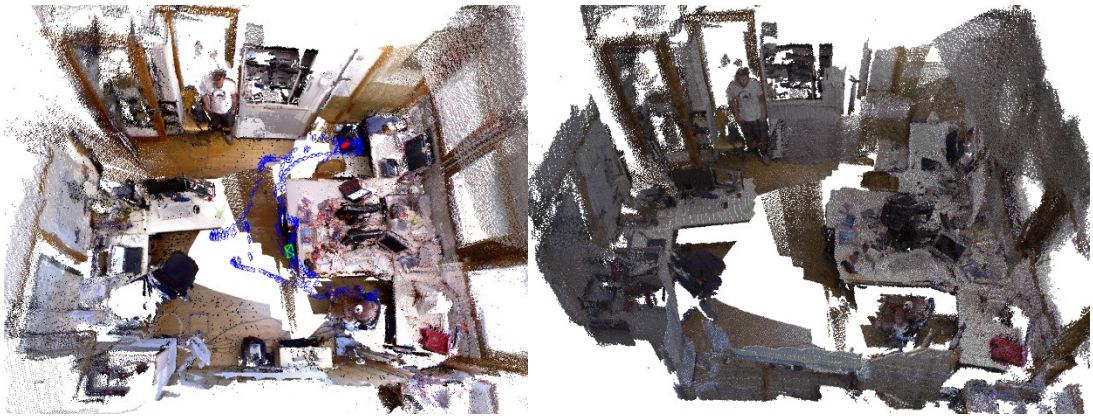
Errors calculated for all sequences with the proposed system is summarized in Table IV, showing a comparison with RGB-D SLAMv2. Results for proposed system are calculated from the average of three runs for fairness. Results for RGB-D SLAMv2 is obtained from [22].

Table IV: TUM Single Sequence RGB-D Reconstruction Performance Comparison

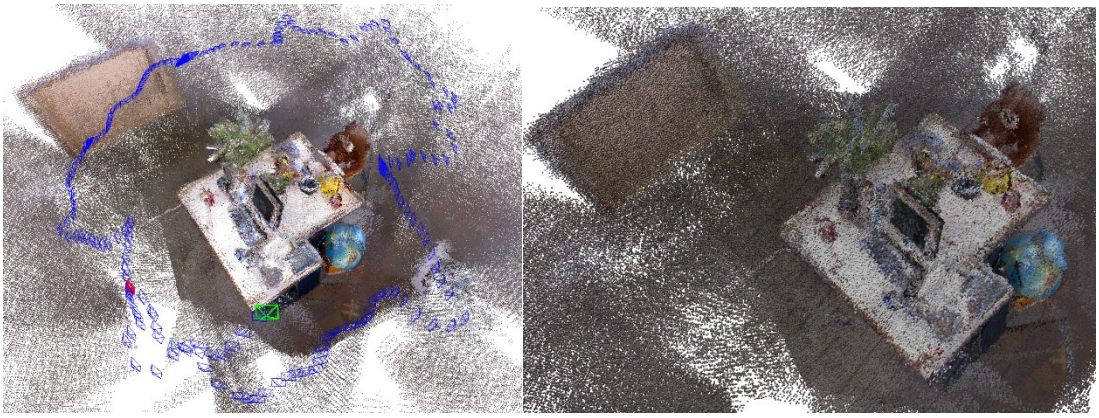
Sequence	*FR1_desk*	*FR1_room*	*FR2_desk*	*FR2_no_loop*
Proposed ATE_{RMSE} (m)	**0.017**	**0.052**	**0.018**	**0.237**
Proposed FPS (Hz)	**23.7**	**24.2**	**19.6**	**17.7**
RGB-Dv2 ATE_{RMSE} (m)	0.026	0.087	0.057	0.860
RGB-Dv2 FPS (Hz)	15.2	14.0	7.34	6.80



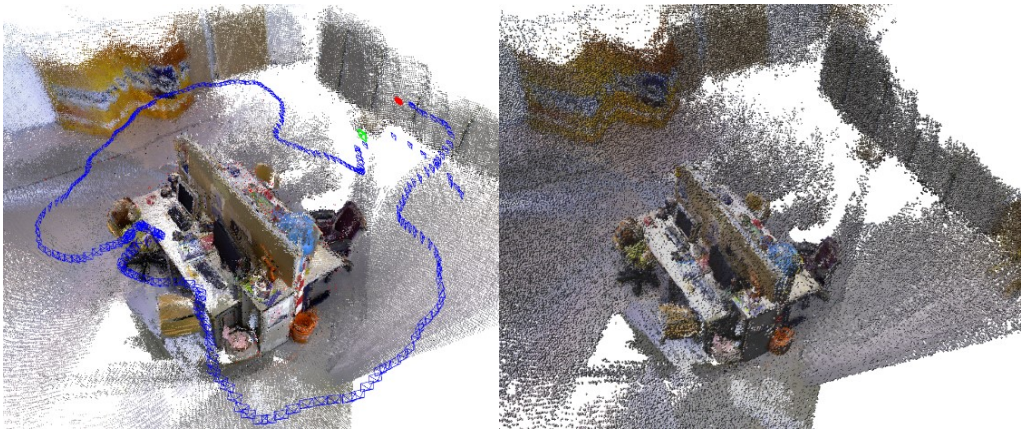
(a) FR1 desk



(b) FR1 room



(c) FR2 desk



(d) FR3_office

(e) FR2_no_loop

Figure 18: Dense Point Cloud (left) and Octomap (right) Reconstructed with RGB-D Camera from Single Sequence in TUM Dataset

It can be summarized that the proposed system exhibits better performance compared to RGB-D SLAMv2 algorithm, the root mean square absolute trajectory error is reduced up to one-fourth. Considering the camera input is at 30Hz in this dataset, we are close to achieve real-time dense reconstruction in small indoor environment. However, the time consumption is still challenging in a relatively large room. This could be improved significantly by reducing the map density, which is coded as a user input that can be easily

modified at preparative stage. Comparing the frame rates in *FR1_room* and *FR1_desk* datasets, it is observed that the smaller-scale scene unexpectedly demonstrates less frame rates, indicating that it could be the hardware that constraints the system performance.

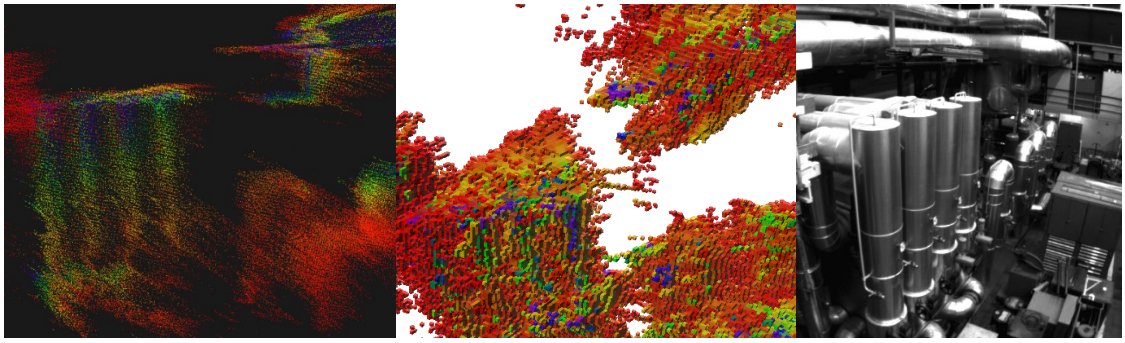


Figure 19: Dense Map (left) and Octomap (mid) Obtained from Pipes (right) in EuRoC

4.2 Stereo Dense Reconstruction

For reconstruction from a stereo camera, we validated our system performance in both indoor and outdoor, with EuRoC and KITTI dataset, respectively.

In terms of the indoor EuRoC dataset, we have run dense reconstruction in two scenes. The first one is captured inside the industrial machine hall in ETH Zurich, therefore containing pipes and tanks. Note that this is a challenging dataset since images are captured from a stereo camera implemented on a drone. The aerial camera movements and changing lighting conditions can exert huge influence on our SLAM system.

To reduce the computational cost while maintaining a relatively reliable visualization, the color information from input images is discarded, and spatial points are assigned with rainbow colors based on its distance from the camera, red is the farthest and vice versa.

An example of reconstruction from the pipes captured in *MH_01* is illustrated as Figure 19.

Figure 20 shows the reconstructed dense map as well as the Octomap.

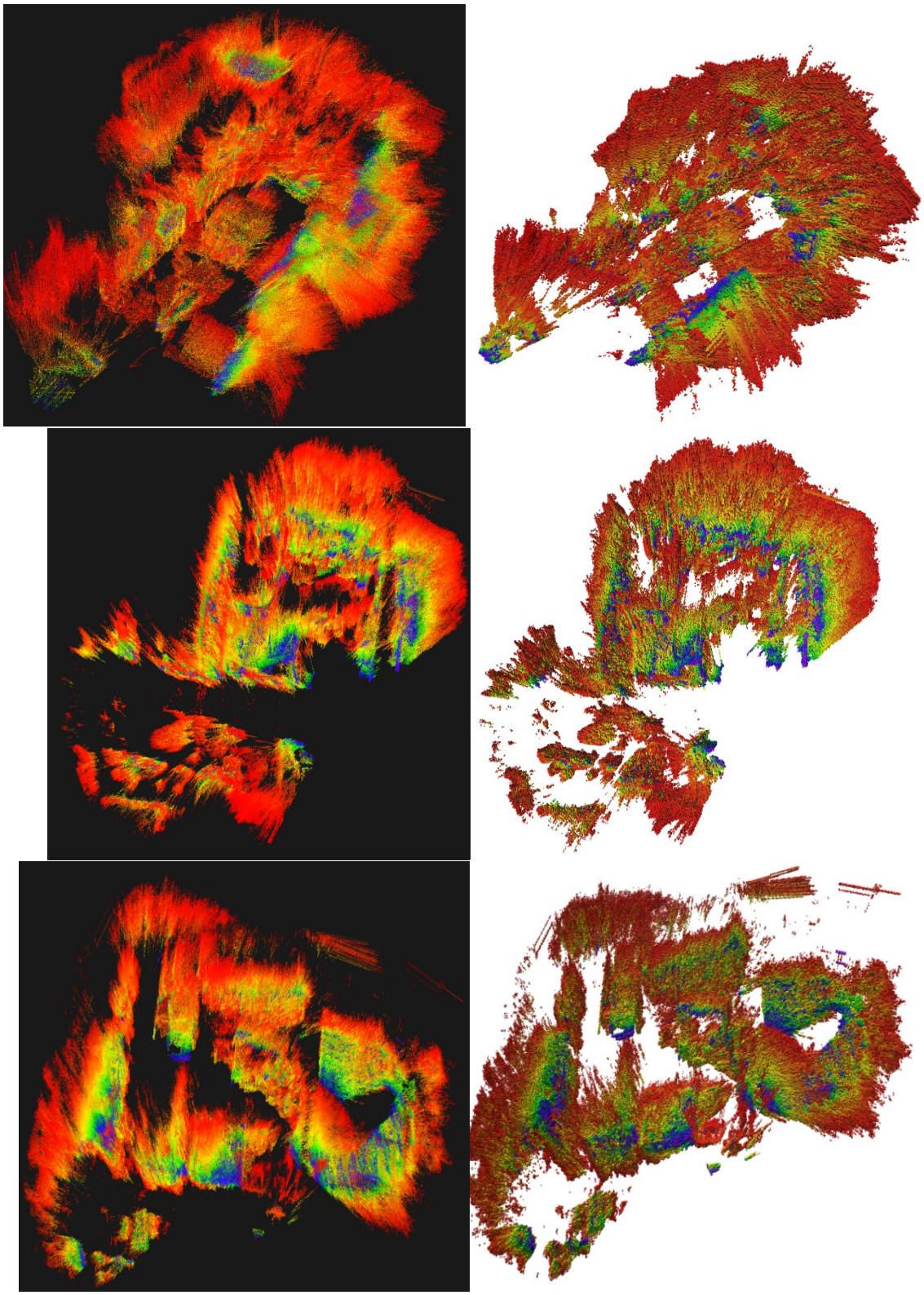


Figure 20: Dense Map (left) and Octomap (right) Reconstructed with Stereo Camera from Single Sequence in EuRoC *MH_01* (top), *MH_03* (mid), and *MH_05* (bottom)

The second scene contains three sequences recorded in the Vicon Room with different obstacle configurations. There are some moving curtains visible while recording these sequences, so having accurate tracking becomes more challenging. Figure 22 presents the dense map and Octomap generated from these sequences.

We are able to rotate these 3D maps in map viewer software, which immensely helps in understanding the room's environment; nonetheless, it is not possible to attach the 3D map in this thesis as PDF. To further enhance the readability of these maps, a sparse map is provided below as Figure 21. Other than what have been explained for Figure 18 above, the red and black dots represent active and inactive keypoints captured from camera, respectively, and green lines shows the co-visibility relationship between keyframes.

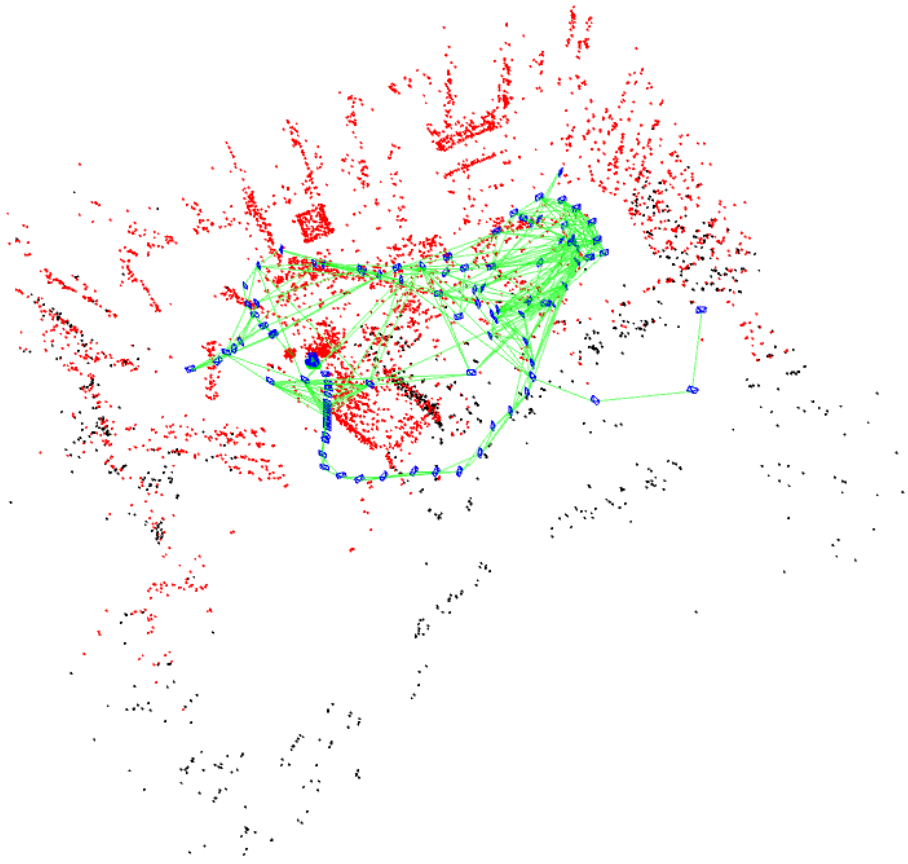


Figure 21: Sparse Map with Co-Visibility Obtained from EuRoC V1_01

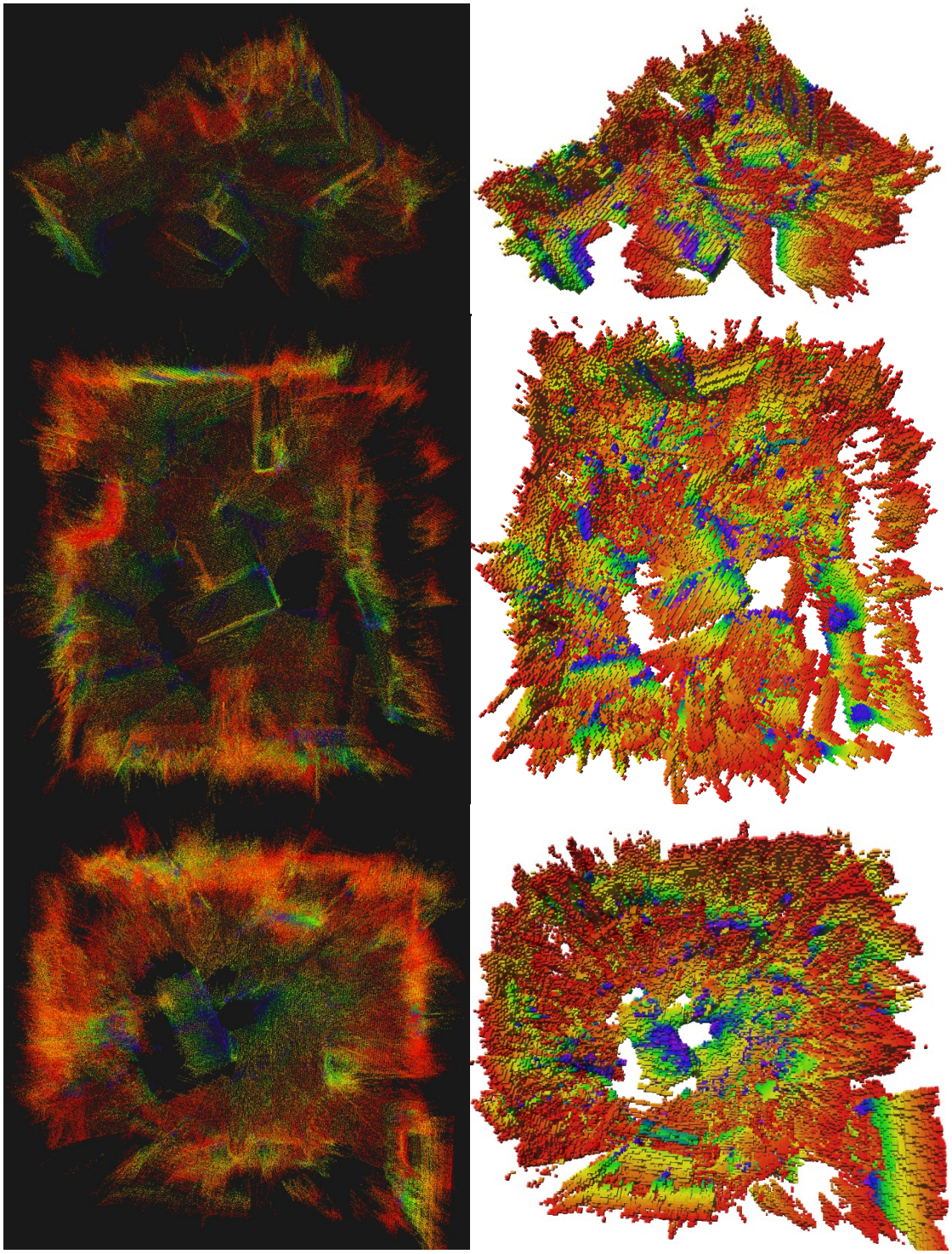


Figure 22: Dense Map (left) and Octomap (right) Reconstructed with Stereo Camera from Single Sequence in EuRoC *V1_01* (top), *V1_02* (mid), and *V1_03* (bottom)

Errors calculated for all sequences using the proposed system are summarized in Table V, compared with the stereo configuration of ORB-SLAM3. Results for proposed system are calculated from the average of three runs for fairness. Results for ORB-SLAM2 and ORB-SLAM3 are obtained from [16] and [19].

Table V: EuRoC Single Sequence Stereo Reconstruction ATE_{RMSE} (m) Comparison

Sequence	*MH_01*	*MH_02*	*MH_03*	*MH_04*	*MH_05*	*V1_01*	*V1_02*	*V1_03*
ORB-SLAM3 stereo	**0.029**	0.019	**0.024**	**0.085**	**0.052**	**0.035**	0.025	0.061
ORB-SLAM2 stereo	0.035	**0.018**	0.028	0.119	0.060	**0.035**	**0.020**	**0.048**
Proposed	0.039	0.028	0.032	0.112	0.061	0.042	0.036	0.107

It can be concluded that with the proposed system, an acceptable trajectory tracking can be achieved. However, unlike the enhancement achieved for the RGB-D configuration compared to RGB-D SLAMv2, we are not able to obtain improved results compared to ORB-SLAM family. A potential reason for this is the parameter tuning for the reconstruction. We intentionally reduced the number of keypoints in each keyframe from the original value used in ORB-SLAM3, decreasing it from 2000 to 1000. This sacrifice brings significant enhancement on real-time performance, boosting the frame rate from less than 10Hz to around 15Hz. More importantly, this modification solves an occasional program die-out issue when running the MH_05 sequence, which sometimes consumes all memory space of the laptop.

With respect of the validation for outdoor environment, KITTI provides 22 sequences recorded in Karlsruhe, Germany. Figure 23 [36] illustrates a map where all sequences are recorded. It is introduced that those high precision with GPS corrections are in red, whereas

the rest in blue shows the GPS absence areas. These sequences are from both rural areas and on highways, so a typical length of one to two kilometers is expected. It is introduced that the first 11 sequences out of these 22 are used for testing purpose, thereby making the ground truth public. The rests are made for validation only, so it is not possible to obtain the ATEs unless a submission to the dataset publisher's website is made. As the result, only sequence *00* to *10* is used in this thesis for analysis.

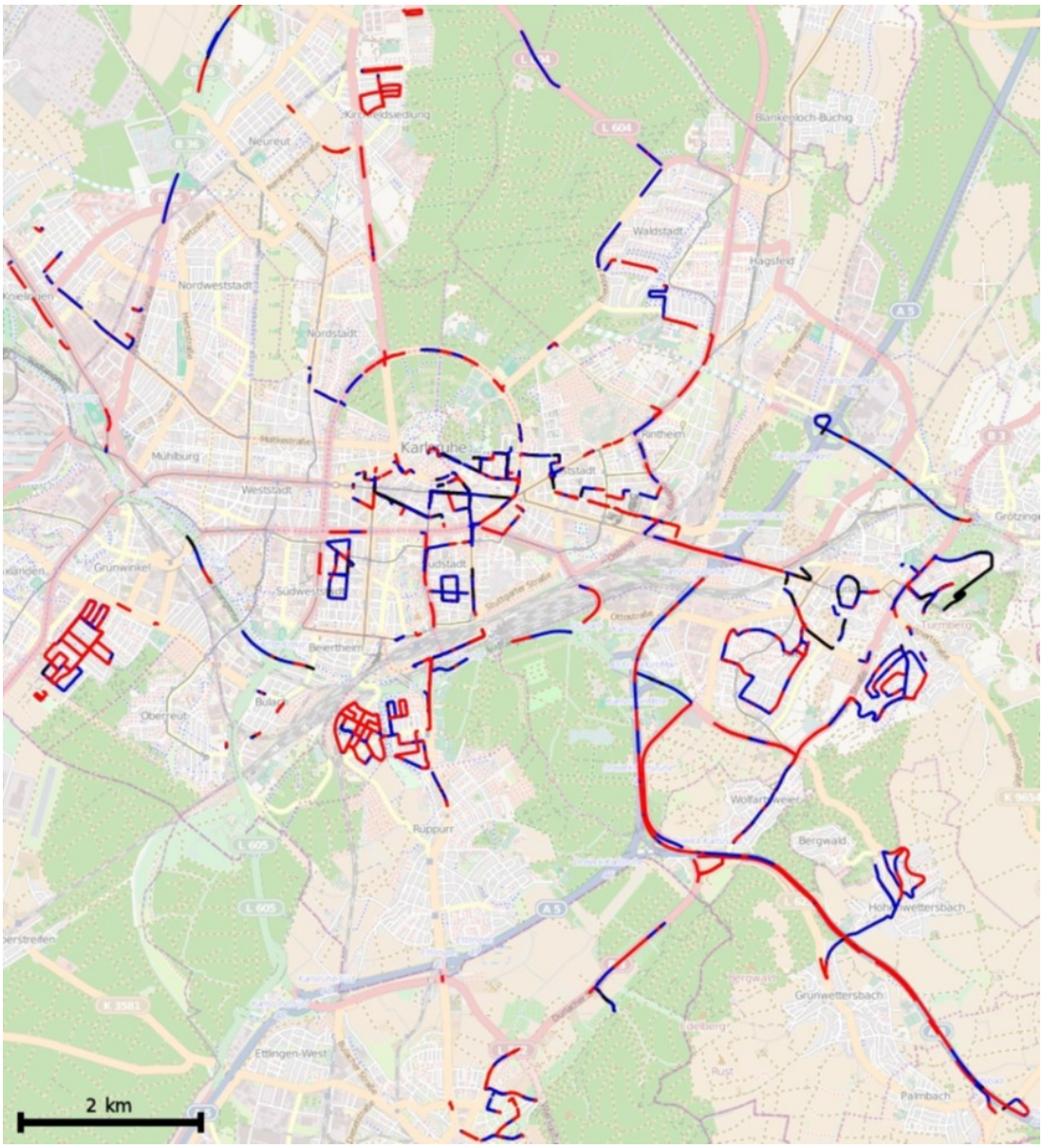


Figure 23: Recording Zone for KITTI Dataset [36]

An example of reconstruction from the road captured in *00* is illustrated as Figure 24.

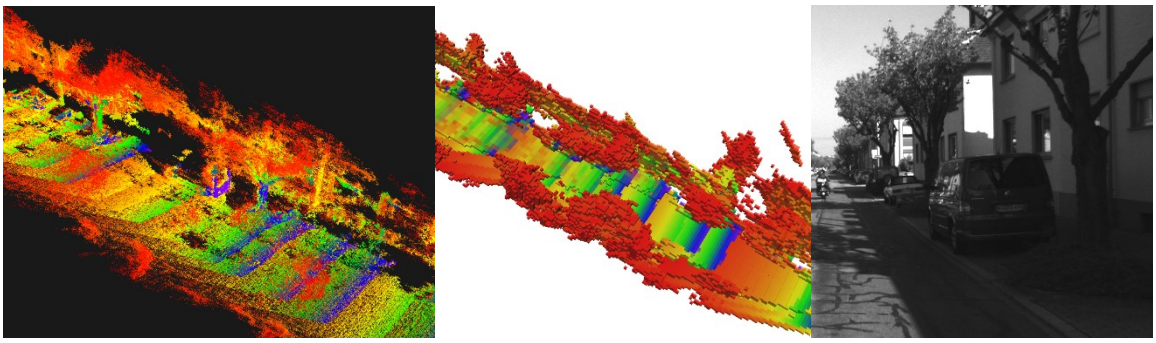


Figure 24: Dense Map (left) and Octomap (mid) Obtained from Road (right) in KITTI

Trees and a vehicle on the road from the input image can be correspondingly found in the reconstructed maps. However, only the back part of this vehicle is captured, which is denoted in purple located at the middle part of the dense map.

Figure 25 shows the dense map and the Octomap reconstructed from this outdoor dataset.

To further validate the accuracy of the proposed system, comparisons of the absolute trajectory errors from sequences *00* to *10* are performed among the proposed system with the latest two generations in ORB-SLAM family. Because the results of ORB-SLAM3 using stereo camera configuration are not reported in their publish, three runs are operated with the same setup as the proposed system, averaged and collected in Table VI below. Results for ORB-SLAM2 is retrieved from [16] directly.

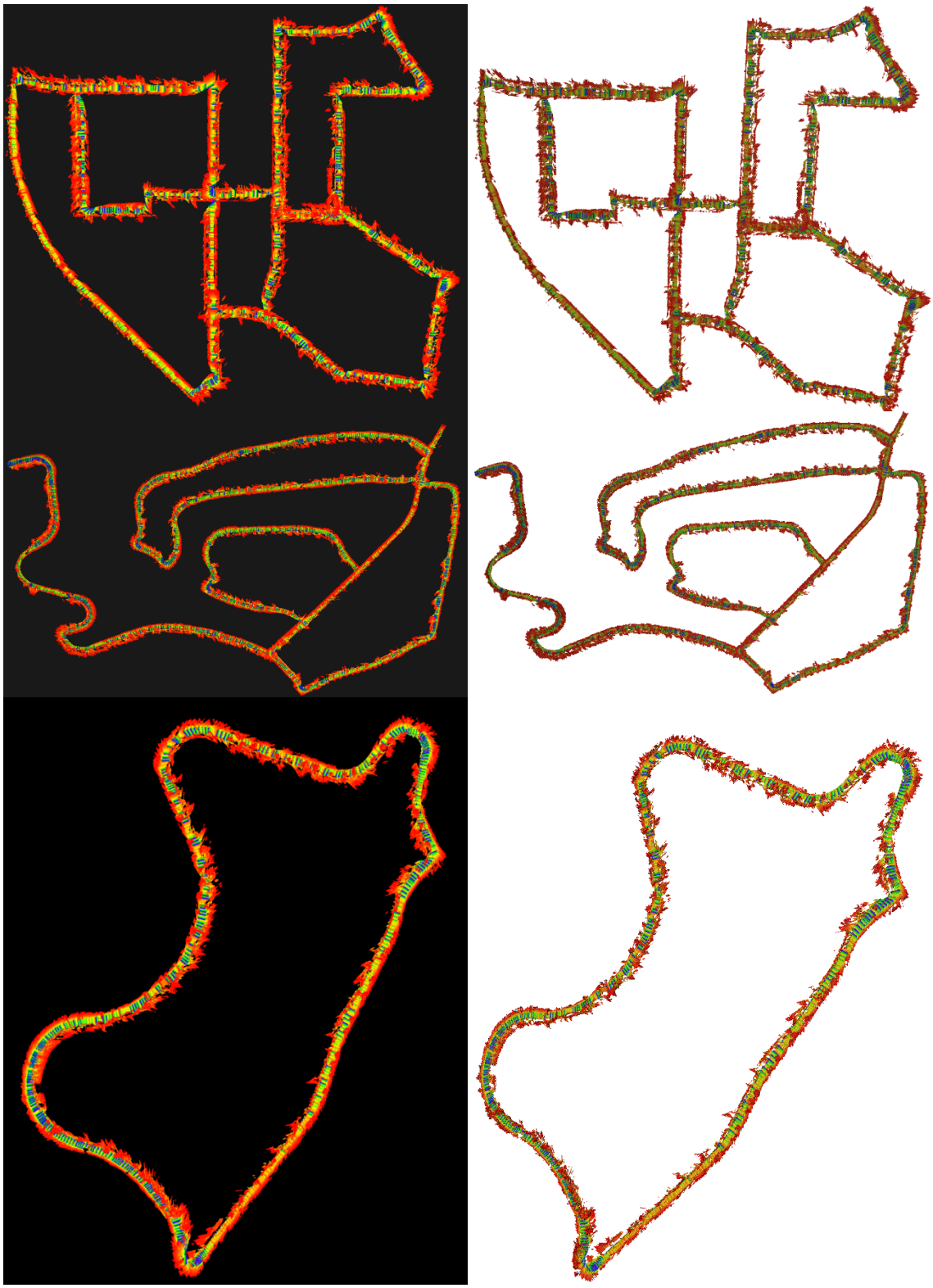


Figure 25: Dense Map (left) and Octomap (right) Reconstructed with Stereo Camera from Single Sequence in KITTI *00* (top), *02* (mid), and *09* (bottom)

Table VI: KITTI Single Sequence Stereo Reconstruction ATE_{RMSE} (m) Comparison

Sequence	*00*	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*
ORB-SLAM3 stereo	1.252	**5.364**	**1.805**	0.749	**0.192**	0.638	1.025	**0.379**	2.691	3.448	**0.927**
ORB-SLAM2 stereo	1.3	10.4	5.7	0.6	0.2	0.8	**0.8**	0.5	3.6	**3.2**	1.0
Proposed	**1.248**	5.596	1.924	**0.637**	0.233	**0.561**	1.352	0.459	**2.654**	3.861	1.245

It can be discovered that the proposed system achieves better accuracy compared to ORB-SLAM2, especially in the first three sequences. However, for the rest sequences there is no significant enhancement. This is most likely due to the parameter selections, which needs to be tuned carefully for best performance. Comparing to ORB-SLAM3, the proposed system holds similar accuracy, it can be assumed that taking average from more iterations will further reduce the difference between these two systems.

Overall, the proposed system is able to achieve dense reconstruction from either indoor or outdoor environments using a stereo camera. Its accuracy is proven after the comparison with other algorithms using the same sensor configuration.

4.3 Multi-Sequence Merging

Multi-sequence operations are performed on both RGB-D and stereo datasets. Specifically, TUM and EuRoC are chosen to validate the performance of multi-sequence merging. KITTI is not considered suitable for this purpose as it only comprises large outdoor sequences without significant overlaps. We are aiming to demonstrate the accuracy of proposed system can be enhanced after merging multiple sequences with overlapping features.

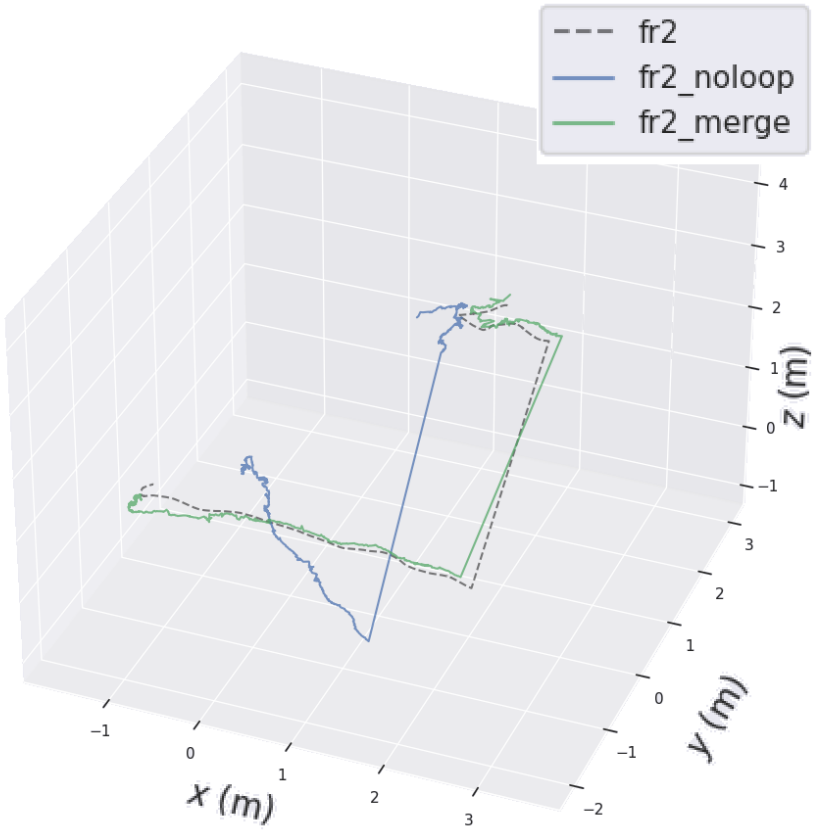


Figure 26: Comparison of Trajectories between Single Sequence and Multi-Sequence Operation on TUM *FR2_large_no_loop* with Ground Truth

We first tested the proposed system with TUM dataset. Figure 26 represents a comparison of trajectories and errors between running *FR2_large_no_loop* as a single sequence and a multi-sequence operation with a priori information collected from another sequence *FR2_large_with_loop*.

Three trajectories are shown in this figure: the ground truth is represented with a black dashed line whereas the single and multisequence operations have trajectories in blue and green. The middle portion of all trajectories is straight since the author of this dataset intentionally turned off GPS for added difficulty, causing the corresponding part to be

missing. In spite of this, it is clearly illustrated that the trajectory after sequence merging is closer to the ground truth, proving that the accuracy of system is improved.

To further explain the enhancement, the instantaneous and cumulative ATE_{RMSE} are plotted in Figure 27. As introduced earlier, that portion without ground truth provided is cropped from the plot, creating a discontinuous on the x-axis for elapsed time of system. As the legend indicates, black and red solid lines represent the instantaneous absolute trajectory error at each individual time stamp, using y-axis on the left; blue and brown lines with transparent fill represent the cumulative absolute trajectory error for single sequence and multiple sequence merging operations, respectively.

It is calculated that the average ATE per frame drops when using multi-sequence configuration, from 0.706m to 0.181m. Therefore, it can be summarized that with the proposed multi-sequence merging method which reuses the knowledge from previous sequences, the system accuracy can be enhanced.

A similar evaluation has been made on the stereo dataset, EuRoC.

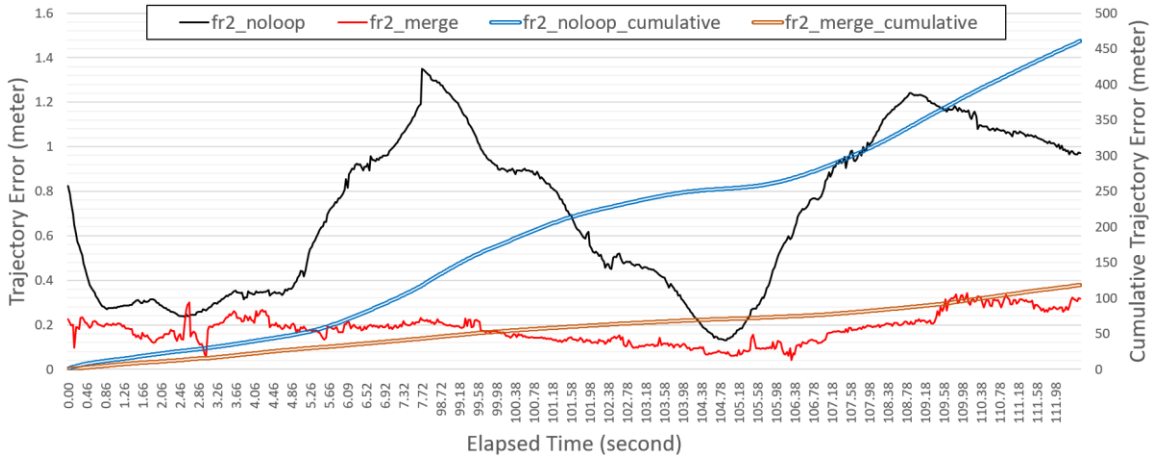


Figure 27: Comparison of Instantaneous and Cumulative ATE_{RMSE} between Single and Multi-Sequence Operation on TUM *FR2_large_no_loop* with Ground Truth

In order to visualize the merging of multiple sequences, different colors are assigned to each of the sequences from MH_01 to MH_05, instead of using a rainbow color like what has been done in the single sequence operation. Figure 28 illustrates a multi sequence merging of *MH_01*, *MH_02*, *MH_03*, *MH_04*, and *MH_05*, which are colored in blue, pink, orange, purple, and green, respectively. The subfigure on the right is a sparse map generated during the final run of five, multiple trajectories are included as the operation of *MH_05* is reusing the knowledge from previous runs.

Figure 29 presents a closer look at these trajectories. Camera poses at keyframes in the former maps are denoted in purple, while in the current map, they are colored in blue. It is evident that the co-visibilities are estimated between these two trajectories, implying cross-dataset connections.

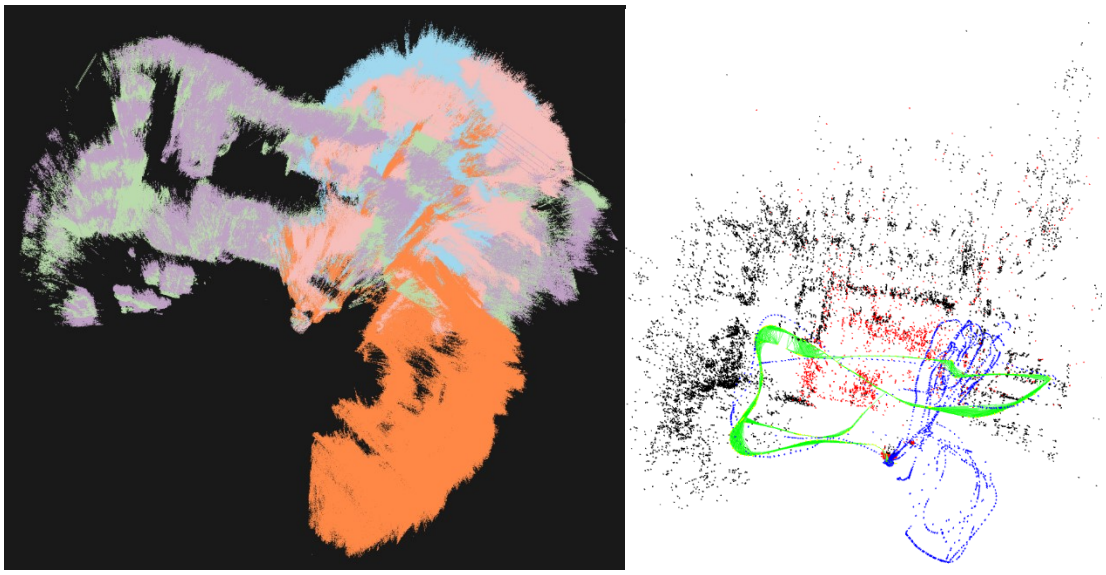


Figure 28: Dense (left) and Sparse (right) Map Generated from Multi-Sequence Merging of EuRoC Sequences, the Latter Sequence Reuses Map from Previous Ones

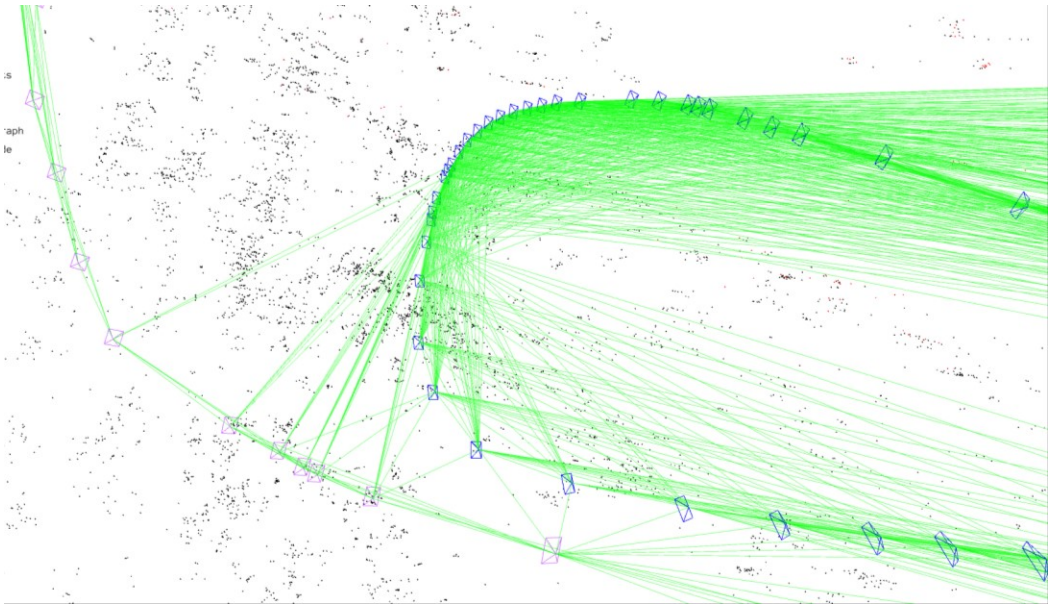


Figure 29: Cross Dataset Co-Visibilities Estimated in EuRoC Multi-Sequence Merging

The root mean square absolute trajectory error of multi-sequence merging is concluded below in Table VII.

It is noteworthy that after merging with the map knowledge from *MH_01* sequence, the ATE_{RMSE} of *MH_02* is reduced from 0.028m, summarized in Table V, to 0.026m. Similar enhancement can be observed for *V1_02* sequence after merging with *V1_01*.

However, for each set of sequences in EuRoC dataset, the difficulty is described as increasing by the author. Therefore, the improvements are not tremendous, most of the time for less than one centimeter. We are expecting to see more significant improvements on sequences with similar difficulties.

Besides, we noticed an unexpected increase of error after merging the maps obtains from first three sequences in Machine Hall with the fourth sequence. After checking the dataset, it is discovered that although *MH_04* shares the same room with other three sequences, it

does not have much common view with the others. The limited enhancement is explained as a result of lacking co-visibility.

Table VII: EuRoC Multi-Sequence Merging ATE_{RMSE} (m) Summary

Sequence	*MH01*	*MH01-02*	*MH01-03*	*MH01-04*	*MH01-05*
ATE_{RMSE}	0.039	0.026	0.027	0.091	0.044
Sequence	*V1_01*	*V1_01 - V1_02*	*V1_01 - V1_03*		
ATE_{RMSE}	0.042	0.028	0.030		

Overall, we have checked the validity of the multi-sequence merging method of the proposed system. Enhancements on the system accuracy after reusing the map knowledge from previous sequences can be observed for both RGB-D and stereo camera configurations.

4.4 Octomap

Four different mapping methods can be used in the proposed system: point cloud map, sparse mapping, dense mapping, and Octomap. Figure 30 compares a laptop mapped by these four methods, from left to right.

The point cloud map contains a collection of 3D points in space, and it is the raw data used to create sparse map and dense map. Although it is visually similar to the dense map, point cloud is unordered, or sparsely distributed.

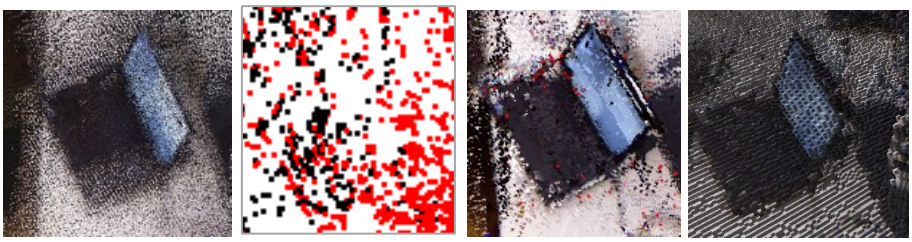


Figure 30: Laptop Mapped by Point Cloud, Sparse, Dense Mapping and Octomap

This becomes even more obvious when using sparse map, all unnecessary information, such as the shape, color, and texture of the laptop, have been discarded. For this case, anything other than the ORB feature points is removed for computational simplicity. Overlapping with those features from other objects, such as the desk underneath, it is difficult for human to tell what objects exist in the map.

Dense mapping solves this problem with a significant higher computational complexity. The texture of laptop is fully captured and there is an occupancy grid related to it. This occupancy grid discretizes the 3D space into grid cells and assigns each cell with a binary value. This value is dynamically updated to represent if the corresponding space is occupied by an object or not, which is crucial to multiple robotic applications.

Octomap, as introduced earlier, is a solution to the huge memory consumption by dense mapping. A comparison of the Octomap generated from TUM *FR1_room* using different leaf size is included below in Figure 31. Table VIII shows the file size of the dense map (*.pcd*) and the Octomap (*.ot*) regarding to different leaf size configurations but for the same sequence. It can be observed that a smaller value should be set for the leaf size if a finer map is required. However, the smaller leaf size is set, the larger file size would be expected. It is noteworthy that when a small leaf size is set for a large map, an integer overflow error may occur before killing the program. Based on experience, using a leaf size around 0.01m in an indoor sequence is considered appropriate; as for outdoor datasets like KITTI, leaf size should not be set less than 0.05m, a typical value for me to use is 0.2m.

Table VIII: File Size (in Megabytes) for Different Leaf Size Configurations

Leaf Size (m)	0.01	0.02	0.05	0.1	0.2
Dense Map	106.4	31.6	6.6	1.2	0.317
Octomap	25.2	6.4	1.2	0.230	0.062

Figure 31: Octomap with Leaf Size 0.01m (top left), 0.02m (top right), 0.04m (bottom left), and 0.08m (bottom right) from TUM *FR1_room*

Table IX shows the comparisons we made on the size of each Octomap (*.ot* file) and its corresponding dense map (*.pcd* file). The compactness of Octomap can be proved by the diminishing size of each map after being converted.

4.5 Runtime Analysis

Similar to the code included in the ORB-SLAM3, which analyzes the time cost for each step during the SLAM process, a piece of code is written to determine the time required to accomplish dense reconstruction.

The time required for each step of the main operations in the proposed system as well as the ORB-SLAM3 has been concluded in Table X. This table contains analysis on both RGB-D and stereo configuration, indoor and outdoor. The data for the ORB-SLAM3 stereo configuration on EuRoC *V2_02* sequence is directly sourced from [19], while the data for the stereo configuration on KITTI *07* sequence and the RGB-D configuration on TUM *FR3_office* sequence are derived from my experiment and averaged from three trials.

Table IX: File Size (in Megabytes) of Dense Mapping Compared to Octomap

TUM	*FR1_desk*	*FR1_room*	*FR2_desk*	*FR2_no_loop*
Dense	36.7	90.4	67.7	52.1
Octomap	7.5	22.6	17.2	7.9
TUM	*FR2_with_loop*	*FR3_office*	**Average Compress Ratio**	
Dense	31.3	34	**5.138**	
Octomap	4.2	8.6		
EuRoC	*MH_01*	*MH_03*	*MH_05*	*V1_01*
Dense	38.2	43.1	71.1	5.8
Octomap	5.5	6.6	11.5	0.94
EuRoC	*V1_02*	*V1_03*	**Average Compress Ratio**	
Dense	11.1	16.6	**6.546**	
Octomap	1.7	2.4		
KITTI	*00*	*01*	*02*	**ACR**
Dense	83.7	4.8	15.2	**4.771**
Octomap	16.1	1.1	3.2	

Table X: Running Time (in milliseconds) of Main Parts in the Proposed System Compared to ORB-SLAM3, on TUM *FR3_office*, EuRoC *V2_02*, and KITTI *07*

*Statistics for ORB-SLAM3 stereo on EuRoC *V2_02* is based solely on author's report.

Settings	Sensor	RGB-D		Stereo		Stereo	
	Dataset	TUM		EuRoC		KITTI	
	Sequence	*FR3_office*		*V2_02*		*07*	
	Input FPS	30Hz		20Hz		10Hz	
	Resolution	640 × 480		752 × 480		1226 × 370	
	ORB Features	1000		1000		2000	
System		**Ours**	**ORB3**	**Ours**	**ORB3***	**Ours**	**ORB3**
Tracking	Stereo Rectification	-	-	1.56	1.32	-	-

	ORB Extraction	9.10	8.84	17.27	15.68	18.28	18.61
	Stereo Matching	-	-	3.05	3.35	4.02	3.91
	Pose Prediction	2.57	2.58	2.73	2.69	2.57	2.56
	Local Map Tracking	6.62	6.56	6.60	6.31	4.79	4.83
	New Keyframe Decision	0.24	0.24	0.21	0.12	0.28	0.28
	Total	**19.41**	**19.15**	**32.16**	**31.48**	**31.92**	**32.17**
Mapping	Keyframe Insertion	8.21	8.20	8.24	8.03	8.83	8.70
	Map point Culling	0.28	0.29	0.35	0.32	0.30	0.31
	Map Point Creation	22.83	23.07	19.52	18.23	19.81	19.22
	Local BA	251.62	255.96	146.79	134.60	67.21	64.91
	Keyframe Culling	5.14	4.87	5.31	5.49	0.94	1.00
	Total	**290.50**	**294.71**	**174.20**	**158.84**	**98.32**	**95.36**
Loop	Database Query	0.75	0.75	0.98	1.06	1.03	1.01
	Compute Sim3/SE3	5.89	5.90	6.19	5.26	9.30	9.88
	Loop Fusion	263.67	265.13	31.86	29.07	84.13	88.42
	Essential Graph Optimization	135.06	134.42	71.43	84.36	132.33	134.13
	Total	**408.99**	**408.34**	**112.46**	**124.94**	**226.79**	**233.44**
Loop Full BA	Full BA	2176.49	2358.61	1809.31	1118.54	2560.22	2439.94
	Map Update	14.95	15.14	16.95	13.65	28.31	30.12
	Total	**2191.44**	**2373.75**	**1826.26**	**1132.19**	**2588.53**	**2470.06**
Dense Reconstruction	Keyframe Insertion	8.31	-	8.19	-	9.19	-
	Depth Acquisition	0.02	-	1.57	-	1.49	-
	Voxel Filtering	91.90	-	82.96	-	99.68	-
	Map Update	85.35	-	89.60	-	107.69	-
	Octomap Conversion	153.27	-	149.32		182.21	-
	Total	**385.70**	**-**	**378.49**	**-**	**447.11**	**-**

It is reported in [19] that the original system is able to run in real time at 30-40 frames and at 3 to 6 keyframes per second. Besides, recognizing a closing loop will trigger the loop closing thread that contains a very time expensive full bundle adjustment, which can cause the system a pause for around one second. This value has been proven to vary significantly depending on the device used, typically increasing to around two seconds on my laptop. With dense reconstruction enabled, the frame rate would be around 25Hz in small environments, decreasing as the sequence becomes larger. it can drop to as low as 10Hz in extremely large outdoor sequence provided by KITTI.

From this analysis, it can be concluded that implementing this dense reconstruction plugin to the original ORB-SLAM3 system does increase the system's running time, but not by an unacceptable amount. People can either reduce the maximum amount of ORB features allowed per keyframe or reduce the input image resolution, sacrificing the system accuracy to achieve improved running time for particular use.

CHAPTER 5 CONCLUSION AND FUTURE WORK

5.1 Conclusion

We proposed a visual SLAM system based on ORB-SLAM3in this paper. Aside from original functions, we offer a 3D dense reconstruction function through either RGB-D or stereo camera. It is also noteworthy that a conversion to generate Octomap is implemented to our system, not only reducing the map size, but also making the system competent in comprehensive applications. Furthermore, utilizing the map saving and loading functions of ORB-SLAM3, we apply probabilistic-based optimizations to merge information from multiple sequences, providing a remarkable enhancement to the system accuracy.

Besides, we are mindful of a possible lessening of computing speed that is caused mainly by the reconstruction, which is comparatively more computing expensive. Consequently, we code with foresight so that the modifications can be seamlessly integrated into the original ORB-SLAM3 software. The applicability is secured since they can be toggled effortlessly without recompilation. A runtime analysis has also been performed to show the loss of computing time is within an acceptable range.

The validity of the proposed system has been tested with both RGB-D and stereo datasets, indoor and outdoor. Comparisons have been made with state-of-the-art SLAM algorithms to demonstrate the effectiveness of the proposed system.

5.2 Future Work

As future work, the proposed system should be implemented with hardware for testing and future improvements.

Moreover, the quality of reconstruction can be further improved with semantic algorithms. Presently, the proposed system lacks the ability to handle dynamic environments. It has been observed, particularly in certain sequences in the outdoor KITTI dataset, that other vehicles presenting on the road could degrade the system's performance. This also applies to the presence of any humans in the office room within the TUM dataset. Besides, the integration of semantic algorithms could facilitate the safe removal of unwanted elements in the dense point cloud maps, such as room ceilings and tree branches along the road.

Furthermore, due to the nature limitations camera holds, fusing it with other types of sensors has proven to bring not only a boost of system accuracy, but also sometimes an extension on the robot's application.

Lastly, I have noticed many researchers have combine their SLAM system with machine learning methods. Some of them are able to achieve incredibly high accuracy in particular environments. This could also become something I would like to challenge in the future.

BIBLIOGRAPHY

[1] H. Taheri, C. X. Zhao, “SLAM; definition and evolution.” *Engineering Applications of Artificial Intelligence*, vol. 97, pp. 104032, 2021.

[2] Y. Abdelrasoul, A. B. S. H. Saman and P. Sebastian, “A quantitative study of tuning ROS gmapping parameters and their effect on performing indoor 2D SLAM,” *2016 2nd IEEE International Symposium on Robotics and Manufacturing Automation (ROMA)*, pp. 1-6, 2016.

[3] S. Nagla, “2D Hector SLAM of Indoor Mobile Robot using 2D Lidar,” *2020 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS)*, pp. 1-4, 2020.

[4] J. Zhang, S. Singh, “LOAM: Lidar Odometry and Mapping in real-time,” *Robotics: Science and Systems Conference (RSS)*, pp. 109-111, 2014.

[5] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti and D. Rus, “LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping,” *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5135-5142, 2020.

[6] A. J. Davison, I. D. Reid, N. D. Molton and O. Stasse, “MonoSLAM: Real-time single camera SLAM,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052-1067, 2007.

[7] T. Bailey, J. Nieto, J. Guivant, M. Stevens and E. Nebot, “Consistency of the EKF-SLAM algorithm,” *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3562-3568, 2006.

[8] R. Martinez-Cantin and J. A. Castellanos, “Unscented SLAM for large-scale outdoor environments,” *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3427-3432, 2005.

[9] S. J. Julier and J. K. Uhlmann, “Unscented filtering and nonlinear estimation,” *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401-422, 2004.

[10] C. Kim, R. Sakthivel and W. K. Chung, “Unscented FastSLAM: a robust and efficient solution to the SLAM problem,” *IEEE Transactions on Robotics*, vol. 24, no. 4, pp. 808-820, Aug. 2008.

[11] G. Grisetti, C. Stachniss and W. Burgard, “Improved techniques for grid mapping with Rao--Blackwellized particle filters,” *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34-46, Feb. 2007.

[12] G. Klein and D. Murray, “Parallel tracking and mapping for small AR workspaces,” *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 225-234, 2007.

[13] R. A. Newcombe, S. J. Lovegrove and A. J. Davison, “DTAM: dense tracking and mapping in real-time,” *2011 International Conference on Computer Vision*, pp. 2320-2327, 2011.

[14] J. Engel, T. Schöps, D. Cremers, “LSD-SLAM: large-scale direct monocular SLAM,” *Computer Vision – ECCV 2014*, vol 8690, pp. 834-849, 2014.

[15] R. Mur-Artal, J. M. M. Montiel and J. D. Tardós, “ORB-SLAM: a versatile and accurate monocular SLAM system,” *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147-1163, Oct. 2015.

[16] R. Mur-Artal and J. D. Tardós, “ORB-SLAM2: an open-source SLAM system for monocular, stereo, and RGB-D cameras,” *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255-1262, Oct. 2017.

[17] D. Galvez-López and J. D. Tardos, “Bags of binary words for fast place recognition in image sequences,” *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188-1197, Oct. 2012.

[18] V. Lepetit, F. Moreno-Noguer, P. Fua, “EPnP: an accurate O(n) solution to the PnP problem.” *International Journal of Computer Vision*, vol. 81, no. 2, pp. 155–166, 2008.

[19] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel and J. D. Tardós, “ORB-SLAM3: an accurate open-source library for visual, visual–inertial, and multimap SLAM,” *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874-1890, Dec. 2021.

[20] R. Elvira, J. D. Tardós and J. M. M. Montiel, “ORBSLAM-Atlas: a robust and accurate multi-map system,” *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6253-6259, 2019.

[21] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers and W. Burgard, “An evaluation of the RGB-D SLAM system,” *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1691-1696, 2012.

[22] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3D Mapping with an RGB-D Camera," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 177-187, Feb. 2014.

[23] A. Hornung, K.M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: an efficient probabilistic 3D mapping framework based on Octrees," *Autonomous Robots*, 2013.

[24] "LS01B1 Rotating 2D LIDAR - 360°, 8 m," *RobotShop*. Retrieved May 2022, from https://www.robotshop.com/ca/en/ls01b1-rotating-2d-lidar-360-8-m.html.

[25] "Ultra Puck," *Velodyne Lidar*, https://velodynelidar.com/products/ultra-puck/.

[26] "C270 HD WEBCAM," *Logitech*. Retrieved May 2022, from https://www.logitech.com/en-ca/products/webcams/c270-hd-webcam.960-000694.html.

[27] "Intel® RealSense™ Depth Camera D405," *Intel® RealSense™*. Retrieved May 2022, from https://store.intelrealsense.com/buy-intel-realsense-depth-camera-d405.html.

[28] G. Bradski, "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, vol. 120, pp. 122-125, 2000.

[29] R. Klette, *Concise Computer Vision: An Introduction into Theory and Algorithms*, Springer Science & Business Media, pp. 59, 2014.

[30] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, 2004.

[31] H. Bay, T. Tuytelaars, L. Van Gool, "SURF: Speeded Up Robust Features," *ECCV 2006. Lecture Notes in Computer Science*, vol. 3951, pp. 404–417, 2006.

[32] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," *2011 International Conference on Computer Vision*, pp. 2564-2571, 2011.

[33] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D slam systems," *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, pp. 573-580, 2012.

[34] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

[35] A. Geiger, P. Lenz and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354-3361, 2012.

[36] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The KITTI dataset,” *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[37] I. Cvišić, I. Marković and I. Petrović, “SOFT2: Stereo Visual Odometry for Road Vehicles Based on a Point-to-Epipolar-Line Metric,” *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 273-288, Feb. 2023.

[38] P. Dellenbach, J.-E. Deschaud, B. Jacquet and F. Goulette, “CT-ICP: Real-time Elastic LiDAR Odometry with Loop Closure,” *2022 International Conference on Robotics and Automation (ICRA)*, pp. 5580-5586, 2022.

[39] Y. Zhou, G. Gallego and S. Shen, “Event-based stereo visual odometry”, *IEEE Transactions on Robotics*, vol. 37, no. 5, pp. 1433-1450, Oct. 2021.

[40] M. Ferrera, A. Eudes, J. Moras, M. Sanfourche and G. Le Besnerais, “OV^2SLAM: A Fully Online and Versatile Visual SLAM for Real-Time Applications,” in *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1399-1406, April 2021.

[41] Sardana, R., Karar, V. & Poddar, S. “Improving visual odometry pipeline with feedback from forward and backward motion estimates,” *Machine Vision and Applications*, vol. 34, pp. 24 , 2023.

[42] T. Ye and G. Zhao, “RT-SLAM:Real-Time Visual Dynamic Object Tracking SLAM,” *2023 IEEE 6th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 677-682, 2023.

[43] H. Yin, S. Li, Y. Tao, J. Guo and B. Huang, “Dynam-SLAM: An Accurate, Robust Stereo Visual-Inertial SLAM Method in Dynamic Environments,” in *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 289-308, Feb. 2023.

[44] M. Frosi and M. Matteucci, “D3VIL-SLAM: 3D Visual Inertial LiDAR SLAM for Outdoor Environments,” *2023 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1-7, 2023.