Line and Plane based Incremental Surface Reconstruction

by

Junaid Ahmad

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Simultaneous Localization and Mapping(SLAM) has been very popular in the past and is gaining more traction in the era of autonomous vehicle research and robot manipulation. Computing accurate surface models from sparse Visual SLAM 3D point clouds is difficult. There have been works where this problem was addressed by space carving methods using map points and lines generated by those points. These methods come with their own drawbacks as point clouds and lines alone don't add sufficient structural information to the scene. In this thesis, we try to take the natural step to also compute and verify 3D planes bottom-up from lines. Our system takes the real-time stream of new cameras and 3D points from a SLAM system and incrementally builds the 3D scene surface model. In previous work, 3D line segments were detected in relevant keyframes and were fed to the modeling algorithm for surface reconstruction. This method has an immediate drawback as some of the line segments generated in every keyframe are redundant and mark similar objects(shifted) creating clutter in the map. To avoid this issue, we track the 3D planes detected over keyframes for consistency and data association. Furthermore, the smoother and better-aligned model surfaces result in more photo-realistic rendering using keyframe texture images. Compared to other incremental real-time surface reconstruction methods, our model has less than half the triangles, and we achieve better metric reconstruction accuracy on the EuRoC MAV Benchmarks. We also tested our method on various off-the-shelf cameras for better generalization.

Preface

The inception of this thesis began with a course project on Model-based 3D tracking where I developed a RAPID-based 3D tracker with equal contribution from a project member Islam Ali. The 3D Computer Vision course (CMPUT-615) and the project got my interest in concepts like Bundle Adjustment and its application in SLAM systems for global mapping. Going through the literature I realized the importance of 3D scene reconstruction in a real-time system and how much can be done to improve it.

I started testing available codes on Line and Point based surface reconstruction algorithms on Benchmarks to analyze the drawbacks of these methods. My supervisor Dr. Martin Jägersand helped me with new ideas that could be used and tested on the SLAM systems. He motivated me towards introducing a plane-based approach to incremental surface reconstruction. We discussed and formulated ways to use planar information in scenes to optimize the modeling process and enhance the qualitative and quantitative accuracy of 3D reconstructed surfaces.

I conceived the idea of a new outlier removal method for 3D line segment extraction to remove depth inconsistencies from line segments. This helped me to get fewer and more relevant lines to detect planes in the scene. Under the tutelage of Dr. Martin Jägersand, I conceived the idea of detecting 3D planes from intersecting 3D lines. I contributed further to this idea by merging planes into major planar objects, validating and matching them over keyframes. These ideas are detailed in Chapter 3 of this thesis. I tested my method on EuRoC MAV data sequences and it outperformed existing point and line-based reconstruction methods. These experiments concluded in submission to *IEEE International Conference on Robotics and Automation (ICRA*

2023) titled *"Line and Plane based Incremental Surface Reconstruction"* co-authored by *Junaid Ahmad, Sait Akturk, Martin Jägersand*. Sait Akturk contributed to setting up experiments in the lab with off-the-shelf cameras.

Acknowledgements

First and foremost, I would like to give my special thanks to all the health workers in Canada and the rest of the world for their tireless work to serve humanity during this pandemic. I would like to thank Dr. B.P Singh and Dr. D.K Vatsal for treating my brother for Spinal tuberculosis (TB) for over 2 years. I feel deeply indebted to them.

I would like to thank my supervisor Dr. Martin Jagersand for his constant support and belief in my ideas. I would like to extend my gratitude to all the members of the Robotics and Vision Group for their constant support in maintaining the lab during these unforeseen times. I would like to thank my friends and labmates Chen, Kerrick, Michael, Dr. Mennatullah, Dr. Xuebin, Dr. Jun, Vincent, Laura, Sait, Faezeh, Jeramy, Jakub and Dhruv. I would also like to thank my closest friends Charvana, David, Javier, Ana, Lucia, Ivana, Eduardo, and Jim for making this master's journey full of life and fun.

Last but not least, I would like to thank my parents for their constant support despite the numerous pain and sufferings the pandemic brought to them. They have been my lightning rod throughout the course of my life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computing 3D surface models from images is an important yet difficult task to be accomplished. In the age of convolutional neural networks, we often forget the importance of the basic essence of geometry in images. One can train networks on a barrage of images and still don't completely comprehend what features are being detected and tracked. In this thesis, we study and propose a geometric approach to model a scene. These approaches rely on feature keypoints that lie on high gradient regions and can be detected and tracked over successive image frames. These methods were further enhanced by using high-level features like lines and planes. Lines and planes provide a more geometric understanding of the scene in the images that key points fail to represent. These properties play an important role in reconstructing 3D surfaces from a set of detected keypoints.

Reconstruction of 3D surfaces is an important task for a number of applications like augmented reality, mobile robot navigation, autonomous driving, robot manipulation tasks, etc. The arrival of methods like bundle adjustment made it possible to obtain 3D point clouds of scenes from a sequence of image frames. Bundle Adjustment methods iteratively optimize a closed solution whereas Structure from motion or Multi-View Stereo which makes use of N-view geometry to globally obtain the 3D points from different views. Some other approaches that use bundle adjustment for pose refinement are SLAM(Simultaneous Localization and Mapping) algorithms which use multiple images and cues in them to refine the pose of the camera over time. SLAM

algorithms can also provide sparse or dense 3D point clouds of the scene which can further be used for scene reconstruction [38]. The most commonly used SLAM systems like ORB-SLAM [42] and LSD-SLAM [13] can provide good enough 3D point clouds which can be used to reconstruct scenes incrementally. Most of these SLAM systems require GPUs to provide usable dense or semi-dense point clouds. However, ORB-SLAM can provide a reasonable semi-dense mapping of the scene which can work on systems with only CPUs.

In this thesis, we make use of semi-dense point clouds given by ORB-SLAM and use them to extract line and plane features from it for efficient scene reconstruction. Surface reconstruction directly from the semi-dense point cloud is expensive and not attainable in real-time using only CPUs.Another drawback of using the point clouds directly is that they do not hold any geometric or structural information about the scene. Hence, we simplify the point clouds by getting lines and planar structures out of it, filtering the heap of points given by the system.

The existing methods that involve structural information from the scene include model-based methods, segmentation-based methods, and methods with structural semantics[17], [40], [45], [68]. Most of these methods use segmentation methods to segment planes, spheres, cuboids, and shape profiles, which are expensive and require dense point clouds for proper segmentation for scene reconstruction. The plane from points methods majorly focuses on major planar surfaces like floors, ceilings, and walls using a learning-based method[26]. These methods pose challenges in modeling the scene in real time considering the complexity and memory required for point clouds. The advancement in line segment approach for surface reconstruction seems a less complex and robust option that provides a decent reconstruction in real-time [4], [22], [49]. Some of the mentioned methods try to combine the lines and planes to avoid using points to extract planes from keyframes [26], [68], [71]. These methods use line segment extraction methods that involve inter-keyframe matching and evaluation of 3D line segments which could pose problems in complex scenes due to depth uncertainty. This issue can be avoided using semi-dense (CPU capable) or dense point clouds (GPU required) generated from existing SLAM

systems.

We can classify these methods based on the features used for the reconstruction of 3D surfaces: point-based, line-based, plane-based, and hybrid methods involving two or more of these features. Point-based approaches use a SLAM or SFM system to get 3D points to reconstruct 3D meshes using a voxel-based (TSDF) or 3D Delaunay Triangulation approach. These methods use a large number of points and don't have a reliable outlier removal technique [23], [38]. Line-based approaches reduce the overload of points by using only line segments to reconstruct surfaces [20], [22]. These methods do not have a reliable clustering or line-matching scheme which results in the depth inconsistency of 3D line segments detected. It also does not guarantee an online surface reconstruction system as their clustering algorithms are computationally complex [21]. Plane-based approaches usually rely on a decent segmentation/clustering algorithm to segment the point cloud [53] or pixels[61] into planar and non-planar regions. They have a better outlier removal technique resulting in superior qualitative reconstructions. Unfortunately, these techniques are computationally complex and require GPU acceleration to achieve real-time performance. Hybrid methods are a combination of point, line, and plane-based approaches. The most common combination in many works is lines and planes [32], [71]. Detecting 3D planes from lines reduces the complexity of plane detection over RANSAC and Segmentation-based approaches. The plane and line matching in consecutive keyframes reduce outliers providing more accurate points to the reconstruction algorithm. However, it is important to constrain the number of planes detected to avoid redundancy and speed up the reconstruction. Figure 1.1 shows the taxonomy of 3D reconstruction methods based on features and the most common characteristics in related works.

Here, we present a novel hybrid approach that combines the line segment approach with point-plane approach for reconstructing the scene incrementally. We reconstruct planes from intersecting 3D line segments. To detect 3D line segments, we use a line segment detection approach similar to [21] where He *et al.* use edge segments to fit 3D lines using the image and the depth

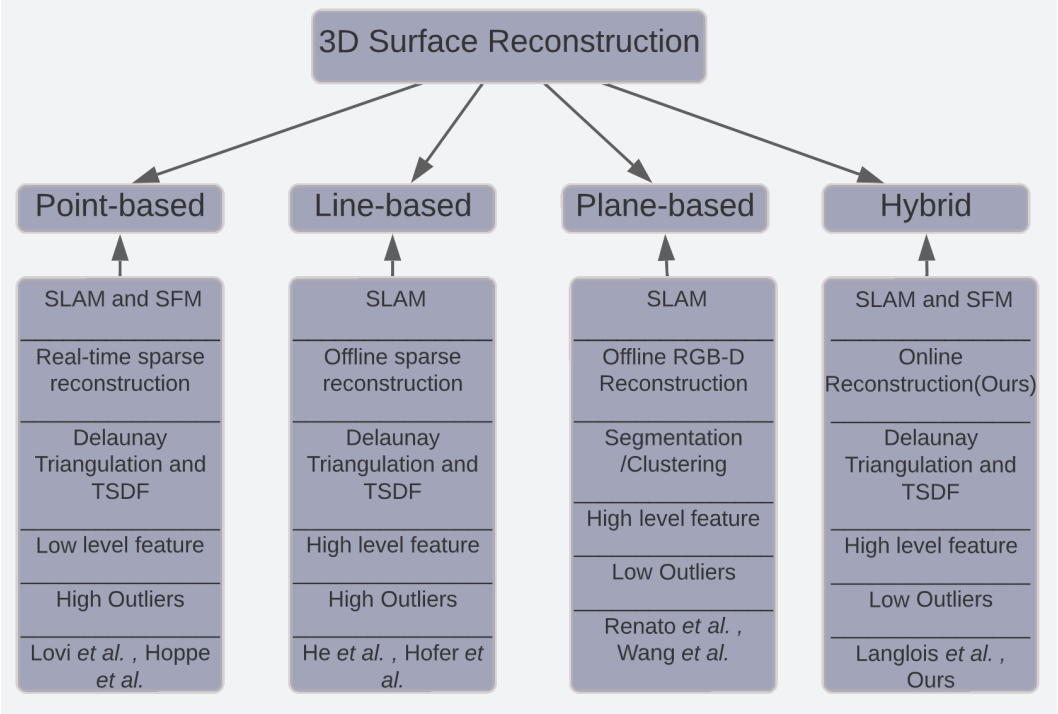


Figure 1.1: Taxonomy of 3D reconstruction approaches. The list below these approaches states the most common features of the reviewed methods.

plane. We improved on this method by checking for planar points only while line fitting as their method failed to address whether the fitted lines belonged to a single plane. Lines passing through multiple planes would result in erroneous surface reconstruction. We then extract planes from these line segments using only intersecting lines for cluttered scenes. The extracted planes are then validated and merged into larger planar surfaces. The planes extracted are further reduced by inter-keyframe plane and line matching. This reduces the expenses of reconstruction by only using relevant lines/ points for surface reconstruction.

We divide this thesis into 5 chapters. Chapter 1 introduces and sheds light on the motivation behind our work. Chapter 2 is a survey of the related works in geometric computer vision. We discuss our proposed method and experiments in detail in Chapter 3 and Chapter 4 respectively. Chapter 5 concludes our thesis with motivation for future research possibilities in this direction.

Chapter 2

Related Works

In this thesis, we mainly focus on monocular SLAM systems (sparse and semi-dense) for surface reconstruction. We detect planar structures in the scene aided by intersecting line segments in 3D space. We perform matching, validation, and merging of planes into major planar regions in the scene. Our method can be extended to stereo and RGB-D with minimal adaptation.

In this Chapter, we discuss the background of our work and detail the works relevant to our approach. We begin with expanding on SLAM systems (Section 2.1) and their application in robotics and vision. We also discuss the sparse, dense, and semi-dense versions of the SLAM system in subsections 2.2.1, 2.2.2, and 2.2.3 respectively. In Section 2.3, we explore the 3D line segment fitting approaches relevant to our method and expand on their limitations. We also elaborate on the previous works which use a plane-based approach (Section 2.4) for scene modeling and reconstruction. Our major contribution combines these two ideologies in a novel way for incremental surface reconstruction. We also discuss approaches for surface reconstruction listing their advantages and disadvantages which made us choose a specific approach.

2.1 Simultaneous Localization and Mapping

A SLAM system aims at tracking multiple feature key points from images obtained by a moving camera (mobile robot, drone) to estimate the pose of the camera. The camera pose is usually estimated using a constant velocity motion model. With the estimated camera pose a 3D map of the tracked

points is triangulated. The camera pose and the 3D map is further refined using a non-linear optimization famously known as 'Bundle Adjustment'[59]. SLAM systems are classified into Direct and Indirect (feature-based) SLAM.

2.1.1 Direct SLAM

Direct SLAM methods make use of the pixel information directly without any feature detection [12], [13], [18], [47]. These methods are more accurate and robust in scenes with less texture making it difficult to extract useful features from the scene. These methods rely on the minimization of the photometric error to compute the depth associated with each pixel. The photometric error between two images with pixel values $\mathbf{x_i}$ and estimated depth values $\mathbf{Z_i}$ can be defined as follows:

$$E_{photo}(\rho) = \sum_i ||I_{ref}(x_i) - I(\omega(x_i, Z_i, \rho_i)||^2 \tag{2.1}$$

where $I : \mathbb{R}^2 \rightarrow \mathbb{R}^+$ returns the pixel values for the given coordinates. $\boldsymbol{\omega}$ projects a point from the reference frame to the new frame. $\boldsymbol{\rho}$ is the camera parameters to be estimated. The main assumption of this error is brightness constancy, lambertian surfaces, and no lens distortion. This approach is incorporated by Engel *et al.* [12], to show that this error can be used in a sparse set of points from images to to perform sliding window photometric bundle adjustment in real time on CPUs.

2.1.2 Indirect SLAM

These methods use feature descriptors to get reliable 2D keypoints that can be detected and tracked in images from different views of the scene [28], [29], [43]. The features are a combination of the keypoints and their respective descriptors. This reduces the image space to just these reliable features and the other pixels can be ignored for tracking. These methods rely on the minimization of reprojection error which has better convergence properties than direct methods. To estimate the camera parameters $\boldsymbol{\rho}$, the minimization of

the distance between $\mathbf{x_j^i}$ and $\hat{\mathbf{x}}_\mathbf{j}^\mathbf{i}$ is given by:

$$E_{reproj}(\rho) = \sum_{R^i, t^i, X_j} ||x_j^i - (R^i X_j + t^i)||^2 \tag{2.2}$$

where $\mathbf{i}$ is the number of views and $\mathbf{j}$ is the number of keypoints in each view. $\mathbf{R} \in \mathbf{SO(3)}$ and $\mathbf{t}$ is the translation vector. These methods work well in scenes rich in texture consisting of edges and corners. The work of Raúl *et al.* [43] uses this method to compute camera poses and form sparse and semi-dense maps in real-time. We built upon this system for incremental surface reconstruction using planes in 3D space.

2.2 Monocular SLAM

Monocular SLAM is a type of SLAM that relies on images from a single moving camera. The first monocular SLAMs used filtering methods like Extended Kalman Filter (EKF) to estimate the camera pose and maps of the scene by passing every frame through the filter. These methods were computationally very expensive and also resulted in error accumulation due to the linearized rotation assumptions. Due to this, Keyframe based approaches became more popular as it significantly reduced the number of frames used by the system for map generation [7]. The first monocular SLAM capable of real-time performance was MonoSLAM [10] that uses EKF for refining camera pose and depth of the probabilistic features. We will discuss the advancement of monocular SLAMs from offline to online systems in the following subsections.

Generally, both classes of SLAM mentioned above are capable of producing sparse, dense, and semi-dense point clouds from the scene at different processing times. Monocular SLAMs can further be classified into Sparse, Dense, and Semi-dense Monocular SLAMs [7].

2.2.1 Sparse Monocular SLAM

In the section above we discussed that filtering-based methods were expensive so the keyframe-based approach was adopted. This idea was further optimized by using parallel threads for camera tracking and mapping of the scene in

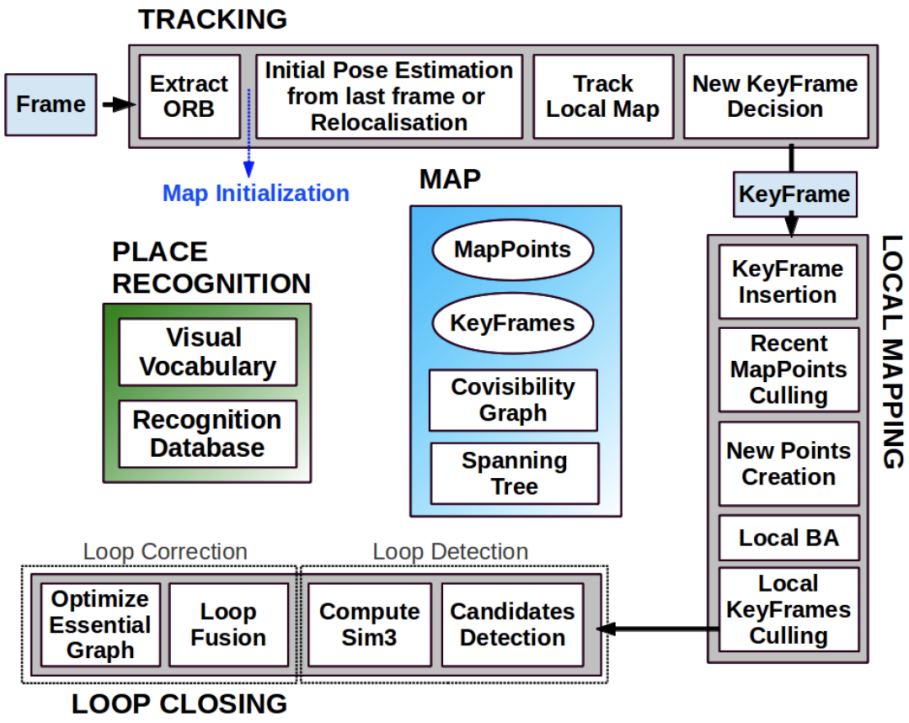


Figure 2.1: Overview of ORB-SLAM [44]

PTAM [29] which was further improved with better relocalization and loop closure [28]. This system generates sparse point clouds in real-time using the multi-threaded approach. The tracking thread tracks all the keypoints using FAST feature matching and the mapping threads collect this matched information to triangulate them into 3D points. The mapping thread also performs local Bundle Adjustment (BA) [59] to refine the poses of the camera and the tracked points in the keyframes. It also performs global BA for global pose refinement when a certain threshold in the pose error has been reached.

The work of Raúl *et al.* [44] further improved this multi-threaded system by using ORB feature descriptor [52], using a local map based on covisibility graph, building pose graphs out of it for pose graph estimation and using Bag of Words (DBoW2) for place recognition to aid loop closing. Fig 2.1 shows the overview of the ORB-SLAM system. Unlike PTAM, ORB-SLAM has an extra thread exclusively assigned for loop closing which makes the process faster and more robust in real-time applications. It makes use of the Pose Graph Optimization technique instead of BA as it is too expensive

to perform BA and inaccurate when the initial solution is far from the real solution. They use relative transformation error between two camera poses and represent them in a graph structure. They also use Sim(3) instead of SE(3) or SO(3) to account for scale drift in monocular systems. This graph structure is then optimized using non-linear optimization methods like Gauss-Newton or a weighted version of it known as Levenberg-Marquardt Optimization. The cost function is defined as follows:

$$C = \sum_{(i,j)\in\chi} \rho(\| \ e_{rel}(i,j) \ \|^2_{\sum ij}) \tag{2.3}$$

where $\mathbf{e_{rel}}$ is the relative pose error between camera/view **i** and **j**. $\sum \mathbf{ij}$ represents the covariance which is used in Mahalanobis distance metric. The pose graph optimization is performed using the g2o library but Ceres-solver is an excellent alternative for non-linear optimization.

We must discuss the robust mapping of ORB-SLAM as we heavily use it in our system and justify why it is a better choice than LSD-SLAM and PTAM [13], [28]. ORB-SLAM creates many keyframes and mappoints after every keypoint matching and 50-frame parsing. These keyframes and mappoints would be an overload if they are not filtered. Hence, they perform culling of mappoints and keyframes in the local mapping. This reduces the memory overload and leaves the system with well-constrained and efficient keyframes and mappoints. This makes the process more robust and efficient than the other two methods mentioned above. Unlike ORB-SLAM, LSD-SLAM and PTAM do not use covisibility graphs which constrain the keyframes and mappoints to a locally covisible area. Fig 2.2 shows the generation of the sparse point cloud from the EuRoC MAV Vicon Room 101 dataset [6].

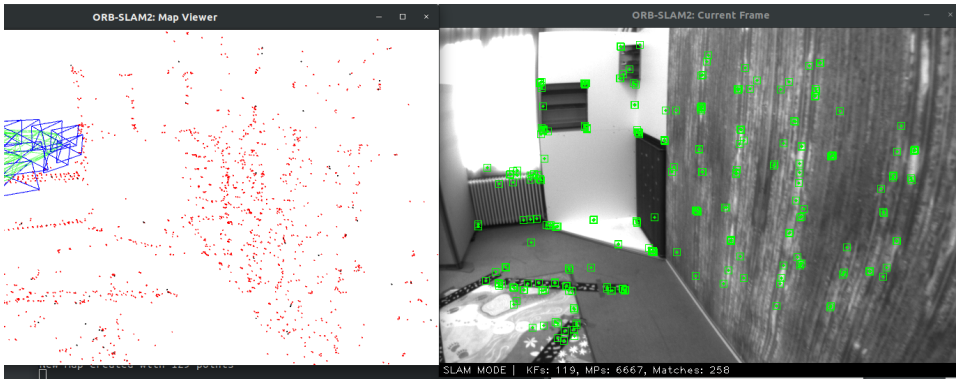


Figure 2.2: Sample sparse point cloud by ORB-SLAM [43]

2.2.2 Dense Monocular SLAM

As discussed briefly in Section 2.1.1, direct SLAM methods can produce dense point clouds if a larger number of pixels are used for pose estimation. These methods do not require feature extraction and thus avoid finding keypoint correspondences. These dense reconstruction methods can be substantially useful in scene or object reconstruction but it is hard to make these methods work in real-time. DTAM [47] and LSD-SLAM [13] are one of the best methods in reconstructing dense point clouds while localizing the camera by photometric optimization (see equation 2.1).

DTAM is one of the first dense reconstruction methods that work at frame rate on GPU. It relies on refining of depth maps for smoothing using a regularized approach. The pose refinement is performed by image alignment to minimize the photometric error between the reference frame and the current frame. There are newer methods inspired by this approach and volumetric point fusion technique that give better results at a frame rate as well [15], [48], [65]. Polarimetric Dense Monocular SLAM is an interesting and progressive work on dense reconstruction which makes use of DSO for pose estimation. It uses a polarization camera to capture the scene under 4 polarization angles. A depth map is initialized using stereo matching on image patches. The matched depth is further optimized by minimizing a combined energy function of photometric, polarimetric, and Spatio-temporal regularization constraints. Figure 2.3 draws the comparison between their method and MonoFusion.

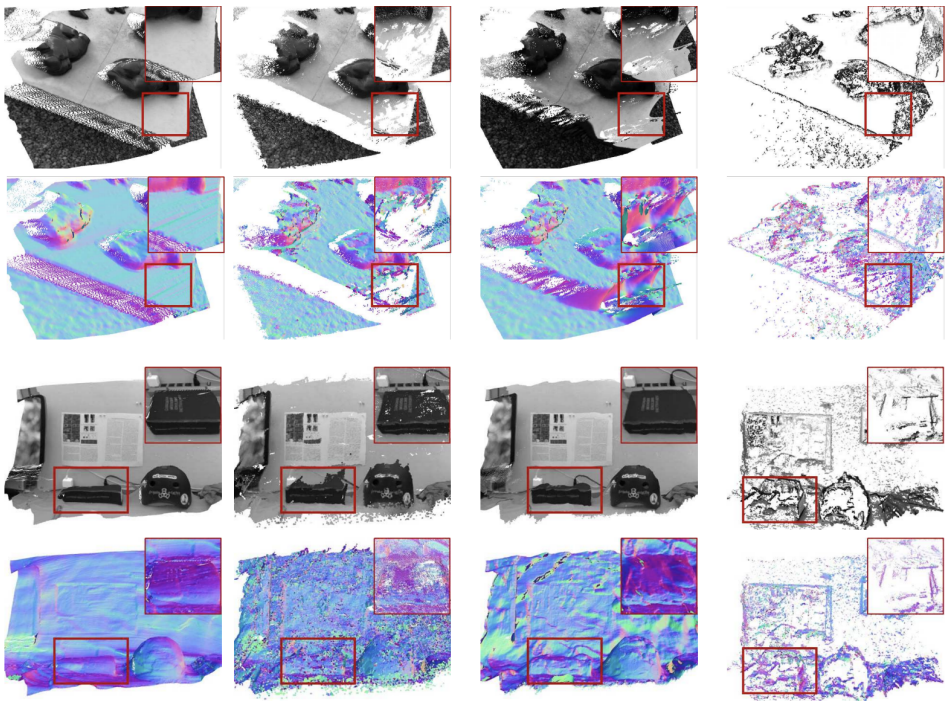


Figure 2.3: Qualitative comparison of Polarimetric[65] against MonoFusion[48] and Remode. i) Polarimetric ii) MonoFusion iii) Polarimetric+MonoFusion iv) Remode

2.2.3 Semi-dense Monocular SLAM

Semi-dense map is a point cloud that contains more points in high gradient areas of the map than a sparse map but much lesser points than a dense map. Semi-dense maps are more useful than dense maps as they have lesser noise and can be reconstructed cheaply. If generated in real-time can be useful for several robotic applications. Systems like DTAM [47], LSD-SLAM [13], and ORB-SLAM with Probability Mapping [42] were able to do so in real-time without GPU acceleration. All of these methods make use of the inverse depth parametrization [8] in one way or the other. DTAM was the first algorithm to make use of inverse depth parametrization in its regularized cost function as follows:

$$\min_{n} \sum_{i=2}^{n} \int_{\Omega} |I_1(x) - I_i(\omega(x, u, \rho_i)| dx + \lambda \int_{\Omega} \psi(x) |\nabla u| dx \tag{2.4}$$

where all the parameters are similar to Equation 2.1 except $\mathbf{u} = \mathbf{1/Z}$ is the inverse depth, $\psi(\mathbf{x})$ assigns small weights to strong gradient changes so that

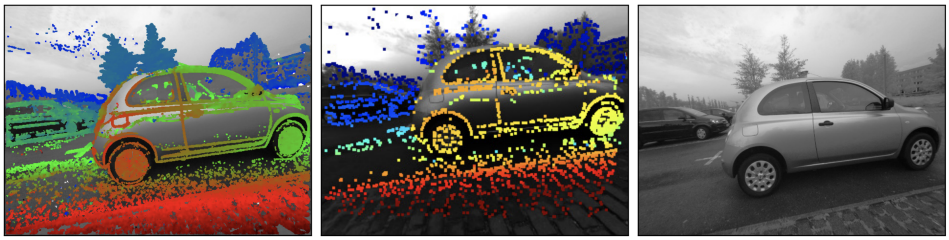


Figure 2.4: Qualitative comparison of semi-dense LSD-SLAM[65] against sparse DSO. i) Semi-desne LSD-SLAM ii) Sparse DSO iii) Original Keyframe

large variations in $\nabla \mathbf{u}$ doesn't explode the cost function. $\mathbf{\Omega}$ here is the pixel space.

The reason the inverse depth parametrization is so effective is that structures that are far away (large depth values) correspond to substantially small pixel space whereas surfaces that are in close proximity correspond to larger pixel space. Hence, using inverse depth in error parametrization and regularizer alleviates this bias. The biggest challenge of these approaches for dense reconstruction is that it requires large-scale optimization to compute the depth of every corresponding pixel. LSD-SLAM uses a similar approach for its semi-dense reconstruction but it only considers pixels with gradient values above a certain threshold. Each inverse depth value calculated is associated with an uncertainty value that is propagated over time pruning out the inverse depths that are uncertain. Fig 2.4 shows an example of semi-dense reconstruction using LSD-SLAM vs sparse reconstruction by DSO [12].

Another efficient method of semi-dense reconstruction is ORB-SLAM's Probabilistic semi-dense mapping [42] which adds a fourth thread to its original ORB-SLAM system [44] for generating semi-dense point clouds. This thread is only used exclusively for extracting 3D points and not refining the camera poses as it is assumed to be already refined from local mapping. Like LSD-SLAM, the search for high gradient points is made but along the epipolar lines on neighboring keyframes. An inverse depth hypothesis (Gaussian Distribution) is proposed for every neighboring keyframe which is then smoothed using an approach similar to LSD-SLAM. Since this method uses a larger baseline than [13], there can be significantly higher outliers. Hence, it strictly com-

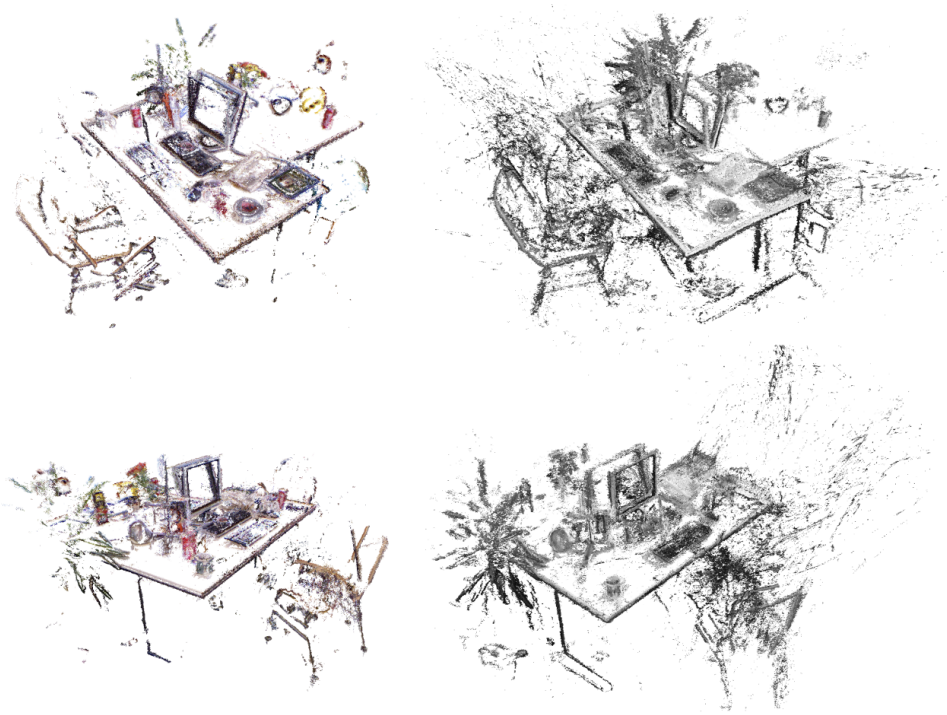


Figure 2.5: ORB-SLAM vs LSD-SLAM in dynamic scene. Left column is ORB-SLAM output and Right column in LSD-SLAM output i) Row 1 is Static scene ii)Row 2 Object shifted from scene

pares the gradients and their orientation instead of comparing pixel intensities directly over the epipolar lines. The pixels are then compared using a similarity function that combines photometric error and gradient error. Another reason this method is superior to [13] is that they included an inter-keyframe depth checking module which removes outliers by projecting the pixels with inverse depth values into neighboring keyframes and checking if it matches the inverse depths in the neighborhood of 4 pixels. This is the reason we prefer this method over LSD-SLAM for our scene reconstruction system. Fig 2.5 shows the comparison between ORB-SLAM and LSD-SLAM semi-dense reconstruction in a dynamic scene where a person shifts objects in a scene.

2.3 3D Lines from points

The features in images are mainly classified into two categories: low-level and high-level. Low-level features are the ones that directly involve the pixel and

its intensity value. High-level features are the ones that add more meaning to the images in terms of geometry or structure. Key Points, blobs, and edges are considered low-level features whereas lines, planes, shapes, and profiles are considered as high-level features. They are considered high-level features because they contain more information about the scene. The SLAM algorithms discussed above primarily focus on the generation of sparse, dense, or semi-dense point clouds of the scene. However, it is hard to use these 3D point clouds directly for robotic tasks or surface reconstruction as it does not explain the scene properly. Therefore, we ponder on methods to extract more information about the scene from these 3D point clouds. Lines are one such feature that preserves the structural information of the scene better than 3D points in the point cloud. Extracting 3D line segments from 3D points is not difficult but extracting meaningful lines from the scene is quite challenging. In this section, we will discuss several methods that have been lately used for detecting meaningful 3D lines and discuss their advantages and limitations.

There have been works that generate 3D lines from 2D images and match them to generate structures [54], [55]. These works produce results that look promising for surface reconstruction or scene modeling but they fail to work at a frame rate in video sequences. There have been advancements in the line matching methods where we can see methods able to produce efficient line matching at frame-rate [62], [69]. LBD combined with LSD [18] has been most commonly used in stereo-matching techniques for generating 3D line segments. Lines being important structures are used in various line-based SLAM systems as well as SFM (Structure from Motion) systems where lines are used as major landmarks [4], [17], [21], [41], [49], [50], [70], [72]. These methods use lines differently and also detect them in a variety of ways.

The most challenging part is detecting 3D lines from the 2D information given in the image sequences. Some methods like Hough Line Transform [11], LSD [18], and EDLines [2] use rudimentary edge detectors to fit 2D lines to the edges. The lines-based SLAM methods use these line segment detectors with line descriptors like LBD and MLSD to triangulate 3D lines. However, these methods fail to consider that the line segments detected in 2D may not

actually correspond to 3D structures leading to a large number of outliers. Given the point cloud of the scene, we can also think about detecting line segments directly from the scene using 3D points. Xiaohu *et al.* [39] describe how 3D lines can be extracted directly from the point clouds using a robust RANSAC-based approach but this method fails in semi-dense SLAM systems where the earlier keyframes have fewer 3D points. It results in a few or no lines at the initial keyframes and more or almost all the lines toward the end of the keyframes. Hence, the lines detected in 3D space do not have sufficient keyframe correspondence to represent real physical structure which makes this approach unreliable in incremental surface reconstruction.

2.3.1 3D lines fitting using Line matching

SLAM methods that use multi-view stereo make use of 2D line matching techniques. Kun *et al.* [50] use LSD [18] to detect lines in every frame and perform stereo matching on the matched lines using LBD [69] detectors directly. They also use the error from the stereo matching to refine the pose in the bundle adjustment [50]. Their method works well in stereo SLAM but is hard to implement in the monocular version. Using per-frame line matching is also expensive and can be avoided using only inter-keyframe line matching. The limitation of using line descriptors is that they are mostly patch-based and can only work accurately for lines on a planar surface. These methods fail to address the depth discontinuity of lines detected by rudimentary line detectors.

Hofer *et al.* [22] addresses this problem by obtaining 2D line correspondences based on their known epipolar geometry [22]. They find the epipolar lines of the endpoints of the line to be matched and intersect them with the line (using a line at infinity approach [19]) in the neighboring frame. The intersecting points obtained are used to form a similarity score for lines. Fig 2.6 shows the line-matching process explained above.

Another approach is the work of Hang *et al.* [72] uses a purely geometric approach for evaluating the direction vector and depth of the points on the line. They use rudimentary 2D line detectors and descriptors for matching 2D lines. Once the lines have been matched in more than 2 keyframes they back-

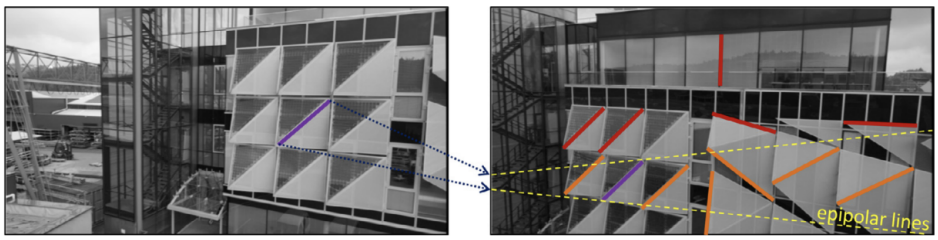


Figure 2.6: Line matching using epipolar constraints. The purple line on the left is matched with the purple line on the left. The epipolar lines in yellow are drawn from the endpoints of the line in the left image [22].

project the line onto the projective camera planes. Since parameterization of the 3D lines is a complicated task they use the property that a family of planes intersects at one line. The normal vectors of these planes are used to solve the least square minimization to get the direction of the lines. The least squares formulation was designed as follows:

$$e_d = \min_d \sum_i |(R_i^{-1} n_i)\vec{d}|^2 \tag{2.5}$$

where $\mathbf{R_i^{-1}}$ is the transformation from camera frame to world frame, $\mathbf{n_i}$ is the normal of the projective planes in the camera frame, and $\vec{\mathbf{d}}$ is the direction vector of the 3D line. Similarly, the points on the lines are reprojected and the reprojection is minimized to get the depth values of the points. This method also comes with the same drawback that the line matching would not result in a decent number of lines as the 2D line detectors and descriptors assume lines in the same 3D planar region. Figure 2.7 compares the point cloud with the lines extracted by their method.

2.3.2 3D Line fitting using 3D points aided by 2D cues

Given the dense or semi-dense point cloud from SLAM or SFM, we can use these points to fit lines by constraining the fitting algorithm with different cues. Woo *et al.* [64] made use of the prior model of the building to get 3D lines from the 2D lines fit the aerial images. The line correspondences were found with the Digital Elevation model (DEM) of the building. Nakayama *et al.* [45] makes use of RGB-D sensor to get approximate depth information

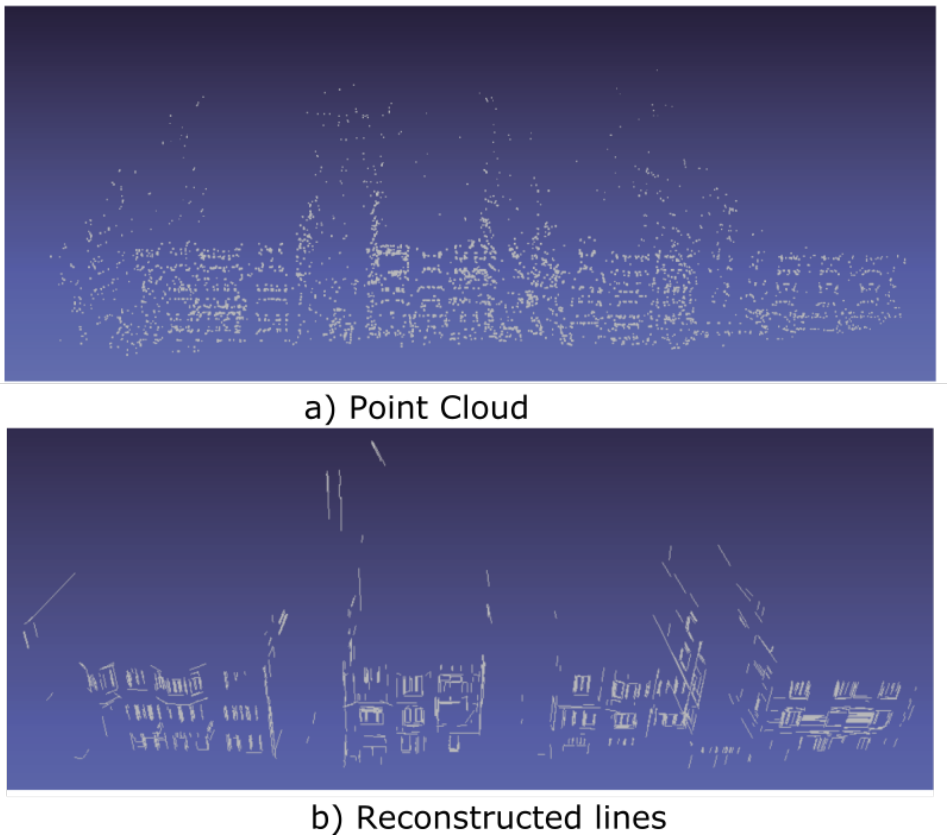
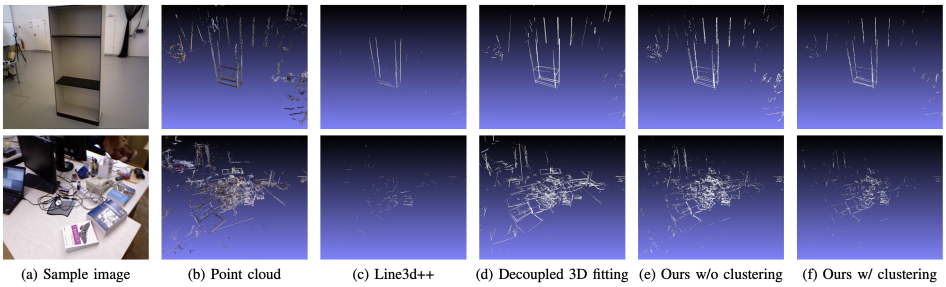


Figure 2.7: Lines reconstructed by Hang *et al.* [72]

from the 2D points on the line. These 3D points are then used to fit 3D lines in 3D space using a RANSAC-type approach. He *et al.* [21] also follows a similar approach to get 3D lines but on monocular SLAM. They make use of semi-dense SLAM points to get 3D points from the 2D points on lines in the images. They only choose pixel chains that have depth values associated with it and fit two 2D lines: one in the image plane and one in the projective plane with depth values. This system overcomes the limitations of the methods that use 2D line matching methods. This method uses Edge Drawing [2] to get the edge maps in the form of pixel chains for line fitting. The filtered keychains are passed to the line fitting algorithm which uses a robust Total Least Square method to fit the line on the points. They also consider outlier detection using a distance metric that marks points as outliers if they are not close enough to both 2D lines (image plane and depth plane). They further go on to cluster the lines to remove the clutter of lines detected around the high gradient areas.



(a) Sample image (b) Point cloud (c) Line3d++ (d) Decoupled 3D fitting (e) Ours w/o clustering (f) Ours w/ clustering

Figure 2.8: Qualitative Analysis of He *et al.* [21] against Line3d++ [22]

This clustering algorithm is based on the angle and distance threshold of lines with neighboring lines. Fig 2.8 shows the qualitative analysis of their method against other methods.

We improve upon this method to use it in our plane extraction method. This method gives promising results in scene reconstruction as well but it comes with its own limitations. The main drawback of this method is that it does not address the fact the lines fitted in this algorithm may be inaccurate as they contain points that lie in different planes. In indoor scenes which contain man-made objects, this method fails to ensure that lines represent real edges of the objects. Clustering is not a viable approach for this method as the lines at the cluster centroid may not always lie on the edges of an object. Inter-Keyframe line matching or tracking is important as the abundance of lines around the same edges is due to the fact that the same lines are visible at different positions in different Keyframes. This causes multiple small lines around the untracked major line. We address these problems in our approach and add a coplanarity threshold on the pixel chains during line fitting. We also replace the clustering algorithm with plane (generated from lines) tracking and use only those lines that were observed in more than 2 Keyframes.

2.4 3D Plane Reconstruction

As we discussed earlier, lines and planes are high-level features as they provide more structural information about the scene than points. The use of lines for structural information comes with its own drawbacks. The 3D line reconstruction methods discussed in 2.3 do not account for planar information and

hence do not necessarily represent a 3D structure in the world frame. Lines are detected based on high gradient pixels which can be present on a highly textured surface that lies in the same plane. This problem can be addressed by filtering the lines and points by adding planar information to them. These filtered lines and points can be then used to reconstruct surfaces from the scene making the scene reconstruction more robust and accurate.

In a 3D world coordinate system, a plane can be defined by 3 non-collinear points. But doing an exhaustive plane search on all dense or semi-dense points is an expensive task and is not feasible in a real-time system. There are approaches that use a randomized sampling method as well as segmentation-based approaches to extract planar structures from the scene. Plane matching has been used to refine the pose of the camera, reducing the drift error and some methods use planes to reconstruct the 3D scene [16], [26], [40], [68], [71]. Most of these works focus on 3D Plane reconstruction directly from the point followed by segmentation and image alignment.

Early methods relied on the photometric information to disambiguate planes that did not represent real surfaces [3]. Later, scene understanding was used to enhance the performance of visual SLAM systems and dense mapping using planar modeling of the scene [67]. Some approaches rely on segmenting the pixels into superpixels or surfels and clustering them into major planar regions [53], [60], [61]. These methods provide quite promising results of incremental mapping which can be used for augmented reality applications.

Unlike these methods, our method does not focus on dense mapping of the scene incrementally. We reconstruct surfaces incrementally using planes and lines derived from a semi-dense point cloud of the scene which is discussed in 2.5. In the following subsections, we will discuss the related methods used for plane reconstruction.

2.4.1 3D Planes fitting using Segmentation

Most of the systems we discussed above work with RGB-D sensors [24], [26], [40], [53]. These approaches segment the point cloud generated by the sensor using a common point cloud segmentation method [58]. Lingni *et al.* [40] de-

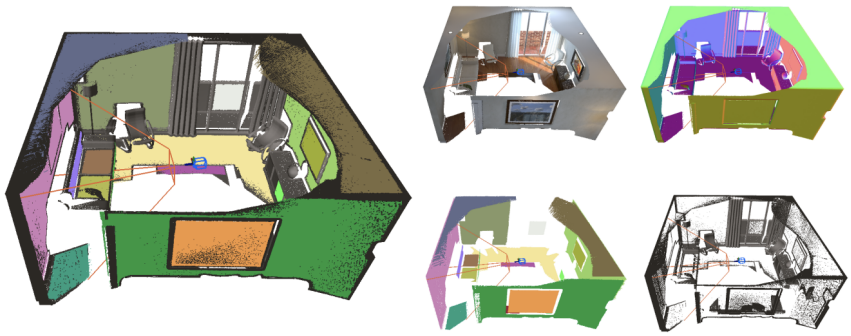


Figure 2.9: Synthetic Scene Reconstruction [53]. Left: (planar+non-planar). Right: The first row shows the color output and normal map, the Second row shows the non-planar regions and planar regions.

fines a plane model which represents flat surfaces in the scene. They segment every keyframe into planar and non-planar regions using agglomerative hierarchical clustering algorithms. A least square fit is done to the planar segments detected in those keyframes to get the plane parameters. They perform plane matching (using distance and angle thresholds) to use it in the bundle adjustment for pose refinement. Mehdi *et al.* [24] segment the point cloud from RGB-D data similar to the previous approach but they also smooth the segment neighborhood for calculating the surface normals. Renato *et al.* [53] uses a similarity measure on a group of pixels (surfels) to assign labels to them that belong with similar plane normals and distances. These clustered segments are then used to fit planes using Principal Component Analysis (PCA). Figure 2.9 contrasts the synthetic reconstruction of the planar and non-planar regions.

The plane reconstruction has also been extended to monocular SLAM systems for scene mapping [61], [66]–[68]. Shichao *et al.* [66] model the scene with a set of cuboid objects and major planes like walls, floor, and ceilings. They segment the planes by detecting edges and choosing only those near a ground-wall boundary by semantic segmentation using a learning-based approach. These 2D plane edges are back-projected to the 3D space. The back projection of 3D points is given by the intersection of the ray passing through the point and the ground plane. Wang *et al.* [61] associate multiple trackers to planar segments of the images called superpixels. These superpixels

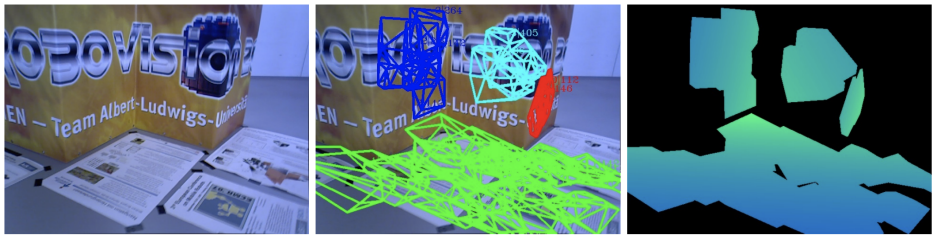


Figure 2.10: Plane Reconstruction by TTSLAM [61]. Left:Original Keyframe, Middle: Template trackers and plane clusters, Right: 3D Reconstructed plane clusters

are segmented using Simple Linear Iterative Clustering (SLIC) [1] per frame. A template tracker is assigned to each superpixel and a set of homography transforms is obtained which is later decomposed into camera pose and plane parameters $(\vec{\mathbf{n}}, \mathbf{d})$. The plane normals are then clustered to obtain a simplified multi-planar representation. The homography decomposition can be represented by:

$$^{2}H_1 = {}^{2}R_1 + \frac{^{2}t_1}{d} n_1^T \tag{2.6}$$

where $(\mathbf{R}, \mathbf{t})$ are pose parameters and $(\mathbf{n^T}, \mathbf{d})$ are plane parameters. The plane clusters are further checked for fusion with neighboring planes in every keyframe based on a plane's normal and distance thresholds. Fig 2.10 shows the tracking, clustering, and depth reconstruction of the planes. Their previous work [60] on pose estimation using planar structures used a RANSAC-based approach to remove outliers in plane clustering from multiple homographies. They address the issue of pseudo-outliers i.e if an object has multiple instances in the scene model. They invalidate all the points in the superpixel region if it is above a certain inlier threshold claiming that it belongs to a distinct plane. This prevents constant overlapping of points in other plane homography reducing the redundancy.

2.4.2 3D Planes from points and lines

Taguchi *et al.*[56] proposed a RANSAC-based approach to extract planes and match them in keyframes. They work on a handheld RGB-D sensor and make use of the depth map provided by it. They randomly select reference points in

the point cloud and fit planes using neighboring points. The pixel is labeled as inliers and outliers based on the distance of the points from the extracted planes. Later, the planes are filtered out by only selecting planes with more than a certain inlier threshold. They later use point-to-point and plane-to-plane correspondences to find the pose of the camera. The point-to-point correspondence is solved by decoupling the rotation and translation components of the pose. For any point set $\mathbf{p}$, $\bar{\mathbf{p}}$ and $\bar{\mathbf{p}}'$ are the centroids of 3D point sets. $\bar{\mathbf{p}} = \frac{\mathbf{1}}{\mathbf{M}} \sum_{\mathbf{i}} \mathbf{p_i}$ and $\bar{\mathbf{p}}' = \frac{\mathbf{1}}{\mathbf{M}} \sum_{\mathbf{i}} \mathbf{p'_i}$. Let the correspondences be $\mathbf{q_i} = \mathbf{p_i} - \bar{\mathbf{p}}$ and $\mathbf{q'_i} = \mathbf{p'_i} - \bar{\mathbf{p}}'$. So the least square formulation for decoupled rotation and translation can be written as follows:

$$\min_{R} \sum_{i} ||q'_i - Rq_i||^2 \tag{2.7}$$

The rotation is calculated first and then the translation is evaluated as the difference between rotated centroids.

$$\hat{t} = \bar{p}' - \hat{R}\bar{p} \tag{2.8}$$

Similarly, the plane-to-plane correspondence is evaluated by minimizing the difference in rotated normals.

$$\min_{R} \sum_{j} ||n'_j - Rn_j||^2 \tag{2.9}$$

Here, the translation can be solved by solving a system of equations formed by 3 or more planes under a linear constraint.

$$n'^T_j t = d_j - d'_j \tag{2.10}$$

Fig 2.11 shows the scene reconstruction from their proposed method.

Andrew *et al.* [16] proposed a similar approach with the main aim to reduce the state space in tracking 3D points in conventional systems. Instead of tracking all 3D points, planar structural components are used to represent the mapped points on a common planar surface. This approach highly reduces the state space and adds more meaning to the scene modeling. A RANSAC-based approach is used to randomly sample points on planes from the current

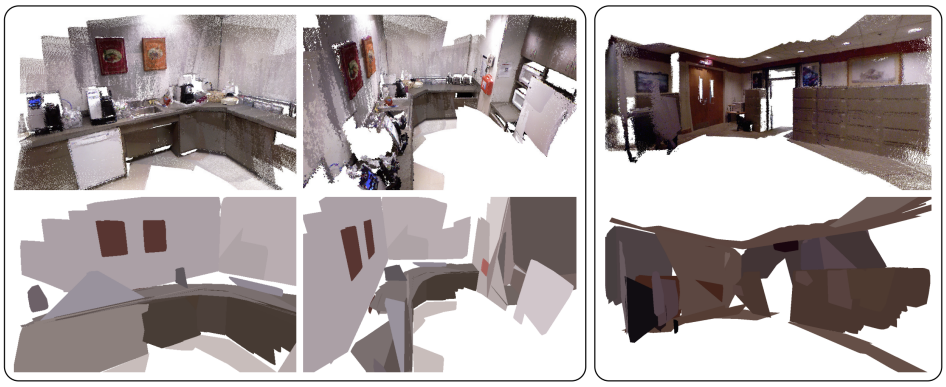


Figure 2.11: Scene Reconstruction by Taguchi *et al.* [56]. Top: Point Cloud, Bottom: Planar scene reconstruction

map. A plane hypothesis is generated from the subset of randomly sampled points under a chosen variance threshold in each dimension. A minimal set of 3-point features is used to get the normal $\vec{\mathbf{n}}^T$ and distance from origin $\mathbf{d}$ of the plane. The plane representation used here is a 7-parameter state vector $\mathbf{m} = [\mathbf{p_0}, \boldsymbol{\theta_1}, \boldsymbol{\phi_1}, \boldsymbol{\theta_2}, \boldsymbol{\phi_2}]$ where $\mathbf{p_0}$ is the plane origin and the orientation is the cross product of two basis vectors on the plane. A consensus set is formed from the points in the point set. A point is considered as an inlier of the consensus set if the perpendicular distance from the plane is less than a distance threshold and its Euclidean distance from the plane is less than a certain threshold. This makes sure that the points are not too far apart. The plane hypothesis with the most inliers in the consensus set from a subset of randomly sampled points is considered as the best-fit plane. The distance from origin $\mathbf{p_0}$ is taken as the mean of the distance of the points on the plane and the orientation is determined from the principal components of the set of inlier points. Let's say we have $\mathbf{m}$ inlier points on the plane, we stack the points to form a $\mathbf{m} \times \mathbf{3}$ matrix $\mathbf{A}$. The eigenvector of $\mathbf{A^T A}$ with the smallest eigenvalue would give the normal to the plane and the other two vectors represent the basis vectors lying on the plane. As the SLAM system progresses, newly found 3D feature points are checked if they lie in any of the best-fit planes using the same approach as that for inliers mentioned above. They use a creative technique to define fixed points on planes by checking if the variance of the 2D planar points becomes smaller than a certain threshold. These points are later used to update the

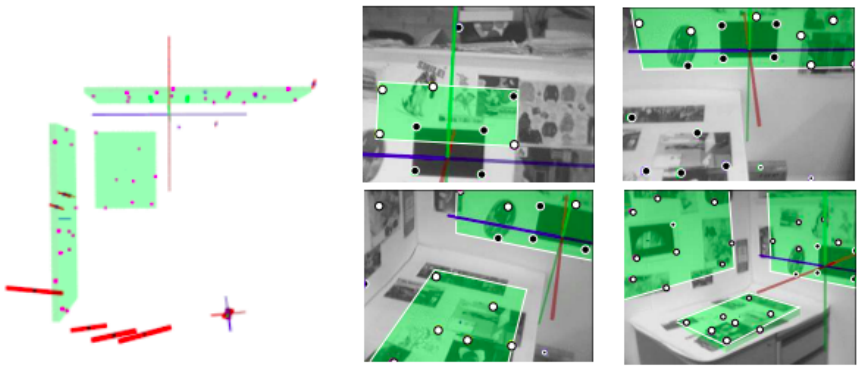


Figure 2.12: Real-time plane segmentation[16]: (Left) Camera Positions view (Right) Detected planar regions in the map. The points marked with black and white are converged 3D and 2D planar points respectively.

associated planar feature as the camera moves in the scene. Figure 2.12 shows the extracted planes in real-time along with the inliers of the consensus set.

One of the recent work by Zhang *et al.* [71] relies on intersecting lines to detect planes associated with these line features. It is a much more efficient method than detecting planes from point clusters. This method reduces the computational overhead of plane detection and allows the process to be performed in real-time on the CPU. Their approach is only experimented with in stereo SLAM for ease but can be expanded to a monocular setting with some modifications. They first detect 2D line segments in both left and right image frames using LSD [18] and then match them using LBD [69] descriptor. The 3D endpoints of the lines are calculated using stereo matching from the left and right images. They claim that planes extracted from parallel lines may have large errors so only lines that intersect are considered. The intersecting lines are found by using the following constraints: i) angle between the lines is larger than a threshold. ii) the distance between their midpoints is smaller than the length of line iii) the endpoints of both lines lie on the same plane. The normal of the plane is found by taking a cross product of the intersecting lines.

$$\vec{n} = l_i \times l_j \tag{2.11}$$

where $\mathbf{l_i}$ and $\mathbf{l_j}$ are direction vectors of the lines. The distance from the origin

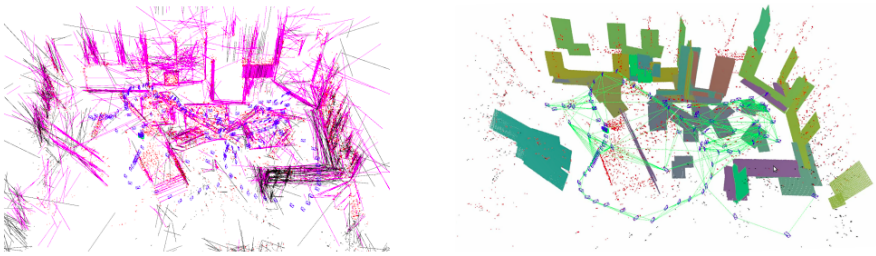


Figure 2.13: (Left) Map formed by 3D lines detected (Right) Map formed by 3D planes detected from intersecting lines

is taken as the arithmetic mean of the distances of endpoints from the plane. Hence, the plane coefficients are $\boldsymbol{\pi} = (\vec{\mathbf{n}}^T, \bar{\mathbf{d}_\mathbf{k}})^T$.

$$\bar{d_k} = \underset{d}{mean}(-\vec{n} \cdot (l_{xi/j}, l_{yi/j}, l_{zi/j})^T) \tag{2.12}$$

The error in planes is tracked in every frame and is used in the bundle adjustment to reduce the drift error in the SLAM system. However, using $\boldsymbol{\pi} = (\vec{\mathbf{n}}^T, \bar{\mathbf{d}_\mathbf{k}})^T$ as plane parameters causes over-parameterization which would require more computations during the optimization process. Hence, the plane is parameterized using the azimuth and elevation angles of the norm instead of the normal itself as $\boldsymbol{\tau} = (\boldsymbol{\phi}, \boldsymbol{\psi}, \mathbf{d})^T$.

$$\tau = q(\pi) = (\phi = \arctan \frac{n_y}{n_x}, \psi = \arcsin n_z, d)^T \tag{2.13}$$

The projection error of planes in the camera plane is hence defined as follows:

$$e(T_{cw}, \pi_w) = q(\pi_c) - q(T_{cw}^{-T}, \pi_w) \tag{2.14}$$

where $\mathbf{T_cw}$ is the transformation matrix from world to camera coordinate system, $\boldsymbol{\pi}_w$ and $\boldsymbol{\pi}_c$ are the plane coefficients in world and camera coordinate system respectively. Figure 2.13 shows the map formed by 3D line segments and the planes extracted from it.

This method produces a fine map of the planar landmarks but it comes with its drawbacks. The line segment detection method does not guarantee the 3D lines detected represent real structures. Hence, planes detected from these lines are susceptible to inaccuracies and would not reconstruct surfaces accurately. The coplanarity of lines is not considered thoroughly so the planes detected can

be inaccurate. This approach is also hard to extend to a monocular setting as line segment detection becomes computationally expensive. We address these problems in our proposed approach to get a smooth and accurate surface reconstruction from the planes.

2.5 3D Surface Reconstruction

The point cloud data recovered from SLAM systems or laser-based scanning devices cannot be used directly in a vision-guided robotic system or AR-VR applications. The point clouds are used to reconstruct meaningful surfaces to create a model of an object or a scene [5]. Unlike systems with laser scanners, SLAM systems don't produce point clouds of the complete scene instantly. SLAM systems produce streams of 3D points in an online system while exploring the scene from different views. The indirect SLAM approach reconstructs the feature points tracked over the keyframes while the indirect approach reconstructs the depth map producing more points. These methods keep reconstructing 3D points as the camera moves over the scene which requires the model to be updated over every keyframe incrementally. Hence, modeling the scene in an online system could be computationally expensive if all the points in the point cloud are used without filtering. Surface reconstruction methods are divided mainly into two categories based on the density of points used in reconstruction: Sparse (Explicit) and Dense methods (Implicit) [27], [33]. In the following subsections, we will briefly discuss these methods and the inspiration behind our proposed method.

2.5.1 Sparse Reconstruction methods

The sparse methods are used to reconstruct meaningful surfaces from a 3D point cloud which is sparse as it only consists of feature points in the scene. These methods help in reducing the space and time complexity for modeling large-scale environments. These methods use a 3D Delaunay triangulation to discretize the space into a set of tetrahedrons such that every point is a vertex of the tetrahedral. These points have information on the camera location

through rays connecting the point and the camera(viewing ray). Previous works classify the tetrahedron into free-space and occupied-space where the face that connects these two approximately represents the required surface [23], [34]–[36], [38]. The faces are in the form of triangles that are combined to form a surface mesh. Faugeras *et al.* [14] explains how 3D Delaunay triangulation is a computationally efficient method to represent geometric structures by marking empty tetrahedrons. It is robust as it is easier to update surfaces when new points are introduced in the system.

Lovi *et al.* [38] proposed an online 3D surface reconstruction method that efficiently works on sparse point cloud. They discretize the space via a 3D Delaunay Triangulation of the input point stream partitioning the convex hull of the point set into a set of tetrahedra. Based on visibility constraints the faces of these tetrahedrons are marked as empty or occupied. As it is an incremental process, some of the tetrahedra are deleted and replaced by new tetrahedra upon the addition of new points in the system. All the new sets of tetrahedra are associated with a set of all free-space constraints that intersect it. Hence, the old set of tetrahedra determines the minimal constraint set to be tested against. This saves a lot of time by avoiding triangulation on all points afresh as batch reconstruction approaches [33]. Reusing the tetrahedrons stabilize the model of the scene, and requires a smaller stream of input points to reconstruct and update surfaces in the scene. They also propose a heuristic to avoid new constraints if they have a high similarity score. The system works on PTAM [29] which makes it less efficient compared to modern SLAM systems. Their work was improved upon by developing a similar system upon ORB-SLAM2 [43] to incrementally reconstruct the surface on a CPU-only setting [25]. Figure 2.14 shows the reconstruction results of their approach.

Hoppe *et al.* [23] improved upon the labeling and visibility information using graph-cut on a dual graph. They mainly focus on incrementally reconstructing surfaces of pertinent objects using SFM systems akin to Labatut *et al.* [31] which reconstructs the scene using multi-view stereo. Unlike Labatut *et al.*[31], their method can operate in real time on 3D points from an online SFM. The edges of the triangulated tetrahedral are weighted based on an en-

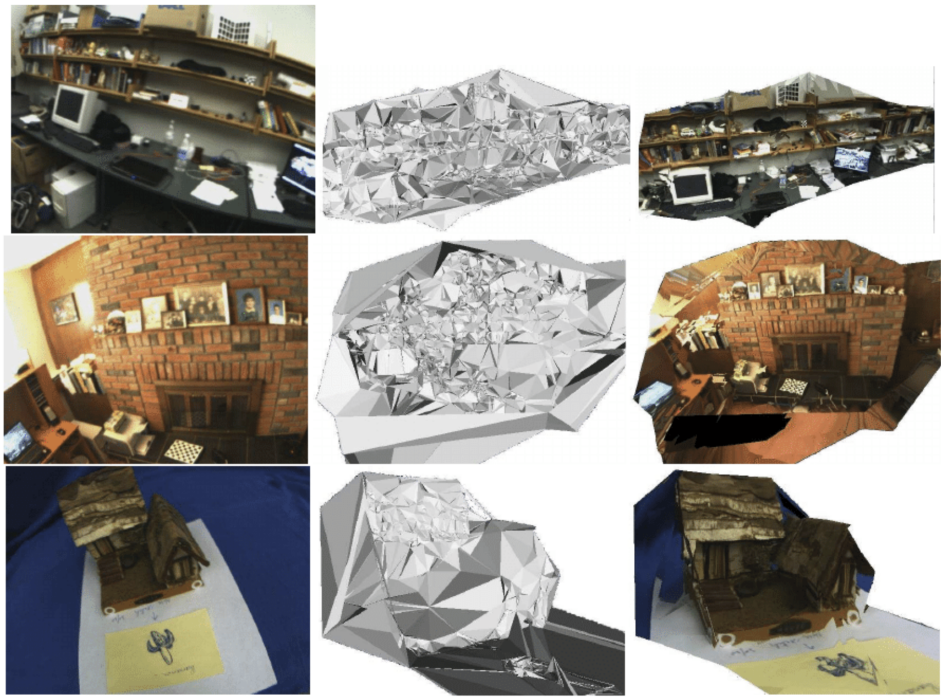


Figure 2.14: Scene reconstruction by [38] on three data sequences: Shelves, Fireplace, and House. (**Left**) Sample Image (**Middle**) Reconstructed Model (**Right**) Texture Mapping

ergy function that combines smoothness and visibility information. They use dynamic graph-cut to obtain the minimum cut efficiently. Dynamic graph cut [30] ensures that the minimum cut is performed as an update on the previous solution instead of recomputing the solution from the scratch. This makes sure that the computational complexity is independent of the overall scene size. Their work does not remove points with redundant visibility information and hence it is susceptible to outliers in viewing rays. Langlois *et al.* [32] consider a scene bounding box instead of the Delaunay Triangulation of points. These bounding boxes are partitioned into 3D cells which are marked as full or empty based on the visibility information of line segments. The intersection of these free and empty cells represents the reconstructed surface. They minimize an energy function that penalizes lines segments that do not lie on segmented surfaces, penalizing surfaces between the path of observation and visibility ray, and penalizing complex shapes. Their method adapts well to a sparse set of points and line segments.

2.5.2 Dense Reconstruction methods

Unlike sparse methods, dense reconstruction methods discretize the space into 3D voxels which is further converted into a surface mesh. This approach works best for dense depth maps to recover a volumetric representation of the scene. Roldão *et al.* [51]use a Truncated Signed Distance Function (TSDF) [63] voxel-based representation to reconstruct surface meshes from Lidar datasets. In TSDF, a 3D environment is represented by a 3D grid of equally spaced voxels where each voxel is defined by the position of its center. The signed density function is used to assign a positive or negative value to each voxel based on the distance between its center and nearest object surface. A weight is also assigned to the voxels to assess the uncertainty of these values. A positive value determines free space (in front of the object) and negative value determines occupied space (behind or inside the object). To avoid computational overhead, the signed density function is truncated for larger distances. Equation 2.16 defines the TSDF values for each voxel where SDF is the signed distance function and t is the truncation distance. For every voxel center v_i, $pic(v_i)$ is the projection of voxel center on to the depth image. $cam_z(v_i)$ is the distance between camera and voxel along the principal axis. This makes sure that points that are far away from an object do not interfere with the surface reconstruction.

$$SDF(v_i) = depth(pic(v_i)) - cam_z(v_i) \tag{2.15}$$

$$TSDF(v_i) = max(-1, min(1, \frac{SDF(v_i)}{t}) \tag{2.16}$$

Roldão *et al.*[51] propose multi-scale neighborhood definition to increase the statistical robustness of neighboring surfaces. The neighborhood of each voxel is represented by 8 adjacent voxels around it at a single level. By increasing the level, the size of the neighborhood increases by a factor of $(2k)^3$ where k defines the level. Since larger values of k will overly smooth the re-

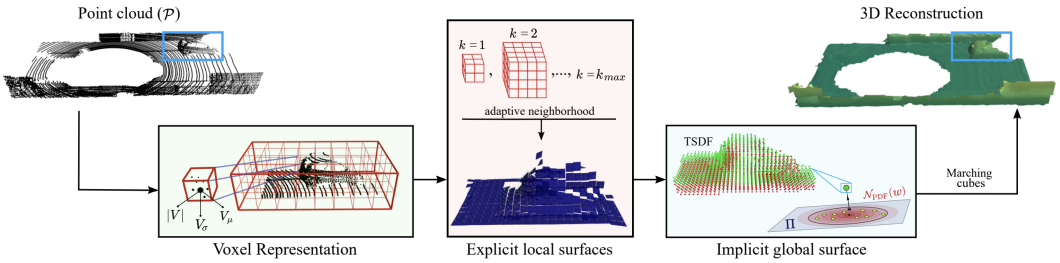


Figure 2.15: Overview of surface reconstruction by Roldão *et al.*[51].(1) The Lidar point clouds are represented by a voxel grid, (2) An optimal neighborhood level is estimated, (3) TSDF is performed on an optimal neighbourhood level, (4) Marching Cubes is used to reconstruct surface mesh

gions with a high density of points, an optimal k value is chosen to compute the optimal neighborhood for each vertex. Planar surfaces are estimated using PCA based on these neighborhoods. The optimal value of k is estimated using a multivariate Gaussian distribution to define the probability of each vertex projected on the estimated planar surface belonging to the distribution. The lowest k value that defines a normal distribution is used as the optimal neighborhood value. The TSDF values are estimated for the optimal neighborhood voxels. The TSDF is followed by Marching Cubes [37] to reconstruct surface meshes for objects. Figure 2.15 is a schematic representation of their surface reconstruction method.

Newcombe *et al.* [46] proposed a real-time SLAM system using Kinect (an RGB-D sensor) to reconstruct local surfaces using the depth maps generated by their infrared sensor. They used a TSDF-based approach which estimates a cumulative weighted signed distance field from the depth maps recorded by the sensor [9]. The camera pose is estimated using a multi-scale Iterative Closest Point (ICP) alignment between the depth maps from sensor input and predicted surface from raytraced global TSDF. Real-time performance was achieved by paralleling the TSDF module on a GPU. A similar approach was performed in a monocular setting in MonoFusion [48] which provides inspiring results from single off-the-shelf cameras. Figure 2.16 shows the experimental results of MonoFusion on nearby objects.

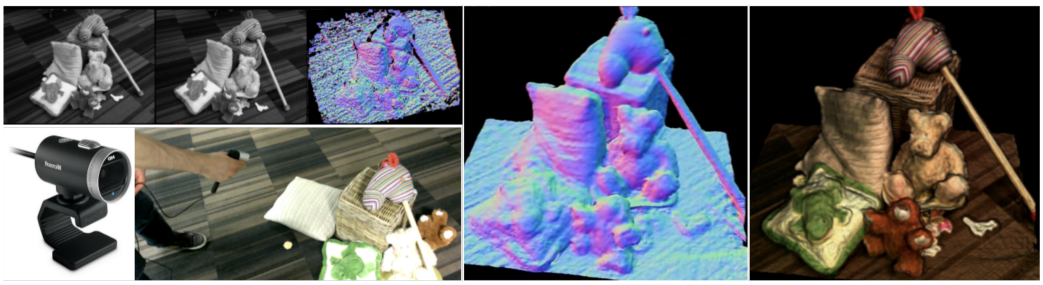


Figure 2.16: Experimental results of MonFusion[48] on nearby objects

However, these methods are concentrated at reconstructing local surfaces rather than modelling the complete scene. Voxel grid approaches heavily rely on the memory available on the GPU as each voxel is allocated a block of memory. These methods cannot be operated in real-time without the help of a strong GPU or an RGB-D sensor.

Chapter 3

Method

In this chapter, we discuss the pipeline of our system and detail our approach for surface reconstruction. We briefly describe our system followed by improvements in 3D Line Segment detection and Plane detection.

3.1 System Overview

Our incremental SLAM system takes live video frames as input and produces a 3D model. Figure 3.1 shows the pipelines of our proposed system. Our system comprises of a SLAM system with semi-dense mapping [42], a 3D line segment fitting approach, and a novel plane detection, validation, and merging approach.

1. **SLAM System**: This system takes video frames as input and gives a semi-dense point cloud of the scene.

2. **3D Line Segment Detection**: It computes lines in 3D space from the semi-dense point clouds incrementally.

3. **Plane Extraction**: It detects planes from 3D lines. The planes are validated, matched, and merged over keyframes.

4. **Surface Reconstruction** : This thread is referred to as CARV modelling in Figure 3.1. Once the planes are validated, the endpoints of the line segments in the planes are used to reconstruct a 3D model using a free-space Delaunay Triangulation-based approach [38].

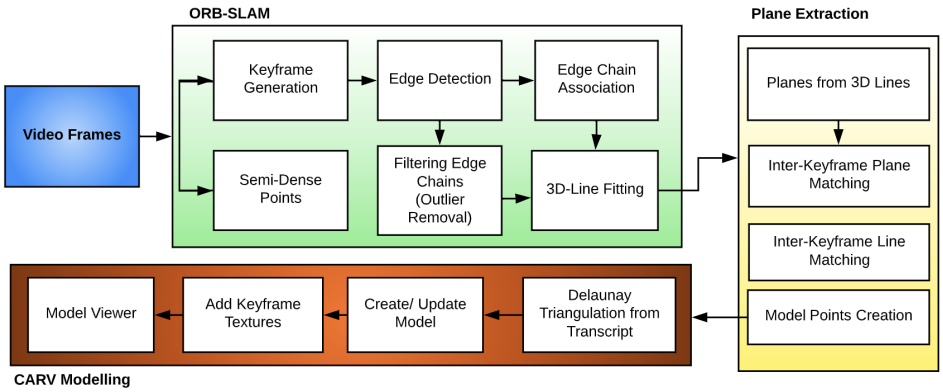


Figure 3.1: Pipeline of our Proposed System: (i) The video frames are passed through a semi-dense ORB-SLAM [42], (ii) 3D lines segments are extracted from the point cloud generated using our proposed Outlier Removal Method, (iii) Our proposed Plane Extraction method extracts planes, validates and merges them into major planar regions, (iv) Planar model points are fed to free-space-carving method by Lovi *et al.* [38] for surface reconstruction.

The entire system runs on 5 threads in real-time on a CPU. No GPU acceleration is required thanks to the efficient modeling using only points on planar structures.

3.2 3D Line Segment Detection

Detecting all possible lines from a point cloud is a computationally complex task with very minimal structural information. We limit the semi-dense mapping search space to high gradient edges to speed up the system and limit the search space for line detection. The lines detected from the edge maps will have fewer outliers with more structural information. We use a RANSAC-based approach to remove the outliers from the pixel chain before fitting the line. We use Edge Drawing [57] to detect edge maps of every keyframe. The edge maps are a chain of interconnected pixels we use to detect 2D line segments. Here we fit two 2D lines, one in the image plane and another in the plane orthogonal to the image plane. Unlike He *et al.* [21], we use a novel outlier removal method that filters the non-planar pixels before the line fitting. This outlier removal approach helps us remove depth inconsistency and detect lines that mostly lie on a major planar structure.

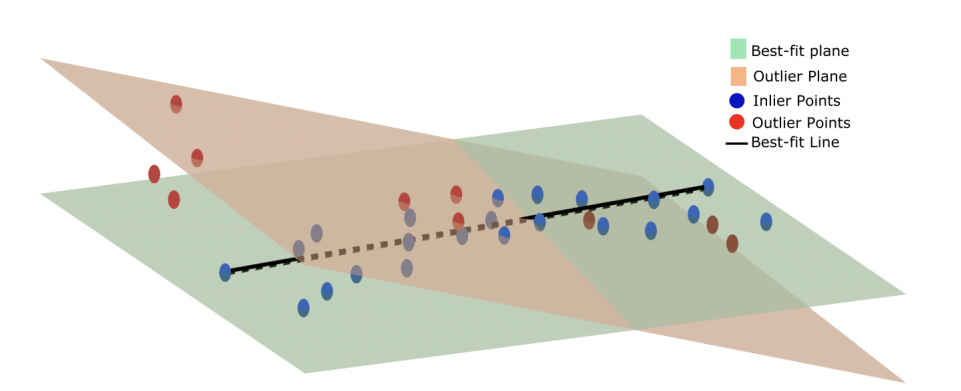


Figure 3.2: Removing outlier points from the pixel chain and fitting a 3D line segment using points from the best-fit plane (Inlier Set)

Let $\mathbf{P_c} = \{\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, ..., \mathbf{p_n}\}$ be the pixel chain with associated depth values where $\mathbf{n}$ is the minimum number of pixels required to fit a line. The coordinates of the image frame are represented by $(\mathbf{p_x}, \mathbf{p_y})$ and $(\mathbf{d}, \mathbf{p_z})$ in the frame orthogonal to the image frame. $\mathbf{d}$ is the distance from the first pixel to its projection in the x-axis, and $\mathbf{p_z}$ is the depth associated with the pixel. We perform our outlier removal method before fitting the 2D lines using Total Least Square. We further remove the pixels if the distance between the pixels and the two lines in the image and depth frame is greater than thresholds $\mathbf{d_1}$ and $\mathbf{d_2}$ respectively. The current line search is stopped and the new line is searched if $\mathbf{n}$ consecutive pixels are outliers.

3.2.1 Outlier Removal

We use a RANSAC-based approach to remove pixels that do not lie on the best-fit plane. $\mathbf{m}$ pixels are selected at random from the pixel chain to form a $\mathbf{A_{3\times m}}$ matrix. This matrix is decomposed using Singular Value Decomposition (USV^T) where the U vector corresponding to the smallest singular value is the normal $\vec{\mathbf{n}}$ of the best-fit plane. Singular Value Decomposition (SVD) is numerically more stable than finding an eigenvector corresponding to the smallest eigenvalue of $\mathbf{AA^T}$. **Proof:** Let $\mathbf{cov} = \frac{\mathbf{AA^T}}{\mathbf{n-1}}$ be the covariance matrix and $\mathbf{A} = \mathbf{USV^T}$ be the singular value decomposition.

Since the covariance matrix is symmetric, it can be diagonalized and the eigen-

vectors can be normalized to become orthonormal:

$$\frac{AA^T}{n-1} = \frac{XDX^T}{n-1} \tag{3.1}$$

Now, formulating the above equality using SVD

$$\begin{aligned} \frac{AA^T}{n-1} &= \frac{(USV^T)(USV^T)^T}{n-1} \\ &= \frac{(USV^T)(VSU^T)}{n-1} \end{aligned} \tag{3.2}$$

Since $\mathbf{S}$ is orthogonal, $\mathbf{V^T V = I}$

$$\frac{AA^T}{n-1} = \frac{US^2U^T}{n-1} \tag{3.3}$$

It can be seen that the formation of $\frac{\mathbf{AA^T}}{\mathbf{n-1}}$ will have a loss of precision whereas SVD does not have that issue and is numerically more stable.

This process is repeated multiple times (3 times in our case) to get the most robust plane. An inlier set $\mathbf{I_s}$ is formed from the pixel chains by checking whether the perpendicular distance of the point from the best-fit plane ($\vec{n} \cdot P_c$) is less than a threshold (2 cm). Only inlier pixels are used to fit the 2D lines using the Total Least Squares, ensuring that the 3D lines detected lie on a real planar structure. Figure 3.2 shows the removal of the outlier points that do not lie on the best-fit plane followed by line fitting on the inlier set resulting in a 3D line segment that lies on the best-fit plane.

Our method drastically reduces the number of lines detected in every keyframe. While some methods require large number of lines or points to reconstruct surfaces, detecting planar structures benefits from fewer lines as explained in the Section 3.3. Figure 3.3 compares our outlier removal approach against He *et al.* [21]. Table 4.1 shows that our outlier removal method filters large number of lines making it easier to detect planar structures accurately.

3.3 Plane Detection

Detecting planes from point clouds is a computationally expensive task and hard to implement in a real-time system. Therefore, we reduce our search space by detecting planes from intersecting 3D line segments detected from

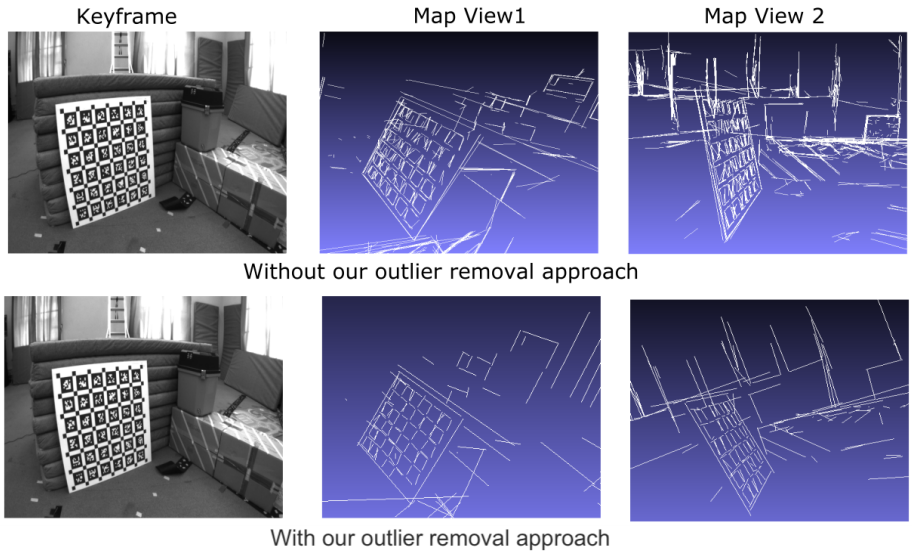


Figure 3.3: **Keyframe**:Vicon Room 101 [6] **(Top)**: 3D line fitting without Outlier Removal **(Bottom)**: Proposed outlier removal method

a semi-dense point cloud. We exclude planes from parallel lines as they may create ambiguities in man-made structures (e.g. windows on walls). Two vertical or horizontal lines can be detected on a surface but they may not represent an actual physical structure. We perform an exhaustive search of 3D planes from intersecting lines. This search is constrained by the following conditions:

(a) The angle between the lines must be greater than a threshold (15°)

(b) The lines must be at least *15 cm* long

(c) The lines must not be associated with more than 3 planes.

(d) The lines must not be skewed, hence concurrency must be confirmed

This exhaustive search detects a large number of plane hypotheses $\boldsymbol{\pi}_k = (\vec{\mathbf{n}}, \mathbf{d})^{\mathbf{T}}$, most of which are either redundant or lie within the vicinity of an existing major planar structure. Let $\mathbf{p_x}$ and $\mathbf{p_e}$ be the endpoints of the line segment then the direction vector can be defined as a unit vector of their difference. The normals of the planes are evaluated by taking the cross-product

of their direction vectors.

$$\begin{aligned} dir &= | < p_{sx}, p_{sy}, p_{sz} > - < p_{ex}, p_{ey}, p_{ez} > | \\ \vec{n} &= dir_i \times dir_j \end{aligned} \tag{3.4}$$

The plane parameters can also be computed using the points on the lines by SVD as discussed in Section 3.2.1. With the computed normal vector, the **d** value can be defined as,

$$d = -\vec{n} \cdot < p_x, p_y, p_z > \tag{3.5}$$

where $< \mathbf{p_x}, \mathbf{p_y}, \mathbf{p_z} >$ are the points on the line. We use the endpoints of the line to define the **d** value. We validate each plane hypothesis in the process called merging and growing, which is constrained by the angle between the normals and the distance between the planes. This process also involves filtering the excess lines being matched in every keyframe, thus reducing the complexity of the process. Algorithm 1 describes the validation, merging, and growing process of our proposed system.

Three Point Method: Let $\boldsymbol{\pi} = (\vec{\mathbf{n}}, \mathbf{d})^{\mathbf{T}}$ define the plane parameters and $\mathbf{X_i}$ be the homogeneous coordinates of points on the plane.

$$\begin{bmatrix} X_1^T \\ X_2^T \\ X_3^T \end{bmatrix} \pi = 0 \tag{3.6}$$

The null space gives the parameters of the plane

$$\pi = (D_{234}, -D_{134}, D_{124}, -D_{123})^T \tag{3.7}$$

where $\mathbf{D_{ijk}}$ is the determinant of the **ijk** rows of the 3×4 matrix $\mathbf{X}$. For any three points on the non-collinear lines, the plane equation can be derived by:

$$\vec{n} = (\tilde{X_1} - \tilde{X_3}) \times (\tilde{X_2} - \tilde{X_3}) \tag{3.8}$$

$$d = (-\tilde{X_3}^T (\tilde{X_1} \times \tilde{X_2})) \tag{3.9}$$

Notations: We define $\mathcal{S}_p = [l_i, l_j]$ as a list of line pairs that define the planes, $\mathcal{P} = [\vec{n}, d]$ is the list of planes corresponding to the line pairs in S_p. We group the validated planes in $\mathcal{V}_p = [\vec{n}, d]$ and the lines are added to a

Algorithm 1 Plane Validation: Merging and Growing

Input $\mathcal{S}_p = [l_i, l_j], \mathcal{P} = [\vec{n}, d]$
Output $\mathcal{V}_p = [\vec{n}, d]$, $\mathcal{CS} = \{[l_i, l_j] : \vec{n}\}$

procedure VALIDATION,MERGING AND GROWING
  // Remove Redundant planes
  for $[\vec{n}_i, d]$ *in* $\mathcal{P}$: **do**
    $temp \leftarrow [\vec{n}_i, d]$
    if $|\vec{n}_i.\vec{n}_j| > \lambda_p$ *and* $|d_i - d_j| < \psi_p$ **then**
      $\mathcal{V}_p \leftarrow temp$
      $\mathcal{P}.pop()$
  // Form Consensus Set
  // Let dir_i:-Direction vector & Pts:-Endpoints
  for $[l_i, l_j]$ *in* $\mathcal{S}_p$: **do**
    for $\vec{n}$ *in* $\mathcal{V}_p$: **do**
      if $|dir.\vec{n}| < \lambda_l$ *and* $|Pts.\vec{n}| < \psi_l$ **then**
        $\mathcal{CS} \leftarrow \{[l_i, l_j] : \vec{n}\}$
        $\mathcal{S}_p.pop()$
      else *continue*
  // Validation
  for $\vec{n}$ *in* $\mathcal{V}_p$ **do**
    if $Count(\mathcal{CS}[\vec{n}]) > ValidationThresh$: **then**
      $\mathcal{V}_p \rightarrow$ **VALIDATED**
  //Merging in InterKeyFrame Matching

consensus set $\mathcal{CS} = \{[l_i, l_j] : \vec{n}\}$. We also introduce λ and ψ as the angle and distance thresholds with p and l subscripts defining plane and line thresholds respectively.

Firstly, we remove all the planes with similar normals and distances that are usually detected by multiple lines near the major structure. The cosine of the angle between the plane normals can be calculated by $|\vec{n}_i \cdot \vec{n}_j|$ and the distance between them is determined by $|d_i - d_j|$. This is an important step before matching of planes in subsequent keyframes as it avoids ambiguity while matching planes. Once the redundant planes are filtered, we form a consensus set of the lines that lie in the detected planes. The cosine of the angle between the plane normal and the line is smaller than λ_l and the distance between the endpoints of the line and the plane is smaller than ψ_l, the line is added to the consensus set $\mathcal{CS}$. The consensus set also maps every plane with the

lines associated with it. If a plane is associated with a number of lines greater than the validation threshold, the plane is deemed valid and is considered for InterKeyframe matching. The plane matching is followed by merging and growing planes into major planar structures as we discover new areas or parts of existing planar regions in new keyframes. However, the length of planar structures is constrained to avoid the seamless merging of planes into larger planar regions (e.g., wall, ceiling, floor).

3.4 InterKeyframe Plane Matching

Once planes are detected in a keyframe they must be tracked over keyframes for consistency. This ensures that the planar structures discovered are mapped and modeled accurately in an incremental way. The plane matching is performed similarly to the filtering of the redundant planes but over keyframes instead of planes in a single keyframe. Once the planes are matched, we also match the line segments associated with the planes. The lines are matched using the angle between them and the distance between their centers.

$$\cos\beta = \frac{\vec{dir}_i \cdot \vec{dir}_j}{|\vec{dir}_i \cdot \vec{dir}_j|_2} \tag{3.10}$$

$$D = ||mid(l_i) - mid(l_j)||_2 \tag{3.11}$$

If the angle between the lines (β) and the distance (D) between them does not exceed the line matching threshold ($\beta = 2^\circ$ and $D = 0.02$), the lines are flagged as matched. The matched planes must contain a minimum number of lines matched lines to be considered valid for modeling.

We introduce the merging and growing of planes to find major planar structures in the scene. We also group the lines of merged planes into a single group. This drastically reduces the number of planes in the scene and the search space in every new keyframe. During the plane matching, planes with their normals inclined at an angle less than the merging threshold (10°) and the distance between them is smaller than a distance threshold($25\ cm$) are merged into a single plane. The planes and lines associated with them are merged into a single group with a common normal of the plane with a larger

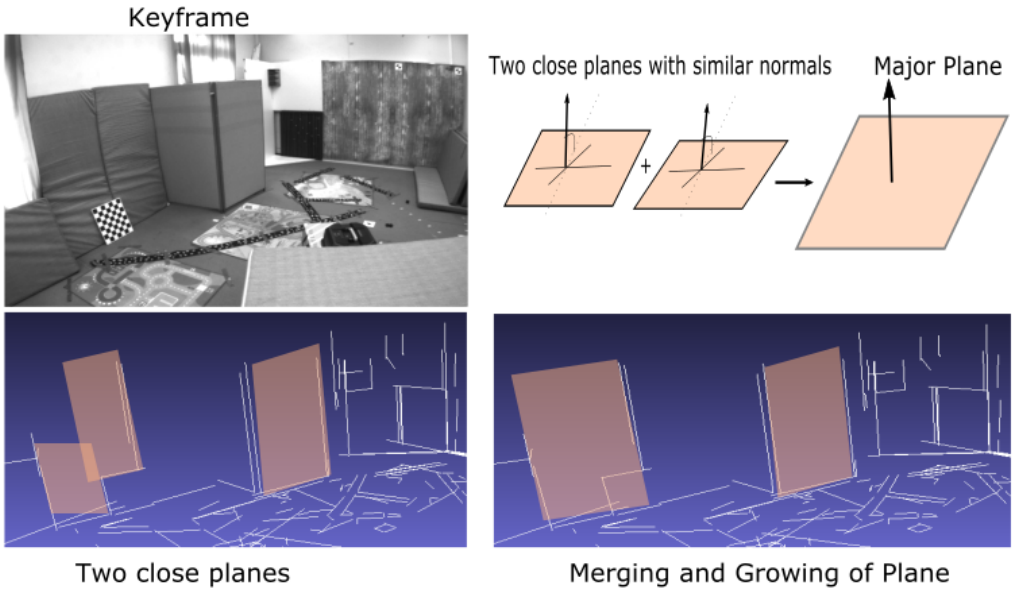


Figure 3.4: **Top:** Two planes with normals inclined at a near-similar angle under a distance threshold are merged into a single plane bound by distance thresholds of lines defining them; **Bottom:** A sample example of two planes combining to represent the cushions in the keyframe.

number of lines associated with it. Lines associated with more than 2 planes are discarded. The $\tilde{d}$ value is updated to be the mean of d values of both planes.

$$\tilde{d} = mean(d_i, d_j) \tag{3.12}$$

We also constrain the growth of the major planes by the number of lines associated with them. The planes associated with more than 30 lines are not considered for the merging and growing process. This helps prevent planar structures from merging into much larger planes like walls, ceilings, and floors. Figure 3.4 gives a brief idea of the merging and growing process. It is worth noting that the merging and growing parameters can be tuned based on indoor/outdoor scenes for better surface reconstruction. The merging thresholds must be noticeably higher for outdoor scenes.

3.5 Surface Reconstruction

Once we obtain the valid planes of the keyframe, we use the 3D lines in these planes to incrementally reconstruct the surfaces of the scene. We use a

lightweight pipeline of surface reconstruction based on 3D Delaunay Triangulation [38]. We choose this method as it is compact and faster than voxel-based approaches discussed in the literature. It also has a robust outlier removal method to remove viewing rays of outlier points in the model. The creation and update of the model are fast and robust upon the insertion of new points.

Space is discretized into tetrahedrons such that each new 3D point passed to the system is included by inserting a vertex and adjusting the neighboring tetrahedrons. For each frame, visibility information is used to carve out (remove) all tetrahedrons where a viewing ray is connecting a visible 3D feature point to the camera center. We use an improved version of this system [21] by applying a graph-cut-based approach to categorize the tetrahedral. This also takes into account the smoothness of the surface to reduce the effects of an outlier viewing ray.

Point and line-based reconstruction methods use a large number of points to reconstruct surfaces. In 3D line-based reconstruction method [20], the results from clustered 3D line segments are worse than the unclustered lines. One potential justification for such an anomaly is that fewer points have insufficient information, however, closer inspection suggests that the clusting method is a larger contributor. The lines are clustered based on a nearest-neighbor approach which results in spurious line segments distorting the surface. Our experimental results demonstrate that better results can be achieved with fewer lines and points passed to the system. Our plane detection method provides the surface reconstruction system with fewer, but accurate points with planar information, resulting in a more accurate representation of surfaces. We use this method to build upon ORB-SLAM for surface reconstruction [25]. End points of the planar lines are fed to this system to compute the scene model. Our system is modular and can easily be adapted to a newer real-time SLAM system in the future.

We will further briefly discuss the components of our surface reconstruction system. There are mainly two components of our reconstruction system:

(a) 3D Delaunay Triangulation with Viewing rays

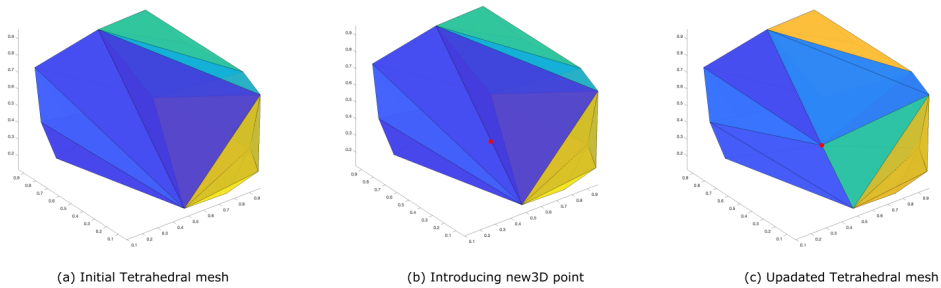


Figure 3.5: Tetrahedral mesh update with 3D Delaunay Triangulation (a) Initial tetrahedral mesh (b) A new vertex in Red is added to the the mesh for mesh update (c) The updated mesh shows that the surfaces near the new vertex are only updated leaving the rest of the model unchanged

(b) Identifying Free Space and Occupied Space from Tetrahedrons

3.5.1 3D Delaunay Triangulation and Viewing rays

Once points are passed to the system, 3D points are triangulated to form a tetrahedral mesh. Using 3D Delaunay triangulation, the new entries can be inserted locally and incrementally updating the current mesh. Figure 3.5 shows how the mesh can be incrementally updated upon new entries without updating the whole model. This results in a fast and efficient update of the model.

Every new 3D point we add to the mesh has visibility information with it. We can connect the camera position of every keyframe to its corresponding 3D point in the mesh which is termed as the viewing ray. These rays help determine free-space and occupied space in the tetrahedrons. All the tetrahedrons intersecting these rays are updated with ray counts after every new entry.

3.5.2 Free Space and Occupied Space

The free space and the occupied space are efficiently identified using a minimum graph cut problem. Two terminal nodes are added to the vertices of tetrahedrons known as the source and the sink. If the edges connecting the source and the vertex are cut then the associated tetrahedron is labeled as occupied space or free space if the edge connecting is cut. Here we categorize the

edges as: the source and the sink edge, based on their connection to the source or the sink. These edges can be grouped together as visibility edges. We use a regularized smoothing factor to account for outliers. The edges connecting the adjacent tetrahedron are called smoothing edges. The total cut weight is minimized by the following least square solution:

$$W_{total} = \sum_{E_{vis}} w_{vis} + \lambda_{smooth} * \sum_{E_{smooth}} w_{smooth} \tag{3.13}$$

where λ_{smooth} is a scaling factor to balance the weights, w_{vis} is the weight for visibility, and w_{smooth} handles the smoothness between adjacent tetrahedrons. The weights are positive and non-zero volumes of tetrahedrons when an edge is connected to either the source or sink. Hence, the edges with the opposite label are cut. w_{smooth} is calculated for all the edges with shared triangles of two adjacent tetrahedral. The edges provide the labels(free space/ occupied space) for the tetrahedron once W_{total} is minimized.

Chapter 4

Experiments and Analysis

In this chapter, we discuss the qualitative and quantitative results achieved by our method on challenging datasets. We evaluate our system on the benchmarks EuRoC MAV Datasets [6] including Vicon Room 101(VR101), Vicon Room 201(VR201), and Machine Room 101 sequences. The scenes in VR101 and VR201 are challenging and poorly textured, thus common texture-based stereo does not work. Point-based free-space methods [34], [35], [38] struggle to align the model with the physical scene structure. The planes extracted from our method help the model match scene planes.

4.1 Implementation

All experiments were performed on a laptop with Intel i7-9700k Quad-core CPU with 16 GB of RAM. We built our system on sparse/semi-dense ORB-SLAM [42] with a lightweight Delaunay Triangulation-based reconstruction pipeline [38]. Our system reconstructs surfaces in real-time over 6 threads. Our system is tested on the latest libraries (e.g., OpenCV, CGAL) and ROS versions. Parameters for our line segment extraction method are $\epsilon = 2cm$, *minPixels = 25*, and *minInlierSet = 18*. Our plane fitting and matching parameters are $\lambda_p = \cos(5^\circ)$, $\lambda_l = \cos(85^\circ)$, $\psi_p = 2cm$, $\psi_l = 0.02cm$, *Validation Threshold = 18 lines*. The line matching parameters are $\beta = 2^\circ$ and *D = 0.02 cm*. The plane merging thresholds are 10° and *25 cm* for angle and distance respectively.

4.2 Quantitative Analysis

We compare metric 3D reconstruction errors for our method, line, and point-based methods using the 3D scanned point cloud provided by the benchmarks. For the comparison, we manually register the SLAM coordinates with the 3D scans. There are very few works and codes available for assessing the accuracy of 3D reconstructed models. We follow the metrics used by [32] to evaluate the closeness of reconstructed models with the ground truth point cloud. Since the ground truth point cloud has a different coordinate than our SLAM-based system, we calculate the global Euclidean Transform and transform the point clouds to the same scale by performing Iterative Closest Point(ICP) search. This way we can align both clouds together to compare them. All vertices of the mesh are transformed and scaled to the same Euclidean transform before evaluation.

Completeness is the measure of the distance from the ground truth (GT) points to the 3D model and counts the percentage of ground truth points that is within *25 mm* of the model. This metric tells us how close the ground truth points are to the reconstructed model. To evaluate this, we traverse through all the ground truth points and calculate the orthogonal distance to the nearest triangle of the mesh using barycentric coordinates. Note that Visual SLAM provides much fewer 3D points than the scanned ground truth points. Hence completeness also measures how well the surface reconstruction generalizes across featureless surfaces. Doing this fill-in well as a motivation for developing our plane-based method. Walls, doors, and sides of man-made objects often lack visual features, and need to be filled in.

Precision is the measure of the distance from each vertex in the computed triangulation to the ground truth scan. We report the fraction of points within *25 mm* of the ground truth. Precision measures how good a method is at choosing accurate 3D model vertices for triangulation, but not how well the model generalizes over the whole scene. These two metrics go hand in hand as completeness helps to prove that the precise points can reconstruct a surface that generalizes well also where there are no SLAM features. We use the

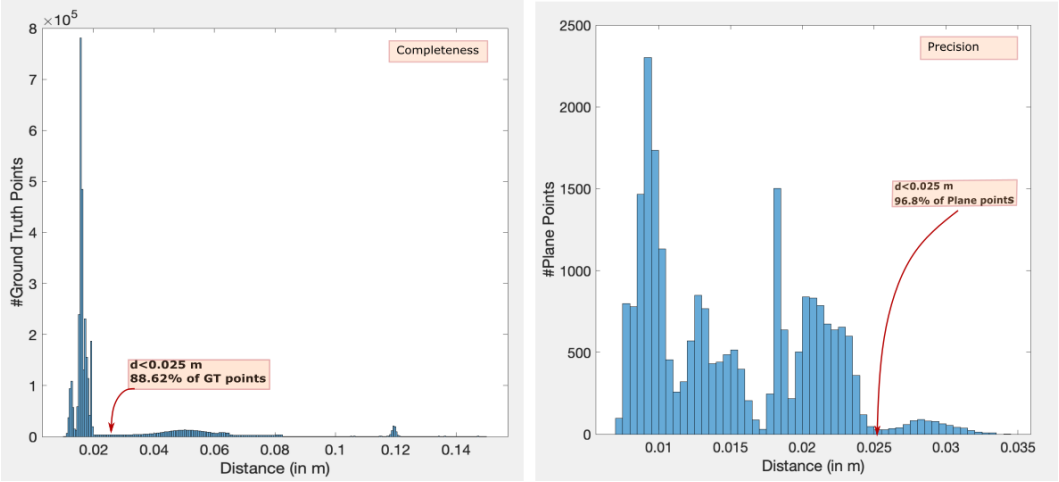


Figure 4.1: Histogram of distance errors(d) from Ground Truth for *VR101* sequence

Ground Truth 3D point cloud provided by the benchmark to perform our calculations. the VR101 and VR201 provide 3.1 million ground truth points. In Figure 4.1 and Figure 4.2, d refers to the distance from the ground truth and the arrows point to the $25mm$ mark of d value. It shows that around 89% and 86% of the ground truth points are closer than *25 mm* to our reconstructed model for VR101 and VR201 sequences respectively. The precision of our model shows that about 96% and 93% of our points used for reconstruction are closer than *25 mm* to the ground truth for VR101 and VR201 sequences respectively.

Table 4.1 compares our model with the point and line-based models based on these metrics. It can be seen that the Precision of the model increases on moving from points to lines but the Completeness is comparatively similar for point and lines based methods. This means that the model reconstructed by the point and line methods does not correspond well with the ground truth. Higher precision in the line-based approach means that the endpoints of most of the lines used as vertices of the reconstructed mesh are closer to the ground truth than using just points. For sequence VR101, Precision increases from 74.5% to 88.33% for line based method. It further increases to 96.8% once we filter all the lines that are not associated with real planar structures. The completeness for VR101 only changes from 70.98% to 77.21% from points to

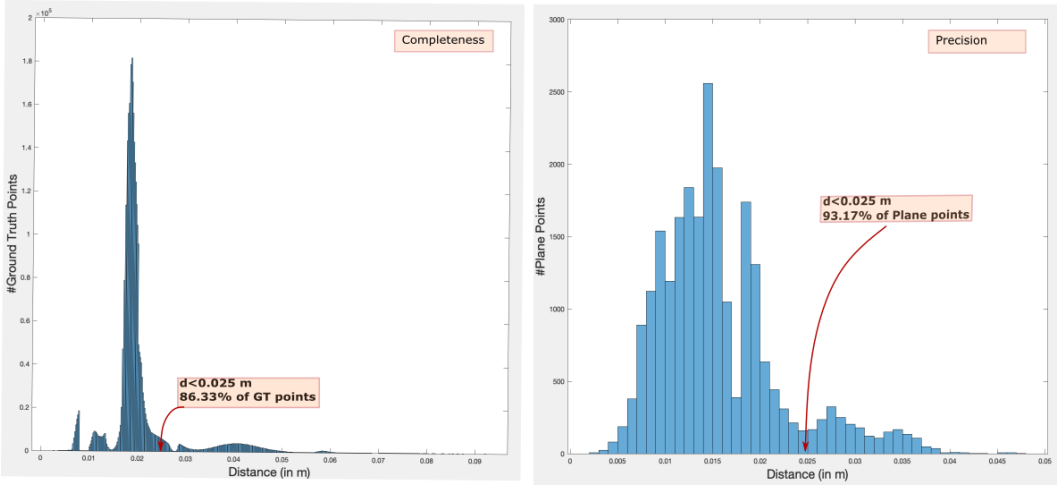


Figure 4.2: Histogram of distance errors(d) from Ground Truth for *VR201* sequence

Method	Lines	Vertices	Prec[VR101]	Comp[VR101]	Prec[VR201]	Comp[VR201]
He *et al.*[21]	15693	31386	88.33%	77.21%	72.93%	69.00%
Hofer *et al.*[22]	13278	26556	79.17%	73.08%	71.55%	65.93%
Lovi *et al.*[38]	**0**	72588	74.5%	70.98%	69.48%	68.21%
Ours(Lines)	9951	19902	91.41%	82.57%	78.98%	75.10%
Ours(Lines + Planes)	5843	**13617**	**96.8%**	**88.62%**	**93.17%**	**86.33%**

Table 4.1: Quantitative Results: Average number of Lines and Vertices for other datasets are similar. (**Prec:** Precision, **Comp:** Completeness)

line-based method which means the model correlates less with the ground truth for both line and point-based methods. The outliers in the line-based approach affect the model drastically which widens the gap between precision and completeness. We address these issues in our plane-based approach making our model closer to the ground truth for both sequences. It is worth noting that our model uses a significantly lower number of lines and points to reconstruct a 3D model that more closely fits the physical model than the others.

Table 4.2 shows the average number of lines, planes, and vertices used by our method for surface reconstruction. The Machine Hall sequence has a drastically lesser number of planes as the scene does not have many planar objects. This is a challenging scenario for our method but Figure 4.3 shows that our method still outperforms the line and point-based methods in terms of the quality of reconstruction. The number of planes detected for Vicon

Sequence	Lines	Planes	Vertices
Vicon Room 101	9951	174	13617
Vicon Room 201	6624	109	11387
Machine Hall 01	8052	93	16588

Table 4.2: Average number of Lines, Planes, and Vertices for Sequences VR101,VR201, and MAH01

Method	Vicon Room 101	Vicon Room 201	Machine Hall 01
He *et al.*[21]	31386	28996	36044
Hofer *et al.*[22]	26556	17411	10387
Lovi *et al.*[38]	72588	42097	68396
Ours(Lines)	19902	22568	27240
Ours(Lines+Planes)	13617	11387	16588

Table 4.3: Comparison of the average number of vertices in the reconstructed mesh for VR101,VR201, and MAH01

Room sequence is higher as there are more planar objects in the scenes hence the reconstruction results are also more accurate. Table 4.3 shows the average number of vertices in the reconstructed mesh by point, lines and plane-based method. It can be seen that the fluctuation in the number of vertices for different sequences is too high for [22] as the number of 3D lines detected in every run varies by a big margin. This makes this method very unreliable for 3D line segment extraction in real time. The Completeness and Precision percentages in Table 4.1 further prove the unreliability of Hofer *et al.*[22]. The number of vertices used for reconstructing the mesh in our method is fewer than the other approaches(except for [22] but it has worse reconstruction results) making the modeling faster.

4.3 Qualitative Analysis

We compare our system against point-based and line-based reconstruction methods on challenging datasets. For point-based approaches we compare our system against Lovi *et al.*[38] as it is lightweight and one of the only real-time systems available to public. We also compare our method against line-based methods like Hofer *et al.*[22] and He *et al.*[21] but these methods do not guarantee a real-time solution. We also perform an ablation study of our system

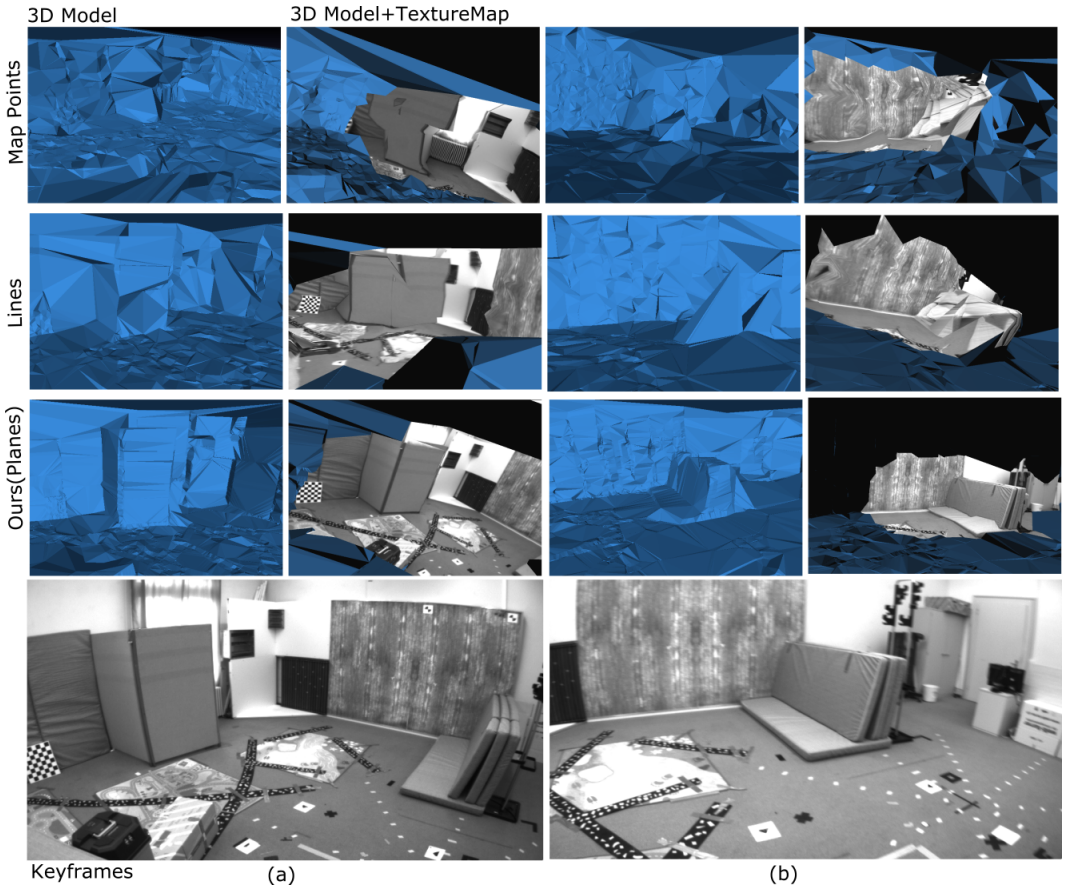


Figure 4.3: Qualitative Results on *EuRoc MAV VR101* with Model and Texture mapping

with only lines and then lines + planes. Tables 4.1 and 4.3 show the quantitative results of our ablation study. Our 3D model aligns well with the physical scene surfaces and also improves texture mapping. Lines play an important role in the detection of planes. Line3D++ [22] uses geometric line matching to reconstruct 3D lines from 2D lines using only geometric constraints of the line. This leads to ambiguities and an inconsistent number of lines generated over keyframes. It is quite unreliable as it reconstructs very few lines in complex scenes like Machine Hall 01 sequence [6]. On the contrary, He *et al.*[21] relies heavily on the accuracy of the semi-dense point clouds. This approach reconstructs 3D lines with depth inconsistencies which causes some lines on planar structures to deviate out of the planes. Their clustering method only provides an approximate idea of an object or structure in the scene as it does not match

the lines over keyframes. Figure 3.3 shows the lines that are supposed to be on the same plane having depth inconsistencies. Our novel outlier removal method ensures that the 3D lines detected lie on an actual planar structure. This filters the outliers and makes the texture mapping more accurate. Table 4.1 depicts the higher Precision of our approach proving that the lines used by our method actually align with real physical structures. The accurate texture mapping by our approach can be observed in Figure 4.3. The vertices are mapped accurately by our method hence the texture distorts less on camera motion. Line and point based methods fail to align the vertices accurately to the texture image leading to severe distortions upon camera motion.

Point-based approaches [23], [35], [38] use the SLAM sparse point cloud to form tetrahedral volumes using Delaunay triangulation. The sparse point clouds produced by the SLAM system can be prone to error, which can create holes or triangulate with model edges not matching scene edges. Furthermore, the larger model size (many triangles) is harder to parse semantically into structures than a small model succinctly representing the scene. Model misalignment and surface normals deviating from the scene surfaces distort the texture mapping. It can be observed that the line-based model is superior to the point-based model but has holes and distorted structures due to depth inconsistency. The detection and merging of planes into major planes aid the accurate and undistorted reconstruction of physical structures. To demonstrate the better generalization of our method, we compare our method over three challenging sequences of data. Figure 4.4 shows the qualitative superiority of our models in complex scenes. In VR101 sequence, it can be observed that [38] fails to reconstruct the stack of cushions and the boxes around it given the textured board of April tags. This shows the sensitivity of the 3D Delaunay Triangular method to outlier points around accurately reconstructed 3D points. The results seem to improve when these outlier points are filtered using the line segment approach [21]. The lines-based model seems to reconstruct the stack of cushions well around the board of April tags but fails to align the model with the physical structure. This is large because the lines fed to the surface reconstruction model have less structural information

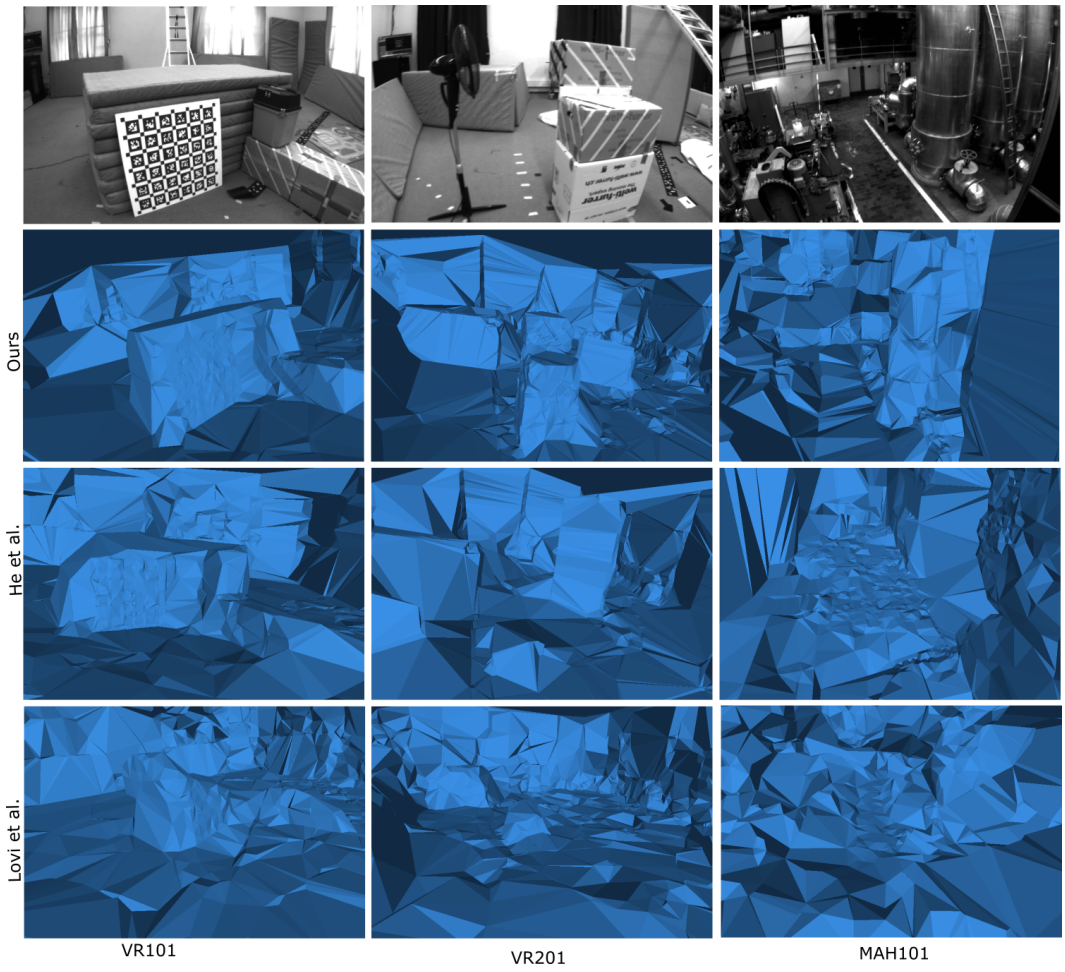


Figure 4.4: Experimental results on benchmarks *EuRoC MAV Vicon Room 101, Vicon Room 201, and Machine Hall 01*

due to which the reconstructed surfaces are not aligned properly. We address this issue by grouping the lines into planar structures and filtering the lines which do not lie on the planar structure. This highly reduces the outliers around the planar physical structures which would distort the shape of the object during 3D triangulation. It can be observed that our model was able to reconstruct the stack of cushions and the boxes around it with more accurate shape alignments. We further test our method on the more complex and cluttered scenarios in VR201(**Middle** - Figure 4.4). It can be observed that the point and line-based models fail to reconstruct the boxes or the cushions around them whereas our model is able to reconstruct most of the meaningful structures from the clutter. During the modeling process, we parse through

the planes reconstructing the scene plane by plane that is arranged close to one another during the merging and growing process. It is worth noting that the surfaces in line and point-based models seamlessly merge into larger planar regions(e.g., wall, floor, ceiling). We further test our method on Machine Hall 01 sequence (**Right** - Figure 4.4) which consists of mostly non-planar objects like pipes. It is a challenging dataset as our method heavily relies on planar structures to reconstruct surfaces. It is worth noting that our model still outperforms the point and line-based methods by reconstructing the pipes into rectangular columns while the other approaches fill in the empty volume around the pipes into a single planar surface(like a wall). Figures 4.5 and 4.6 show the texture mapping of the models on Machine Hall 01 and Vicon Room 201 sequence respectively. It can be observed that our model has the most accurate and undistorted texture mapping of the three models.

We further used our system to run in a real-time setup organized in our lab. Figure 4.7 shows a textured 3D reconstruction of our setup from different viewpoints. We choose this setup to motivate applications of our work in assistive robotics and predictive display for simple robotic manipulation tasks(e.g., pick-up and place tasks).

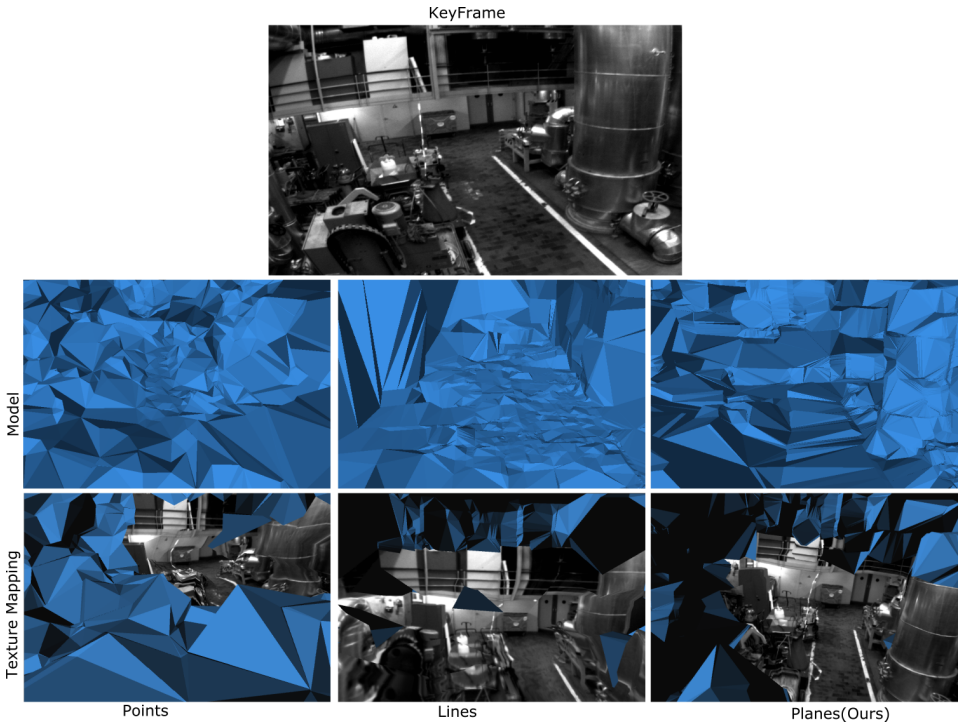


Figure 4.5: Qualitative Analysis: Texture Mapping on *Machine Hall 01* sequence

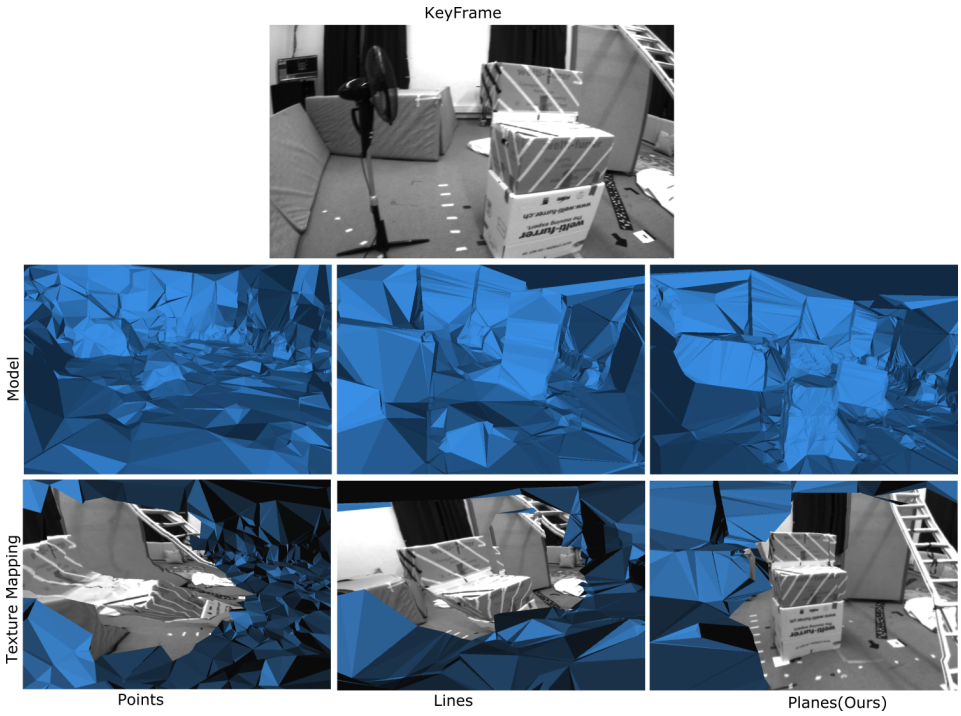


Figure 4.6: Qualitative Analysis: Texture Mapping on *Vicon Room 201* sequence

Figure 4.7: Textured Reconstruction of a 3D scene in a real-time setup. Rows 1 and 3 are the reference keyframes and Rows 2 and 4 show the reconstructed model from four different viewpoints

4.4 Runtime Complexity

To deduce the runtime complexity of our system, we analyze the runtime of line segment extraction, plane extraction and validation, and surface reconstruction. The line segment extraction method is mostly linear in the number of pixels in the pixel chains detected by the edge detector[57]. Our RANSAC-based outlier detection method makes it a bit slower than He *et al.*[21] but is still fast enough to operate in real-time. We avoid their clustering algorithm which is $O(N^2)$ in complexity for detecting N number of line segments. Their clustering algorithm is slow and also gives inaccurate 3D line segments for reconstruction. Our line segment extraction method reduces the number of detected drastically which helps make the plane extraction and surface reconstruction faster. The worst-case complexity of our plane extraction method is $O(N^2)$ where N is the number of lines in every keyframe. Practically, it is faster as the search space decreases when the planes are either merged or invalidated. Table 4.4 shows the average time taken per keyframe for line segment extraction, plane extraction, and validation over three data sequences. Figure

Sequence	Lines	Planes	Validation
Vicon Room 101	8.53	1.18	0.003
Vicon Room 201	8.14	1.02	0.002
Machine Hall 01	8.00	1.02	0.002

Table 4.4: Average Time (*in milliseconds*) for Line segment extraction, Plane extraction, and Plane Validation for Sequences VR101,VR201, and MAH01

4.8 shows the average runtime of the surface reconstruction model by points, lines, and plane-based methods over the VR101 data sequence. It must be noted that the timestamps were recorded after every model update. Since the modeling thread runs parallel to the SLAM and Lines+Plane detection thread it does not update at every keyframe. The model also updates only when a certain number of new vertices are inserted into it. The time is taken to update the model increases as the model size increases. Therefore, the point-based model performs poorly as the model becomes large very quickly and the model update takes longer. It is worth noting that all three methods start at

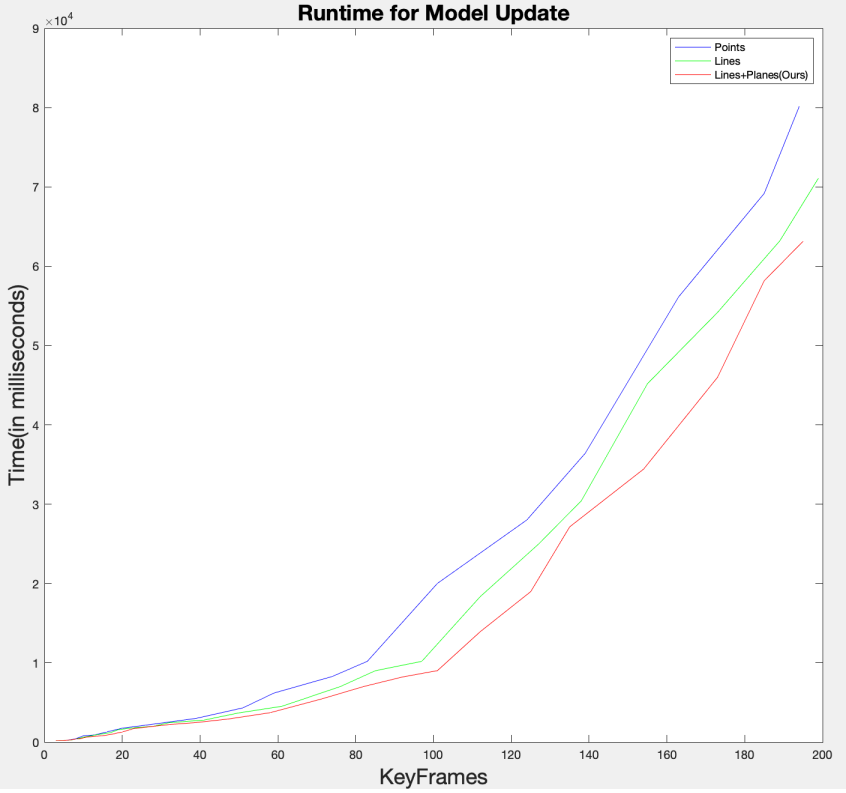


Figure 4.8: Runtime Complexity of Point, Line and Plane(Ours) based surface reconstruction. The timestamp represents the average time(*in milliseconds*) taken for a model update at a Keyframe.(Blue) Point based Modeling (Green) Line based modeling (Red) Line+Plane based modeling(Ours).

a similar pace until 25 keyframes as the semi-dense point cloud boost the line segment detection and plane extraction which is initially used for reconstruction. However, the planes and lines are filtered out during plane matching making the model update faster as new planar structures are discovered.

Chapter 5

Conclusion

In this thesis, we proposed a novel incremental surface reconstruction approach using 3D points and views from Visual SLAM to reconstruct surfaces using planes and lines combined with view-ray constraints to create a free-space scene volume. We analyzed previous approaches and found that one of the biggest concerns in surface reconstruction(from lines) is outlier removal in 3D line segment detection. We improved upon this approach and showed that this method has better outlier detection and removal resulting in a model closer to Ground Truth 3D point cloud on EuRoC MAV benchmarks.

As most of the scenes in benchmarks as well as real-world scenarios are comprised of man-made objects that are mostly planar (boxes, windows, doors, buildings, tables), we used this to leverage the surface reconstruction by detecting planes and using it to triangulate 3D surfaces. Our method aims to reconstruct texture-less areas of objects better than lines and point-based approaches. We were able to achieve real-time performance on our semi-dense reconstruction model over dense or direct triangulation approaches. We were able to show that, compared to methods using dense point cloud and direct triangulation, with fewer model triangles but ensuring that these are more accurately aligned with the scene lines and planes, we get a small model which succinctly represents the scene and objects therein. Our plane and line matching methods make the data association more accurate and in line with the real physical structures.

Limitations and Future Work: Although our method surpasses the existing reconstruction methods in real-time scenarios, we observed that the results in highly non-planar environments were not satisfactory. Our set of experiments shows that the reconstruction result was poor for the EuRoC Machine Hall data sequence which consists mostly of cylindrical pipes and machinery that is not planar. Practically, these results do not allow robotic teleoperation tasks like pipe replacement in hazardous environments. But we can ponder on these situations and think if the use of conics or shape profiles can be used for such environments. A possible solution to address the issue of non-planar objects in the scene could be a hybrid method of planar and shape profile based approach. For more precise tasks like pipe replacement tasks, the knowledge of semantics would help in a top-down reconstruction of the scene.

Tracking and Modeling in a monocular setup using off-the-shelf cameras is always a challenging task in a localized environment. Plane-based SLAM methods would not necessarily perform the best in scenes without loop closures for reconstructing surfaces as loop closure facilitates the reduction of camera pose errors for a consistent global map. For a local setup like a table-based setup for a robot pic-up and place tasks, an RGB-D sensor-based reconstruction would be more efficient in reconstructing local surfaces like in MonoFusion.

Reconstruction using point clouds from feature based SLAMs like ORB-SLAM does not produce accurate results in dynamic scenes. A possible solution is to cull keyframes that have changed to a threshold and replacing them with newer keyframes and points. Inertial Movement Unit (IMU) can be used to enhance the detection of changed keyframes and update them.

Plane and line-based reconstruction methods are reliable in many scenarios to accurately and efficiently reconstruct scenes but they can be generalized for non-planar environments with the help of RGB-D sensors or shape profiles which is an open research problem.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "Slic superpixels compared to state-of-the-art superpixel methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274–2282, 2012. 21

[2] C. Akinlar and C. Topal, "Edlines: Real-time line segment detection by edge drawing," in *2011 18th IEEE International Conference on Image Processing*, 2011, pp. 2837–2840. 14, 17

[3] A. Bartoli, "A random sampling strategy for piecewise planar scene segmentation," *Comput. Vis. Image Underst.*, vol. 105, pp. 42–59, 2007. 19

[4] A. Bartoli and P. Sturm, "Structure-from-motion using lines: Representation, triangulation, and bundle adjustment," *Computer Vision and Image Understanding*, vol. 100, no. 3, pp. 416–441, 2005. 2, 14

[5] M. Berger, A. Tagliasacchi, L. Seversky, *et al.*, "A Survey of Surface Reconstruction from Point Clouds," *Computer Graphics Forum*, p. 27, 2016. DOI: 10.1111/cgf.12802. 26

[6] M. Burri, J. Nikolic, P. Gohl, *et al.*, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016. 9, 36, 44, 49

[7] C. Cadena, L. Carlone, H. Carrillo, *et al.*, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016. 7

[8] J. Civera, A. J. Davison, and J. M. M. Montiel, "Inverse depth parametrization for monocular slam," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 932–945, 2008. 11

[9] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '96, New York, NY, USA: Association for Computing Machinery, 1996, pp. 303–312, ISBN: 0897917464. DOI: 10.1145/237170.237269. [Online]. Available: https://doi.org/10.1145/237170.237269. 30

[10] 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

[11] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," vol. 15, no. 1, pp. 11–15, 1972. 14

[12] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 3, pp. 611–625, 2018. 6, 12

[13] J. J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *ECCV*, 2014. 2, 6, 9–13

[14] O. Faugeras, E. Bras-Mehlman, and J. Boissonnat, "Representing stereo data with the delaunay triangulation," *Artificial Intelligence*, vol. 44, no. 1, pp. 41–87, 1990, ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(90)90098-K. 27

[15] L. Gallagher, V. R. Kumar, S. K. Yogamani, and J. B. McDonald, "A hybrid sparse-dense monocular SLAM system for autonomous driving," *CoRR*, 2021. 10

[16] A. Gee, D. Chekhlov, W. Mayol-Cuevas, and A. Calway, "Discovering planes and collapsing the state space in visual slam," Jan. 2007. 19, 22, 24

[17] A. P. Gee and W. Mayol-Cuevas, "Real-time model-based slam using line segments," in *ISVC*, 2006. 2, 14

[18] R. Grompone von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall, "LSD: a Line Segment Detector," *Image Processing On Line*, vol. 2, pp. 35–55, 2012. 6, 14, 15, 24

[19] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Second. Cambridge University Press, ISBN: 0521540518, 2004. 15

[20] S. He, "Incremental 3d line segments extraction for surface reconstruction from semi-dense slam," *Master's Thesis, University of Alberta*, 2018. 3, 41

[21] S. He, X. Qin, Z. Zhang, and M. Jagersand, "Incremental 3d line segment extraction from semi-dense slam," in *International Conference on Pattern Recognition, ICPR 2018*, 2018. 3, 14, 17, 18, 33, 35, 41,

[22] M. Hofer, M. Maurer, and H. Bischof, "Efficient 3d scene abstraction using line segments," *Computer Vision and Image Understanding*, vol. 157, Mar. 2016. DOI: 10.1016/j.cviu.2016.03.017. 2, 3, 15, 16, 18, 47–49

[23] C. Hoppe, M. Klopschitz, M. Donoser, and H. Bischof, "Incremental surface extraction from sparse structure-from-motion point clouds," Sep. 2013. DOI: 10.5244/C.27.94. 3, 27, 50

[24] M. Hosseinzadeh, Y. Latif, and I. Reid, "Sparse point-plane slam," 2017. 19, 20

[25] J. Jin, L. Petrich, S. He, M. Dehghan, and M. Jagersand, *Long range teleoperation for fine manipulation tasks under time-delay network conditions*, 2019. DOI: 10.48550/ARXIV.1903.09189. 27, 41

[26] M. Kaess, "Simultaneous localization and mapping with infinite planes," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4605–4611, 2015. 2, 19

[27] A. Khatamian and H. Arabnia, "Survey on 3d surface reconstruction," *Journal of Information Processing Systems*, vol. 12, pp. 338–357, Jan. 2016. DOI: 10.3745/JIPS.01.0010. 26

[28] G. Klein and D. Murray, "Improving the agility of keyframe-based slam," Springer Berlin Heidelberg, 2008, pp. 802–815. 6, 8, 9

[29] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2007, pp. 225–234. 6, 8, 27

[30] P. Kohli and P. Torr, "Dynamic graph cuts and their applications in computer vision," in Apr. 2010, vol. 285, pp. 51–108, ISBN: 978-3-642-12847-9. DOI: 10.1007/978-3-642-12848-6_3. 28

[31] P. Labatut, J.-P. Pons, and R. Keriven, "Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts," in *2007 IEEE 11th International Conference on Computer Vision*, 2007, pp. 1–8. DOI: 10.1109/ICCV.2007.4408892. 27

[32] P.-A. Langlois, A. Boulch, and R. Marlet, "Surface reconstruction from 3d line segments," in *2019 International Conference on 3D Vision (3DV)*, 2019, pp. 553–563. 3, 28, 45

[33] M. Lhuillier, "Surface reconstruction from a sparse point cloud by enforcing visibility consistency and topology constraints," *Computer Vision and Image Understanding*, vol. 175, pp. 52–71, Oct. 2018. 26, 27

[34] M. Lhuillier and S. Yu, "Manifold surface reconstruction of an environment from sparse Structure-from-Motion data," *Computer Vision and Image Understanding*, vol. 117, no. 11, pp. 1628–1644, Nov. 2013. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01635450. 27, 44

[35] Y. Ling and S. Shen, "Building maps for autonomous navigation using sparse visual slam features," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1374–1381. 27, 44, 50

[36] V. Litvinov and M. Lhuillier, "Incremental solid modeling from sparse structure-from-motion data with improved visual artifacts removal," in *2014 22nd International Conference on Pattern Recognition*, 2014, pp. 2745–2750. 27

[37] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '87, New York, NY, USA: Association for Computing Machinery, 1987, pp. 163–169, ISBN: 0897912276. 30

[38] D. Lovi, N. Birkbeck, D. Cobza, and M. Jägersand, "Incremental free-space carving for real-time 3d reconstruction," Jan. 2011. 2, 3, 27, 28, 32, 33, 41, 4

[39] X. Lu, Y. Liu, and K. Li, *Fast 3d line segment detection from unorganized point cloud*, 2019. 15

[40] L. Ma, C. Kerl, J. Stückler, and D. Cremers, "Cpa-slam: Consistent plane-model alignment for direct rgb-d slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1285–1291. 2, 19

[41] B. Micusik and H. Wildenauer, "Structure from motion with line segments under relaxed endpoint constraints," in *2014 2nd International Conference on 3D Vision*, vol. 1, 2014, pp. 13–19. 14

[42] R. Mur-Artal and J. Tardos, "Probabilistic semi-dense mapping from highly accurate feature-based monocular slam," Jul. 2015. 2, 11, 12, 32, 33, 44

[43] R. Mur-Artal and J. D. Tardos, "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, 2017. 6, 7, 10, 27

[44] 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, 2015. 8, 12

[45] Y. Nakayama, H. Saito, M. Shimizu, and N. Yamaguchi, "3d line segment based model generation by rgb-d camera for camera pose estimation," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9010, 2015, pp. 459–472. 2, 16

[46] R. A. Newcombe, S. Izadi, O. Hilliges, *et al.*, "Kinectfusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, 2011, pp. 127–136. DOI: 10.1109/ISMAR.2011.6092378. 30

[47] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "Dtam: Dense tracking and mapping in real-time," in *2011 International Conference on Computer Vision*, 2011, pp. 2320–2327. 6, 10, 11

[48] V. Pradeep, C. Rhemann, S. Izadi, C. Zach, M. Bleyer, and S. Bathiche, "Monofusion: Real-time 3d reconstruction of small scenes with a single web camera," in *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2013, pp. 83–88. 10, 11, 30, 31

[49] A. Pumarola, A. Vakhitov, A. Agudo, A. Sanfeliu, and F. Moreno-Noguer, "Pl-slam: Real-time monocular visual slam with points and lines," 2017, pp. 4503–4508. 2, 14

[50] K. Qian, W. Zhao, K. Li, X. Ma, and H. Yu, "Visual slam with boplw pairs using egocentric stereo camera for wearable-assisted substation inspection," *IEEE Sensors Journal*, 2019. 14, 15

[51] L. Roldao, R. Charette, and A. Verroust, "3d surface reconstruction from voxel-based lidar data," Oct. 2019, pp. 2681–2686. DOI: 10.1109/ITSC.2019.8916881. 29, 30

[52] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571. 8

[53] R. F. Salas-Moreno, B. Glocken, P. H. J. Kelly, and A. J. Davison, "Dense planar slam," in *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2014, pp. 157–164. 3, 19, 20

[54] C. Schmid and A. Zisserman, "Automatic line matching across views," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1997, pp. 666–671. 14

[55] C. Schmid and A. Zisserman, "The Geometry and Matching of Lines and Curves Over Multiple Views," *International Journal of Computer Vision*, vol. 40, no. 3, pp. 199–233, 2000. 14

[56] Y. Taguchi, Y.-D. Jian, S. Ramalingam, and C. Feng, "Point-plane slam for hand-held 3d sensors," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 5182–5189. DOI: 10.1109/ICRA.2013.6631318. 21, 23

[57] C. Topal and C. Akinlar, "Edge drawing: A combined real-time edge and segment detector," *Journal of Visual Communication and Image Representation*, vol. 23, no. 6, pp. 862–872, 2012, ISSN: 1047-3203. 33, 55

[58] A. Trevor, S. Gedikli, R. Rusu, and H. Christensen, "Efficient organized point cloud segmentation with connected components," *Proceedings of Semantic Perception Mapping and Exploration, S.*, pp. 1–6, Jan. 2013. 19

[59] B. Triggs, P. Mclauchlan, R. Hartley, and A. Fitzgibbon, "Bundle Adjustment – A Modern Synthesis," in *International Workshop on Vision Algorithms*, Springer-Verlag, Sep. 2000, pp. 298–372. 6, 8

[60] X. Wang, M. Christie, and E. Marchand, "Relative pose estimation and planar reconstruction via superpixel-driven multiple homographies," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 10 625–10 632. 19, 21

[61] ——, "Tt-slam: Dense monocular slam for planar environments," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 11 690–11 696. 3, 19–21

[62] Z. Wang, F. Wu, and Z. Hu, "Msld: A robust descriptor for line matching," *Pattern Recognition*, vol. 42, no. 5, pp. 941–953, 2009. 14

[63] D. Werner, A. Al-Hamadi, and P. Werner, "Truncated signed distance function: Experiments on voxel size," vol. 8815, Oct. 2014, pp. 357–364, ISBN: 978-3-319-11754-6. DOI: 10.1007/978-3-319-11755-3_40. 29

[64] D.-M. Woo, S.-S. Han, Y.-K. Jung, and K.-W. Lee, "Generation of 3d building model using 3d line detection scheme based on line fitting of elevation data," in *Proceedings of the 6th Pacific-Rim Conference on Advances in Multimedia Information Processing - Volume Part I*, Jeju Island, Korea, 2005, pp. 559–569. 16

[65] L. Yang, F. Tan, A. Li, Z. Cui, Y. Furukawa, and P. Tan, "Polarimetric dense monocular slam," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3857–3866. 10–12

[66] S. Yang and S. Scherer, "Monocular object and plane slam in structured environments," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, Jun. 2019. 20

[67] S. Yang, Y. Song, M. Kaess, and S. Scherer, "Pop-up slam: Semantic monocular plane slam for low-texture environments," Oct. 2016, pp. 1222–1229. 19, 20

[68] J. Zhang, G. Zeng, and H. Zha, "Structure-aware slam with planes and lines in man-made environment," *Pattern Recognition Letters*, vol. 127, pp. 181–190, 2019. 2, 19, 20

[69] L. Zhang and R. Koch, "An efficient and robust line segment matching approach based on lbd descriptor and pairwise geometric consistency," *Journal of Visual Communication and Image Representation*, vol. 24, no. 7, pp. 794–805, 2013. 14, 15, 24

[70] L. Zhang, C. Xu, K.-M. Lee, and R. Koch, "Robust and efficient pose estimation from line correspondences," in *Computer Vision – ACCV 2012*, K. M. Lee, Y. Matsushita, J. M. Rehg, and Z. Hu, Eds., 2013. 14

[71] X. Zhang, W. Wang, X. Qi, and Z. Liao, "Stereo plane slam based on intersecting lines," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 6566–6572. 2, 3, 19, 24

[72] H. Zhou, D. Zhou, K. Peng, W. Fan, and Y. Liu, "Slam-based 3d line reconstruction," in *2018 13th World Congress on Intelligent Control and Automation (WCICA)*, 2018, pp. 1148–1153. 14, 15, 17