APPEARANCE AND GEOMETRY ASSISTED VISUAL NAVIGATION IN URBAN AREAS

A Dissertation

by

JOSEPH SUNG LEE

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Dezhen Song
Co-Chair of Committee,	Ricardo Gutierrez-Osuna
Committee Members,	John Keyser
	Byung-Jun Yoon
Head of Department,	Dilma Da Silva

May 2016

Major Subject: Computer Science

ABSTRACT

Navigation is a fundamental task for mobile robots in applications such as exploration, surveillance, and search and rescue. The task involves solving the simultaneous localization and mapping (SLAM) problem, where a map of the environment is constructed. In order for this map to be useful for a given application, a suitable scene representation needs to be defined that allows spatial information sharing between robots and also between humans and robots. High-level scene representations have the benefit of being more robust and having higher exchangeability for interpretation. With the aim of higher level scene representation, in this work we explore high-level landmarks and their usage using geometric and appearance information to assist mobile robot navigation in urban areas.

In visual SLAM, image registration is a key problem. While feature-based methods such as scale-invariant feature transform (SIFT) matching are popular, they do not utilize appearance information as a whole and will suffer from low-resolution images. We study appearance-based methods and propose a scale-space integrated Lucas-Kanade's method that can estimate geometric transformations and also take into account image appearance with different resolutions. We compare our method against state-of-the-art methods and show that our method can register images efficiently with high accuracy.

In urban areas, planar building facades (PBFs) are basic components of the quasi-rectilinear environment. Hence, segmentation and mapping of PBFs can increase a robot's abilities of scene understanding and localization. We propose a vision-based PBF segmentation and mapping technique that combines both appearance and geometric constraints to segment out planar regions. Then, geometric constraints such as reprojection errors, orientation constraints, and coplanarity constraints are used in an optimization process to improve the mapping of PBFs.

A major issue in monocular visual SLAM is scale drift. While depth sensors, such as lidar, are free from scale drift, this type of sensors are usually more expensive compared to cameras. To enable low-cost mobile robots equipped with monocular cameras to obtain accurate position information, we use a 2D lidar map to rectify imprecise visual SLAM results using planar structures. We propose a two-step optimization approach assisted by a penalty function to improve on low-quality local minima results.

Robot paths for navigation can be either automatically generated by a motion planning algorithm or provided by a human. In both cases, a scene representation of the environment, i.e., a map, is useful to specify meaningful tasks for the robot. However, SLAM results usually produce a sparse scene representation that consists of low-level landmarks, such as point clouds, which are neither convenient nor intuitive to use for task specification. We present a system that allows users to program mobile robots using high-level landmarks from appearance data.

ACKNOWLEDGEMENTS

I would like to thank Dr. Ricardo Gutierrez-Osuna for his support and patience with my research during my graduate studies at PRISM Lab. With his support, I was able to receive the SMART scholarship with a job offer and also pursue the research in machine vision, which I am most interested in. Also, his pattern classification and neural networks classes were among the best courses that I have taken in the graduate program.

I would like to sincerely thank my academic father, Dr. Dezhen Song, who has been more than a great advisor, but also a mentor. His constant encouragement to the NetBot Lab members as being top researchers has always been a positive influence on research. Many of the contributions presented in my research originated from his mentorship, and I am most indebted to Dr. Song for all the research training that he has provided.

I would like to thank the PRISM Lab and NetBot Lab members for their friendship and helpful discussions on research. They are the ones who made graduate life fun. I would also like to thank my committee members, Dr. John Keyser and Dr. Byung-Jun Yoon, for their kindness and interest in my work. Last but not least, I would like to thank my family and friends for their prayers and support and also God who has created all things for us to wonder at and enjoy researching.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

TABLE Page

1. INTRODUCTION AND RELATED WORK

Navigation is a fundamental task for mobile robots in applications such as exploration, surveillance, and search and rescue. When a robot navigates through an unknown environment, it has to keep track of its location and also construct a map of the environment using sensor measurements. The former and latter tasks, i.e., localization of the robot and mapping the environment, are coupled problems and is referred as simultaneous localization and mapping (SLAM). For the map generated in SLAM to be useful for a given application, a suitable scene representation needs to be defined.

When the application involves more than a single robot, the scene representation becomes more important. It can be considered as a language for robot-to-robot or human-to-robot interactions, where robots will not understand each other with different scene representations, and humans will have difficulties using a robot without a proper scene representation. (See Fig. 1.1 for illustration.) To mitigate this problem, a suitable scene representation needs to be developed to allow information sharing between robots and improve interactions between humans and robots.

Different types of scene representation have been used for the SLAM problem. In the early 80's, occupancy grid maps [84] have been proposed to represent the environment. Occupancy grid maps typically have fixed resolution, and each cell holds a probability value about the occupancy. In this representation, no assumptions are made on the type of features. At a similar time, landmark-based maps have also been proposed [68]. The advantage of landmark-based map is that it can be more precision. Example features that are used in landmark-based SLAM are points, lines, and planes. Nowadays, the majority of the SLAM work uses landmarks to represent the environment.

To represent the scene, early monocular SLAM work use feature points as landmarks.

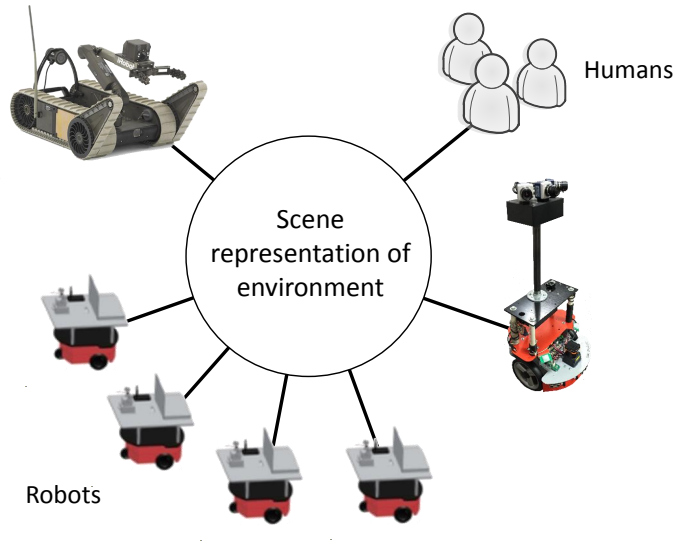


Figure 1.1: A scene representation for robot-to-robot and human-to-robot interactions

Over a decade ago, Davison and Kita [20] have presented a SLAM framework using an extended-Kalman filter (EKF) for a controlled robot. Subsequently, Davison [19] demonstrates a real-time monocular SLAM with a hand-held camera using a constant velocity model. This method is termed as MonoSLAM in [21]. Klein and Murray [58] present a feature point-based parallel tracking and mapping (PTAM) system for a real-time augmented reality application. To achieve real-time performance, both MonoSLAM and PTAM use sparse point clouds to map the environment which results in limited scene reconstruction.

While point features are usually more accurate and easy to detect, many lines exist in structured environment which makes it useful for depth estimation. Smith et al. [95] present a real-time monocular SLAM with straight lines. In their work, lines are represented by their two endpoints so that it can be easily integrated into the point-based MonoSLAM framework. Lemaire and Lacroix [67] use Plücker coordinates to represent 3D lines for an EKF-based SLAM. Eade and Drummond [27] presents a method using

edgelets for landmarks in a particle filter framework. Each edgelet is represented by the center point and direction of the edglet. Zhao et al. [108] parameterizes line features by the azimuth and elevation angles of the plane passing each camera center and observed line segment.

Planes are also used for landmarks in vision-based SLAM and can provide a more semantically meaningful and compact representation of the environment. Molton et al. [83] present a real-time SLAM framework that estimates the normal of small planar patches. However, these plane normals are not included in the state vector of the EKF. Gee et al. [40] develop a method that reduces the state size in the EKF by collapsing points lying on a common plane. Haines et al. [45] show the potential of using machine learning for mapping planes. Their method learns planar structures from appearance and use it for 3D plane mapping in keyframes. Concha and Civera [14] presents a method that uses the contour of segmented regions to estimate planes in monocular SLAM. In their work, segmented regions are assumed to be planar regions and planes are initialized through hypothesis testing. The results show that featureless planes can be correctly estimated in a SLAM framework.

A combination of the above landmarks have also been used. In [41], Gee et al. detect lines and planes during the SLAM process. Martinez-Carranza and Calway [81] present a method that unifies the point and plane representation by adding normal vectors to each 3D point. In [77], Lu et al. propose a building facade mapping method where they use a multi-layered feature graph (MFG) [69] that expresses the relationship between points, lines, and planes for robust estimation.

The landmarks described above are from different levels of the scene representation spectrum illustrated in Fig. 1.2, where feature points and point clouds are at the lowest-level and planes/regions and objects are at the higher-end which has more semantic meaning. Different scene representations have different properties: low-level features are easy

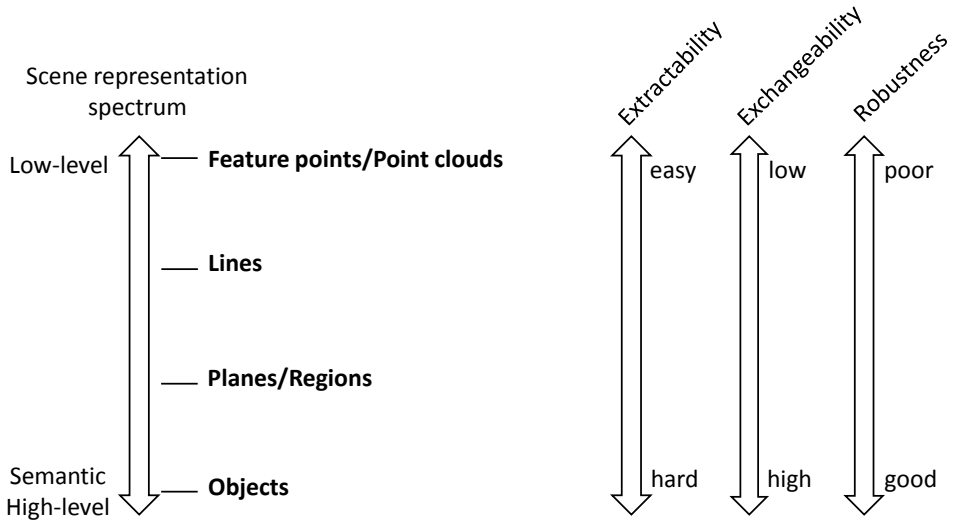


Figure 1.2: Scene representation spectrum and properties

to extract but difficult to be shared among different robots for interpretation, while high-level landmarks are more difficult to extract but are more robust to use for SLAM applications. With the aim of higher level scene representation, in this work we further develop landmarks and their usage using geometric and appearance (i.e., a region of pixel intensities) information to assist mobile robot navigation in urban areas. A brief introduction to each chapter is given in the following.

In chapter 2, we study the image registration problem which is a key problem in visual SLAM. While feature-based methods such as scale-invariant feature transform (SIFT) [76] matching is popular for image registration, it does not utilize pixel intensities as a whole and will suffer from registering low-resolution images without many features. We study appearance-based methods and propose a method that can register planar regions with sub-pixel accuracy based on their appearance. This is achieved by integrating the scale-space model in SIFT into the appearance-based Lucas-Kanades method [79] so that it can estimate geometric transformations and also take into account images with different resolutions. We compare our method against state-of-the-art methods and show that our

method can register images efficiently with high accuracy.

In chapter 3, we propose a vision-based planar building facade (PBF) segmentation and mapping technique. In urban areas, PBFs are basic components of the quasi-rectilinear environment. Hence, segmentation and mapping of PBFs can increase the robot's abilities of scene understanding and localization. Our method first combines both appearance and geometric constraints to segment out planar regions. Then, geometric constraints such as reprojection errors, orientation constraints, and coplanarity constraints are used in an optimization process to improve the mapping of PBFs. We have tested our method on the building structures at Texas A&M University. The experiments included segmenting out PBFs from images and estimating their 3D position from two views with short baseline. The results show that our method reduces the angular error of the reprojection error-based 3D mapping by an average of 82.82%.

Visual SLAM using a monocular camera often generates low-quality maps due to scale drift. On the other hand, while depth sensors, such as lidar, are free from scale drift, these type of sensors are usually more expensive and less portable. To overcome these issues, in chapter 4, we consider the case where a 2D lidar map is pre-built using a single lidar sensor and shared by multiple mobile robots equipped with a monocular camera. This will enable low-cost mobile robots to rectify imprecise visual SLAM results to obtain accurate position information. Using vertical planar structures which can be identified in both sensing modalities, we propose a two-step optimization approach to address the high-dimensional map fusion problem.

Finally, we present a system in chapter 5 that allows users to program mobile robots for navigation. For navigation, a robot path needs to be preprogrammed, automatically generated by a motion planning algorithm, or given by a human in a teleoperation scenario. In these cases, a good scene representation of the environment is needed to specify meaningful tasks for the robot. However, SLAM results usually produce a sparse scene

representation that consist of low-level landmarks, such as point clouds, which are neither convenient nor intuitive to humans, who normally have poor spatial reasoning. The average user will need easier and flexible robot programming tools for mobile robots as opposed to the specialized programming skills used for industrial robots [8]. Our system allows the user to program the robot with a set of motion commands and high-level landmarks selected directly from appearance data. The system automatically generates a robot path from these high-level commands and can be used to for robot navigation.

2. APPEARANCE-BASED IMAGE REGISTRATION WITH SCALE-SPACE*

2.1 Introduction

Image registration is a crucial step in a variety of computer vision tasks, from image stitching to object recognition and super-resolution (SR). In general, registration aims to align two images (the *reference* image and the *observed* image) by estimating translation, rotation, and scale. Mainly two image registration approaches exist: *feature-based* and *area-based* methods; the former tries to align images using local features, whereas the latter uses the whole image region [110]. Classical examples of each approach are scale-invariant feature transform (SIFT) matching [76] and Lucas-Kanade's (LK) method [79], respectively.

In SIFT matching, scale-invariant local features are extracted from the two images using a *scale-space* model [73] and then corresponding features are matched. The advantages of SIFT are that it can register images with high disparity and that it is robust to occlusion. However, if the image region is small, as is the case of very low resolution images, SIFT becomes problematic since the images will not contain enough features to be matched. SIFT is also computationally intensive because it first builds an explicit *image pyramid* [1] structure, which approximates the scale-space, by downscaling the images and then estimates scale changes at every level. In case of a video, where neighbouring frames have similar scale, estimating scale changes at every level becomes unnecessary. Also, since feature matching is usually followed by RANSAC [35], to reject outlier correspondences, the computational load will increase.

On the other hand, the LK method jointly optimizes the affine registration parame-

*Reprinted with permission from "SILK: Scale-space integrated Lucas-Kanade image registration for super-resolution from video" by J. Lee, R. Gutierrez-Osuna, S. S. Young, 2013 IEEE International Conference on Acoustics, Speech and Signal Processing ©2013 IEEE.

ters with an area-based approach. The advantage of this method is that it can efficiently register images by searching in the gradient direction and that it avoids the feature correspondence problem by using the whole image. In the LK method, the relationship between the reference image I_r and observed image I_o is expressed as

$$I_o(\mathbf{x}) = I_r(\mathbf{w}(\mathbf{x};\mathbf{p})) \ , \tag{2.1}$$

where $\mathbf{w}(\mathbf{x};\mathbf{p})$ warps the two-dimensional coordinate $\mathbf{x}$ with affine transformation parameters $\mathbf{p}$ [6]. The algorithm iteratively estimates $\mathbf{p}$ by minimizing $\|I_o(\mathbf{x}) - I_r(\mathbf{w}(\mathbf{x};\mathbf{p}))\|^2$. A drawback of this approach, however, is that equation (2.1) does not consider changes in image intensity as in SIFT, which occur when the scale changes (i.e., due to image resolution) [23].

Motivated by the limitation of these two classical problems when dealing with LR video, we present an image registration technique for SR that integrates the scale-space model of SIFT into the LK framework; this is achieved by embedding the Gaussian kernels used in the scale-space model into a non-linear least squares estimation. We adopt the area-based approach since the method has to work on very small image regions without many features, while the scale-space model is adopted to handle scale changes correctly. Additionally, the non-linear least squares framework allows us to avoid building a predetermined pyramid structure and compares the two images only at the scale levels selected by the gradient direction. Thus, the method has the efficiency of LK and can simultaneously estimate sub-pixel translation, rotation, and scale while still addressing the resolution issue of equation (2.1). Though the latter issue can be somewhat alleviated by interpolation or smoothing, our approach handles this issue naturally by explicitly relating the image scale to image resolution. This allows the method to automatically smooth and downsample (anti-alias) correctly in scale-space for each iteration update.

2.2 Related Work

Our work is related to SR registration methods based on the LK algorithm. One of the earliest algorithms was proposed by Keren et al. [56]. To obtain high-accuracy sub-pixel motion, the authors developed a spatial domain technique that iteratively estimates translation and rotation. The classical work by Irani and Peleg [50] also applied this registration method and used a back-projection kernel to update the SR image. Recently, Costa et al. [16] examined the effect of registration errors under x-y translation using the LK method; the authors show that some degree of registration errors can be advantageous for SR since they provide some degree of regularization. These previous studies, however, do not consider the pixel intensities affected by image resolution and scale; which our proposed method takes into account. Besides the LK registration, SIFT matching has also been used recently for SR registration. Vrigkas et al. [105] have used SIFT matching to estimate registration parameters and generate higher-resolution images using mutual information. Ferreira et al. [34] present an example-based video super-resolution method with mixed-resolution image sequences. However, the images used in these studies are of sufficient resolution to contain many features. In contrast, our aim is to super-resolve images with a small image size with very low resolution, where a 2-fold increase in resolution can have a significant impact.

2.3 Image Model

Consider the problem of registering two images $I_o(x,y)$ and $I_r(x,y)$, defined by their pixel intensities at index (x,y). Namely, we wish to find the registration parameters t_x, t_y, θ, and s that minimize the difference between I_r and I_o where (t_x,t_y) is the x-y translation, θ is the rotation angle, and s is the scale factor.

The scale factor between I_r and I_o determines the inherent image resolution. The rela-

tionship between these two images can be expressed as

$$I_o(i,j) = \sum_{x,y} G(x,y;i,j,s) I_r(x,y) \ , \tag{2.2}$$

where (x,y) are the indices of reference image I_r, and (i,j) are the indices of image I_o, and the scale parameter s determines the width of a two-dimensional Gaussian kernel G:

$$G(x,y;x_m,y_m,\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-x_m)^2+(y-y_m)^2}{2\sigma^2}} \ , \tag{2.3}$$

where σ is the standard deviation and (x_m,y_m) is the mean vector. By using a Gaussian kernel, we ensure that the number of local extrema of image intensity does not increase at coarser levels [73]. The model in equation (2.2) represents a single observation pixel as a weighted sum of all the pixels in the reference image.

For convenience, we assume that the pixel width and height of the reference image $I_r(x,y)$ are set to unit length. Thus, if the observed image is scaled by a factor s relative to the reference image, an observed pixel will cover a width of $1/s$ in the reference image. Since most of the mass in a Gaussian density is contained within $\pm 3\sigma$, equating the observation pixel and the Gaussian window yields $6\sigma = 1/s$. Therefore, the standard deviation σ can be expressed in terms of the scaling factor s as $\sigma = 1/(6s)$.

The mean location of the kernel $G(x,y;i,j,s)$ can be expressed in terms of the observed image coordinates (i,j) and the observed image pixel width $1/s$ as

$$\begin{bmatrix} x_m \\ y_m \end{bmatrix} = \begin{bmatrix} x_o + (i-1)/s + 1/(2s) \\ y_o + (j-1)/s + 1/(2s) \end{bmatrix} \ , \tag{2.4}$$

where (x_o,y_o) are the coordinates of the image origin of the observed image. Substituting

expressions σ and (x_m, y_m) in the kernel equation (2.3) yields

$$G(x,y;i,j,s) = \frac{1}{2\pi\left(\frac{1}{6s}\right)^2} e^{-\frac{\left(x-x_o-\frac{2i-1}{2s}\right)^2+\left(y-y_o-\frac{2j-1}{2s}\right)^2}{2\left(\frac{1}{6s}\right)^2}} . \tag{2.5}$$

The above equation expresses the Gaussian weights as a function of the observation pixel locations and scale parameter. Therefore, when combined with equation (2.2) we are able to relate I_r and I_o with a single scale parameter s. More importantly, because the scale parameter is a continuous variable, the observed image I_o can be simulated at any scale in the scale-space of the reference image I_r. When compared to equation (2.1), though, the scale parameter is no longer linear as in the transformation $\mathbf{w}(\mathbf{x};\mathbf{p})$.

In order to generalize equation (2.2), translation and rotation parameters t_x, t_y, and θ are incorporated into the Gaussian kernel mean location. The mean location (x_m, y_m) of the Gaussian window can be transformed by

$$\begin{bmatrix} x'_m \\ y'_m \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_m \\ y_m \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} . \tag{2.6}$$

Therefore, we can generalize equation (2.5) as

$$G(x,y;i,j,t_x,t_y,\theta,s) = \frac{1}{2\pi\left(\frac{1}{6s}\right)^2} e^{-\frac{(x-x'_m)^2+(y-y'_m)^2}{2\left(\frac{1}{6s}\right)^2}} . \tag{2.7}$$

Finally, the model equation (2.2) for the reference image and the observed image becomes

$$I_o(i,j) = \sum_{x,y} G(x,y;i,j,t_x,t_y,\theta,s) I_r(x,y) . \tag{2.8}$$

As in equation (2.5), the translation and rotation parameters are no longer linear terms. In contrast with the previous models, though, equation (2.8) allows us to take into account

the physical effects of the scale parameter when representing each observed pixel $I_o(i,j)$ as a weighted sum of the reference pixels $I_r(x,y)$.

2.4 Image Registration Algorithm

Since the registration parameters t_x, t_y, θ, and s are non-linear terms of the model equation (2.8), we estimate them by non-linear least-squares using Gauss-Newton method [17]. Namely, we wish to find an estimate $\hat{\mathbf{p}}$ that minimizes the objective function

$$J = \frac{1}{2}[\tilde{\mathbf{y}} - \mathbf{f}(\hat{\mathbf{p}})]^T W [\tilde{\mathbf{y}} - \mathbf{f}(\hat{\mathbf{p}})] \,, \tag{2.9}$$

where $\tilde{\mathbf{y}}$ is the observed image in raster scan order and $\mathbf{f}(\hat{\mathbf{p}})$ is a transformed image of the reference image with registration parameters $\hat{\mathbf{p}}$; i.e., $\mathbf{f}(\hat{\mathbf{p}})$ is obtained from equation (2.8). The weight matrix W in equation (2.9) allows us to emphasize certain observation pixels. For the experiment in this study, we use an identity matrix, which puts equal weight to each observed pixel.

An initial value $\mathbf{p}_c$ for $\hat{\mathbf{p}}$ is required to start the estimation process. Since adjacent frames are close to each other, we assume that a good initial estimate between the reference and observed image is provided. For a given estimate $\mathbf{p}_c$, its goodness is computed by the error term $\Delta\mathbf{y}_c = \tilde{\mathbf{y}} - \mathbf{f}(\mathbf{p}_c)$. The Jacobian matrix $\frac{\partial \mathbf{f}}{\partial \mathbf{p}}$, which expresses the linear change of the predicted image at current state $\mathbf{p}_c$, is computed as

$$\frac{\partial \mathbf{f}}{\partial \mathbf{p}} = \begin{bmatrix} \left.\frac{\partial f_1}{\partial p_1}\right|_{\mathbf{p}_c} & \cdots & \left.\frac{\partial f_1}{\partial p_4}\right|_{\mathbf{p}_c} \\ \vdots & \ddots & \vdots \\ \left.\frac{\partial f_m}{\partial p_1}\right|_{\mathbf{p}_c} & \cdots & \left.\frac{\partial f_m}{\partial p_4}\right|_{\mathbf{p}_c} \end{bmatrix} \,, \tag{2.10}$$

where $[p_1, p_2, p_3, p_4]^T = [t_x, t_y, \theta, s]^T$ and m is the number of observation pixels. Once $\Delta\mathbf{y}_c$

and $\frac{\partial \mathbf{f}}{\partial \mathbf{p}}$ have been computed, the correction term can be expressed as:

$$\Delta \mathbf{p} = \left[\left(\frac{\partial \mathbf{f}}{\partial \mathbf{p}} \right)^T W \left(\frac{\partial \mathbf{f}}{\partial \mathbf{p}} \right) \right]^{-1} \left(\frac{\partial \mathbf{f}}{\partial \mathbf{p}} \right)^T W \Delta \mathbf{y}_c \ . \tag{2.11}$$

Once the correction term $\Delta \mathbf{p}$ has been calculated, the current state estimate $\mathbf{p}_c$ is updated as $\mathbf{p}_c = \mathbf{p}_c + \Delta \mathbf{p}$. The above process continues iteratively until the maximum number of iterations is reached or when the stopping criteria given by $\frac{|J_i - J_{i-1}|}{J_i} < \frac{\varepsilon}{\|W\|}$ is satisfied, where $J_i = \Delta \mathbf{y}_c^T W \Delta \mathbf{y}_c$ is the predicted residual at the i-th iteration and ε is a small value that determines the tolerance [17].

How does the proposed update equation compare to the LK method? In the original LK algorithm, the parameter $\mathbf{p}$ is updated according to:

$$\Delta \mathbf{p} = \left[\sum_{\mathbf{x}} \left(\nabla I_r \frac{\partial \mathbf{w}}{\partial \mathbf{p}} \right)^T \left(\nabla I_r \frac{\partial \mathbf{w}}{\partial \mathbf{p}} \right) \right]^{-1} \sum_{\mathbf{x}} \left(\nabla I_r \frac{\partial \mathbf{w}}{\partial \mathbf{p}} \right) \Delta \mathbf{y}_c \ , \tag{2.12}$$

where $\Delta \mathbf{y}_c = I_o(\mathbf{x}) - I_r(\mathbf{w}(\mathbf{x}; \mathbf{p}_c))$. When compared to our update equation (2.11), the LK update in equation (2.12) multiplies the gradient image $\nabla I_r = (\frac{\partial I_r}{\partial x}, \frac{\partial I_r}{\partial y})$ with the Jacobian warp to compute the steepest descent image $\nabla I_r \frac{\partial \mathbf{w}}{\partial \mathbf{p}}$. The LK update has the advantage that the gradient image is computed only once, and only the Jacobian warp needs to be computed in each iteration. However, the LK model does not correctly represent how the images are affected by scale changes. Instead of using only the gradient image ∇I_r in the x-y directions, our method directly computes the gradient $\frac{\partial \mathbf{f}}{\partial \mathbf{p}}$ with respect to x-y translation, rotation, and scale in scale-space.

To compare the computational cost with LK, assume n registration parameters and an image I_o of size m. For each iteration, the computational complexity of computing $\nabla I_r \frac{\partial \mathbf{w}}{\partial \mathbf{p}}$ in equation (2.12) is $O(mn)$ [6]. In our proposed method, when the Gaussian kernel size is k at an iteration, the algorithm will take time $O(kmn)$ to compute $\frac{\partial \mathbf{f}}{\partial \mathbf{p}}$ according to equation

(2.10).

2.5 Experiments

We tested our proposed method on registering LR license-plate images. The registered images were then used for SR. We first demonstrate the proposed technique on synthetic sequences of LR frames where the ground-truth is known. Then we use real image sequences to generate SR images. For each experiment, we compared our proposed method against both LK optical flow and SIFT matching. The LK optical flow implementation in OpenCV [10] and Lowe's SIFT implementation [75] were used for comparison.

2.5.1 Synthetic Image Sequences

For the simulated experiment, we use the 91 images in the license-plate category of the Caltech-256 [44] dataset. We manually cropped the license-plate regions from each image and generated 100 LR frames for each sequence by applying translation, rotation, and scale. For translation, the *x*-*y* translation followed a spiral trajectory $(x,y) = (at\cos(2t), at\sin(2t))$ where a is the amplitude and t is the frame index. Throughout the frame sequence, the license-plate moved within ± 2 pixels in the LR domain. For rotation, a random value within ± 5 degrees was applied to each frame. The downscaling was performed by pixel averaging (as described in [5]) so that each LR sequence was no greater than 40x40 pixels.

We illustrate the performance of the method on three image registration scenarios: image sequences with (1) *x*-*y* translation only, (2) *x*-*y* translation and rotation, and (3) *x*-*y* translation, rotation, and scaling. For the LK optical flow method, we used a 3x3 kernel to smooth the images beforehand so that the method does not suffer from aliasing effects, and we used a 7x7 window to compute the flow vectors [74]. After computing the dense optical flow of LK, we estimate the global transformation using the flow vectors. Our proposed method did not require any manual smoothing and worked directly on the images.

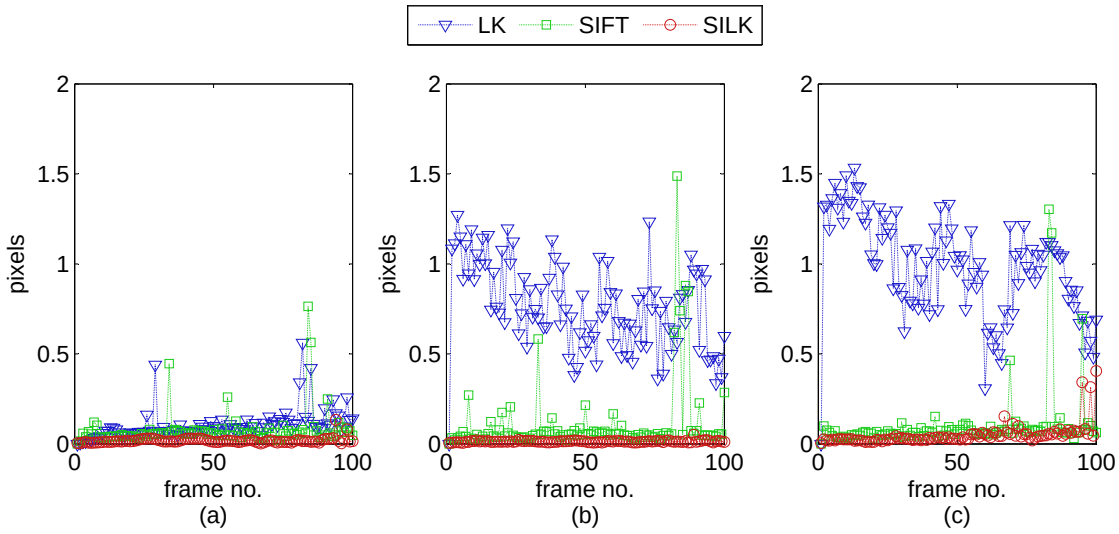


Figure 2.1: RMS error between ground-truth and estimated parameters for the three synthesized sequences: (a) translation only, (b) translation and rotation, (c) translation, rotation, and scale. For each scenario, SIFT includes only (a) 18%, (b) 11%, and (c) 10% of the 91 image sequences.

Fig. 2.1 shows the root-mean-square (RMS) error of the length $\sqrt{t_x^2 + t_y^2}$ between the ground-truth and estimated parameters (t_x, t_y). In particular, these results indicate that the LK optical flow method can work relatively well when there is only translation. However, as rotation and scale changes are introduced, the LK optical flow becomes less accurate. In addition, and as shown in Fig. 2.1(b) and (c), registration errors of the LK optical flow can be more than a pixel width, when sub-pixel accuracy is required for SR. Although SIFT matching appears to be able to effectively estimate the registration in each scenario, Fig. 2.1 only includes the cases when SIFT was able to find more than 5 feature matches between the reference and observed images. Overall, SILK is almost always more accurate than SIFT matching unless SIFT can find enough features, in which case both algorithms perform comparably.

We also compared SIFT and SILK in terms of run time; although SIFT is known to be computationally intensive for typical sizes of images, our experiment involves only a small

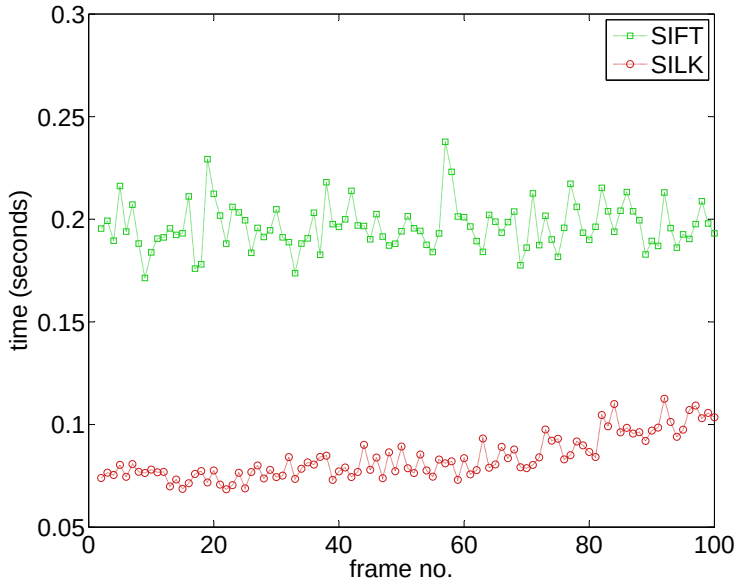


Figure 2.2: Speed comparison between SIFT keypoint detection and SILK. Each data point represents the average run time of the third scenario in Fig. 2.1(c). As shown, SIFT keypoint detection takes approximately constant time for each image. In contrast, registration time for SILK depends on the number of iterations performed for each image match.

image region. Results are summarized in Fig. 2.2, which shows the average run time over the 9 image sequences in the third scenario of Fig. 2.1(c) (translation, rotation, and scale). On a dual-core 2.6GHz processor with 6MB cache, SIFT takes an average of 0.20 seconds per frame. In contrast, our method takes 0.08 seconds per frame or twice as fast as the SIFT feature extraction. However, because SILK uses a gradient search, the registration requires more iterations as the images become more separated, as shown in Fig. 2.2.

Finally, we compared the three algorithms in terms of their ability to produce accurate registrations for SR. Namely, given the estimated registration data from each image sequence, we construct a linear system $\mathbf{y} = H\mathbf{x}$ as in [72] where $\mathbf{x}$ is the unknown high-resolution pixels and H is the weight matrix that relates the high-resolution pixels to the observation pixels $\mathbf{y}$. Since the matrix H is sparse, we use LSQR [90] to super-resolve the

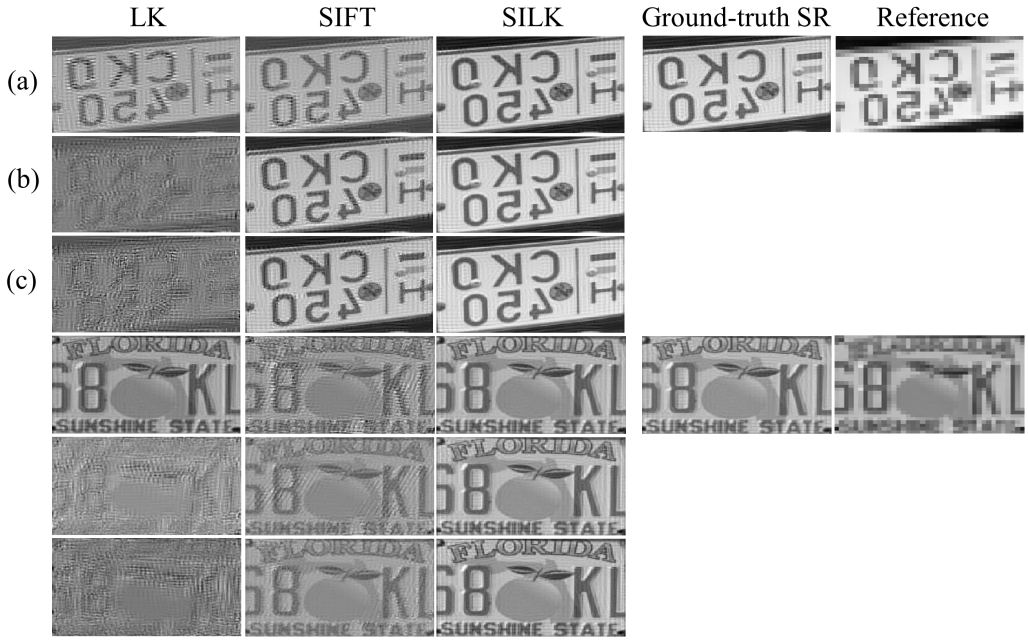


Figure 2.3: SR results from synthetic sequences of images; the image sequence was obtained by applying the three transformation scenarios shown in Fig. 2.1. The reference image (*rightmost*) was super-resolved after aligning the frame sequence. For each sample, the row corresponds to each scenario: (a) translation only (b) translation and rotation (c) translation, rotation, and scale. The first three columns show the reconstruction results from LK optical flow, SIFT matching, and SILK. The fourth column shows the SR result with ground-truth alignment.

reference image [88]. Fig. 2.3 shows some examples of the super-resolved images. The SR results are consistent with the previous plots: (1) LK performs well when there is only translation, and (2) SIFT is robust to rotation and scale.

2.5.2 Real Image Sequences

For our second experiment, we applied the method to register real LR image sequences. We used a web-camera and captured a video of stationary cars from a distance so that the text on the license-plate would be difficult to read. The license-plate regions were smaller than a 20x40 image; see Fig. 2.4. The web-camera was hand held so that a jittering motion could be applied. For each sample sequence, we captured 300 frames. We also took an

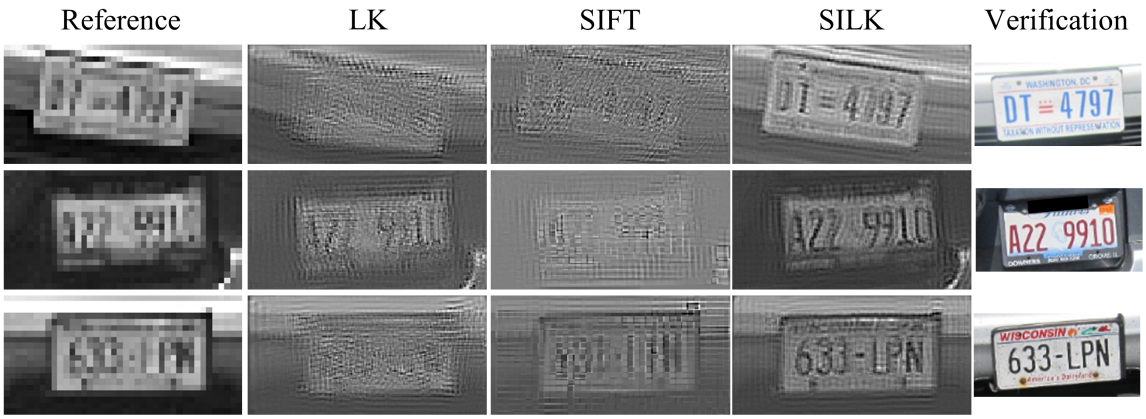


Figure 2.4: SR results from registration parameters of real image sequences. The reference image (a) is reconstructed using (b) LK optical flow, (c) SIFT matching, and (d) proposed SILK. Column (e) are the verification images. For each sample sequence, SIFT was able to match 16, 5, and 3 frames out of 300 frames.

image of the license plate at a closer distance to verify the characters on the license plate after super-resolving an image.

The same image fusion method was used as in the previous experiment for SR. Reconstruction results are shown in Fig. 2.4. After registering the LR images with SILK, we were able to obtain a super-resolved image where the characters on the license-plate become readable. In contrast, SIFT matching failed to find enough features for most of the frames; less than 6% of the frames had more than 5 matches). As shown in the figure, LK optical flow gave poor reconstruction results as well.

2.6 Conclusions

In this work we have presented an image registration technique (SILK) that can simultaneously estimate translation, rotation, and scale between images and also take into account intensity differences caused by discretization. Compared to SIFT matching, our method assumes that a rough initial estimate is given. However, our method can estimate

scale changes as accurately as SIFT and more efficiently in the continuous scale-space. Compared to the original LK method, the improvements in accuracy comes at the price of higher computational complexity; however, when run on LR image regions (e.g., 40x40) the algorithm can run at frame rates on a contemporary office workstation. Our experimental results show that SILK can outperform LK optical flow and SIFT matching for the purpose of SR from very low resolution image sequence when translation, rotation, and scaling are present.

3. PLANAR BUILDING FACADE MAPPING FROM TWO VIEWS*

3.1 Introduction

Vision is an important sensory modality for robots navigating in GPS-challenged environments. Since such environments are mostly man-made urban environments, they are often quasi-rectilinear. Planar building facades (PBFs) can be viewed as basic components of quasi-rectilinear environments. For better scene understanding and the establishment of high level landmarks for robust motion estimation, it is important to detect PBFs. Unfortunately, this is a nontrivial task. At first sight, PBFs may be detected using homographies between two images due to their geometric relationship. However, the plane homography can be difficult to be obtained when a PBF is far away relative to the baseline distance between the two camera views. This often happens when a monocular robot takes an image with a small step or when the baseline of a stereo camera is limited by the physical size of the robot. Also, the non-planar objects in the scene often bias the homography estimation.

In this study, we propose a method that uses both geometric and appearance information to extract PBFs and uses geometric constraints to refine the PBF mapping. Fig. 3.1 partially illustrates our approach. The contribution of this work is twofold: 1) we propose a rectilinear index metric which allows us to identify building regions from its surroundings. Then we identify homographies using vanishing points as constraints for better planar surface detection, and 2) we combine three geometric constraints, i.e., the reprojection error, orientation constraint, and coplanarity constraint, in an optimization process to improve the mapping of the building structure. We have tested our method in physical experiments. We compare our algorithm with the state-of-the-art J-linkage-based PBF de-

*Reprinted with permission from "Planar building facade segmentation and mapping using appearance and geometric constraints" by J. Lee, Y. Lu, D. Song, 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems ©2014 IEEE.

tection [37] and reprojection error-based 3D mapping. The results show that our method outperforms the J-linkage-based approach by an average of 45.27% precision increase and reduces the angular error of the reprojection error-based 3D mapping by an average of 82.82%.

3.2 Related Work

Reconstructing PBFs in a man-made environment has been widely studied for visualization and robot navigation. For visualization, computer graphics research has focused on reconstructing complex and accurate architectural models using polygonal meshes [54]. This usually requires either repetitive scene scanning or range finders that can produce dense measurements with high precision. In this work we do not focus on fully reconstructing the robot's environment since this is often not the main task in robot navigation.

In robot navigation, when a range finder is used, planes can be mapped by directly fitting a 3D plane model to the depth data. For example, in [54], a real-time plane segmentation algorithm is developed for noisy 3D point clouds acquired by less accurate and portable LIDAR. Recently, as RGB-D sensors became more available, a number of researchers have used these sensors to map planar surfaces in an indoor environment [101, 65, 99]. Delmerico et al. [24] present a stereo-based building facade mapping method which fits a plane model in the 3D disparity space. In their facade detection step, local surface normals are first computed for each point in 3D with a Markov random field model. Then, random sample consensus (RANSAC) [35] is used to progressively cluster these points with the same plane normal one plane at a time. Although they do not use a range finder, their plane mapping method is similar to the range finder-based methods in that the plane is directly fitted in the 3D space. Our work only uses a passive vision sensor which does not provide depth measures. However, this allows us to explore the usage of appearance data for planar surface segmentation in 2D prior to estimating building facades

in 3D so that noise from non-planar objects can be reduced.

For vision-based navigation, detection of planes in the 2D domain precedes the 3D mapping. Plane detection in 2D is usually done by using the homography constraint. Since RANSAC is not suited to detect multiple models, Fouhey et al. [37] use an agglomerative clustering technique, called J-linkage [103], to detect multiple building facades from two images using the homography model. Baillard and Zisserman [4] propose a method that reconstructs piecewise planar building rooftops from multiple aerial images using half-planes and their associated homographies. Zhou and Li [109] develop a modified expectation-maximization (EM) algorithm that uses the homography constraint to classify ground-plane pixels and non-ground plane pixels for a mobile robot. Other existing methods use appearance information to help planar surface detection and mapping. Li and Birchfield [71] use an image-based segmentation method to detect ground planes in an indoor corridor. In their method, vertical and horizontal line segments are used to find the wall-floor boundary. Visual mapping has also been done using learned structural priors from appearance data [45]. In our work, we use both geometric and appearance information to help the homography detection process.

Previously, a building facade mapping method was proposed in [77] where they use a multi-layered feature graph [69] for robust estimation of building facades. For better reconstruction results, the reconstruction process enforces geometric constraints such as parallelism and coplanarity. In this work we focus on extracting coplanar features using appearance data and use different cost functions for the coplanarity constraint.

3.3 Problem Statement

3.3.1 Notations

We denote two input images by I and I', where the prime symbol $(')$ denotes the entities of the second image. Feature points in projective space $\mathbb{P}^2$ and $\mathbb{P}^3$ are denoted as $\mathbf{x}$ and

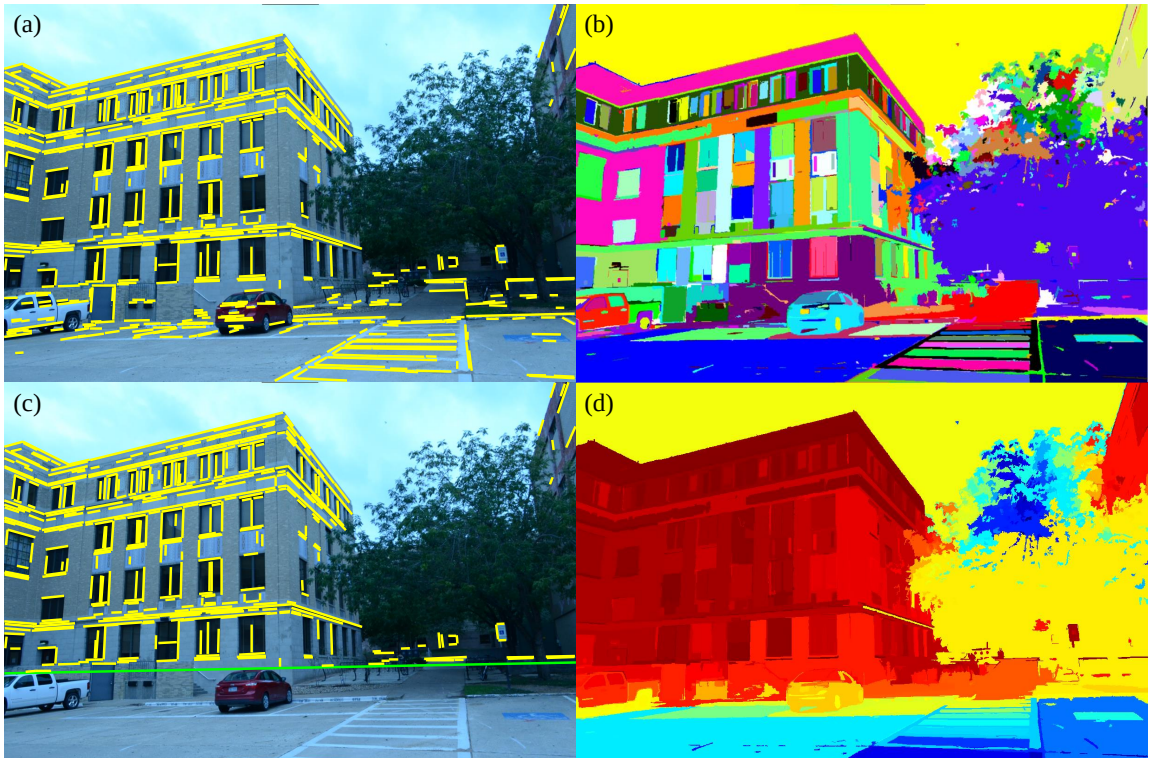


Figure 3.1: Computing rectilinearity of regions. (a) Detected line segments on an image (b) Segmented homogeneous regions (color-coded) (c) Ground line segments removed below the (green) horizontal vanishing line. (d) Heat map visualization of rectilinearity indices. Regions with warmer color indicate rectilinear regions.

$\mathbf{X}$, respectively. Similarly, $\mathbf{e} \in \mathbb{P}^2$ and $\mathbf{E} \in \mathbb{P}^3$ denote line-segment endpoints. We define line segments by their two endpoints, i.e., $\mathbf{s} := (\mathbf{e}_1, \mathbf{e}_2)$ and $\mathbf{S} := (\mathbf{E}_1, \mathbf{E}_2)$. A plane in $\mathbb{P}^3$ is defined by $\Pi := [\mathbf{n}^\mathsf{T}, d]^\mathsf{T}$, where $\mathbf{n}$ is the plane normal and $d/\|\mathbf{n}\|$ is the distance from the plane to the origin. Formally, we define:

Definition 1 (PBF) *A* PBF *in I is defined as a 3-tuple*

$$\phi := (X, S, G), \tag{3.1}$$

where X and S denote a set of n_x feature points $\{\mathbf{x}_i\}_{i=1}^{n_x}$ and a set of n_s line segments

$\{\mathbf{s}_i\}_{i=1}^{n_s}$, *respectively, which lie in the pixel region G. In 3D world coordinates, a PBF is defined by*

$$\Phi := (X, S, \Pi), \tag{3.2}$$

where X and S denote a set of feature points $\{\mathbf{X}_i\}_{i=1}^{n_x}$ *and a set of line segments* $\{\mathbf{S}_i\}_{i=1}^{n_s}$, *respectively, which are associated to plane* Π.

3.3.2 Assumptions

We make the following assumptions in our approach.

- Lens distortion is removed from I and I'.
- The intrinsic camera parameter matrices $\mathtt{K}$ and $\mathtt{K}'$ are known from pre-calibration.
- The distance between the camera centers for I and I' is known and nonzero.

3.3.3 Problem Definition

Problem 1 *Given I and* I', *extract a set of* n_f *corresponding PBFs* $\{\phi_i\}_{i=1}^{n_f}$ *and* $\{\phi_i'\}_{i=1}^{n_f}$ *and map their 3D positions* $\{\Phi_i\}_{i=1}^{n_f}$ *relative to the cameras. Also, estimate the extrinsic camera parameters of rotation* $\mathtt{R}'$ *and translation* $\mathbf{t}'$.

3.4 System Design

Fig. 3.2 shows an overview of our system which consists of three main blocks: 1) feature detection and matching, 2) PBF segmentation, and 3) 3D structure estimation. The second and third blocks are the main contribution of this work.

Given images I and I', the first block extracts geometric primitives, such as feature points and line segments, along with regions with homogeneous appearance. Feature points are detected and matched using scale-invariant feature transform (SIFT) [75], and line segments detected by the Line Segment Detector [39] are matched using the method

proposed by Fan et al. [32] which uses keypoint matches. From the raw line segments, vanishing points are estimated using [104]. We use only vanishing points $\mathbf{v}$ where the vanishing direction $\mathtt{K}^{-1}\mathbf{v}$ and $\mathtt{R}'\mathtt{K}'^{-1}\mathbf{v}'$ match. Then, vanishing lines are computed from each pair of vanishing points. For appearance data, we use a graph-based segmentation algorithm in [33] to extract regions with homogeneous appearance (see Fig. 3.1b for an example).

The second block uses the geometric and appearance data from above to detect corresponding 2D building facades $\{\phi_i\} \leftrightarrow \{\phi_i'\}$ using a two-step approach. In the first step, rectilinear building regions are segmented out using both line segments and homogeneous appearance regions (Sec. 3.5.1). Then, the next step detects each PBF by homography clustering using only the feature points and line segments that lie on the segmented building region (Sec. 3.5.2).

The final block estimates the 3D structure of the building by estimating the set of PBFs $\{\Phi_i\}$. The camera pose, $\mathtt{R}'$ and $\mathbf{t}'$, and the 3D positions of the feature points and line segments are first initialized. Then, the parameters of $\{\Phi_i\}$ are further optimized using geometric constraints (Sec. 3.6).

3.5 2D PBF Segmentation

In this section, we describe how a set of PBFs $\{\phi_i\}$ are segmented from a 2D image using both geometric and appearance data. When detecting multiple PBFs using a homography-based clustering approach such as in [37], two problems can occur: 1) features from non-building objects will clutter the homographies and 2) facade boundaries become more ambiguous when the building is relatively far away compared to the baseline distance. These problems cannot be solved by merely adjusting the parameters of the clustering technique. Here, we propose a two-step approach, i.e., the building region segmentation step and the planar facade extraction step (Boxes 1&2 in Fig. 3.2), where

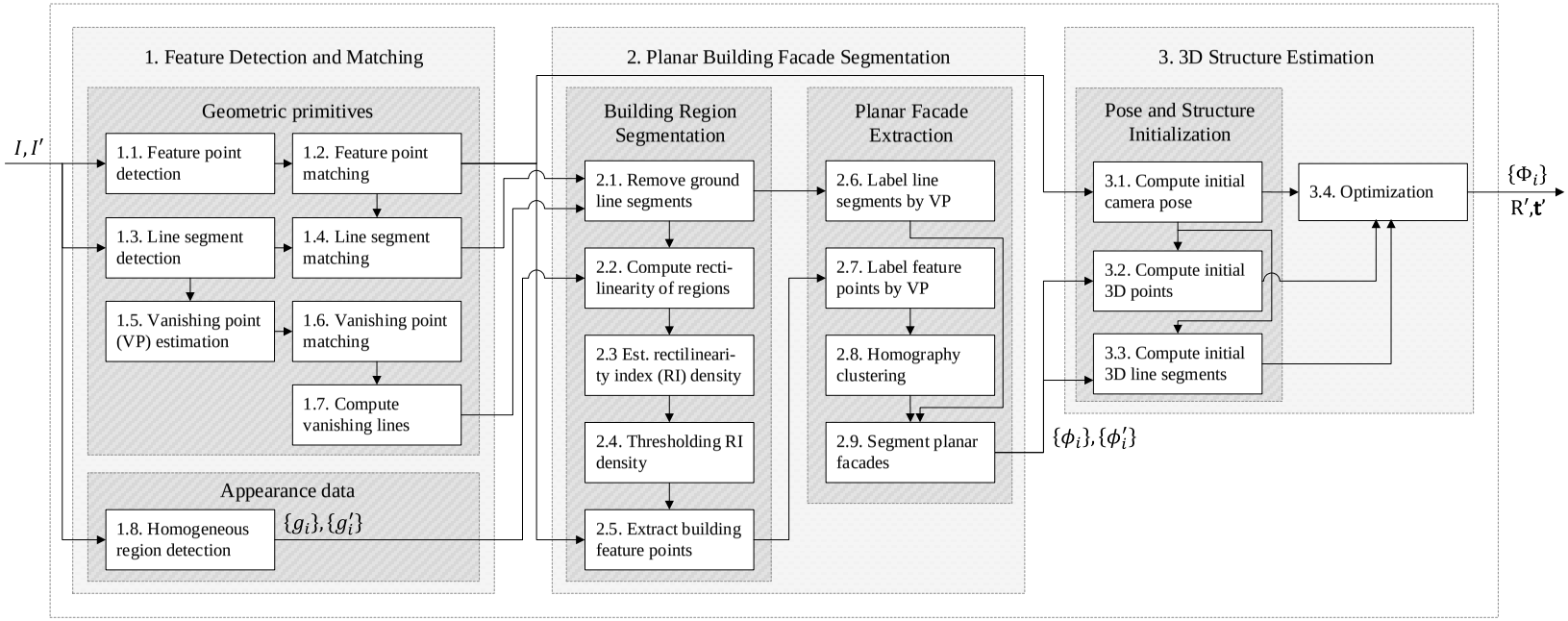


Figure 3.2: System diagram

the above two issues are addressed step by step. The first step extracts the target building region to suppress the non-building objects by combining geometric and appearance data. In the second step, to avoid ambiguous boundaries, homography clustering is applied to each group of feature points with the same horizontal vanishing direction.

3.5.1 Building Region Segmentation

Let $\{g_i\}_{i=1}^{n_g}$ be the set of n_g homogeneous pixel regions segmented from I using the algorithm in [33]. Since buildings are usually characterized by their rectilinear structure, we use line segments to determine which region g_i belongs to a PBF. We use the following steps (Boxes 2.1-2.5 in Fig. 3.2) to segment the building region.

3.5.1.1 Remove Ground Line Segments

When an image is taken from a ground robot, most of the line segments below the horizontal vanishing line belong to the ground. Although this is not valid for near objects, in many cases, this is true for line segments on a distant building. Suppose a horizontal vanishing point $\mathbf{v}_i = [x_i, y_i, 1]^\mathsf{T}$ is counterclockwise from another horizontal vanishing point $\mathbf{v}_j = [x_j, y_j, 1]^\mathsf{T}$ with respect to the zenith vanishing point $\mathbf{v}_z = [x_z, y_z, 1]^\mathsf{T}$, i.e., $\left|\begin{smallmatrix} x_i - x_z & x_j - x_z \\ y_i - y_z & y_j - y_z \end{smallmatrix}\right| < 0$. Let $\mathbf{w}$ denote the vanishing line computed by $\mathbf{v}_i \times \mathbf{v}_j$. Then, we can remove line segments $(\mathbf{e}_1, \mathbf{e}_2)$ that satisfy the condition:

$$\mathbf{e}_1^\mathsf{T}\mathbf{w} > 0 \vee \mathbf{e}_2^\mathsf{T}\mathbf{w} > 0. \tag{3.3}$$

Fig. 3.1c shows an example. After removing ground line segments, we use the remaining line segments to extract rectilinear regions in the following steps.

3.5.1.2 Compute Rectilinearity of Regions

If a region g_i contains a group of line segments, it is a strong cue for g_i being a rectilinear region. To measure the rectilinearity of a region we define the rectilinearity index

(RI) as follows:

Definition 2 (RI) *Suppose b_i is the set of boundary pixel coordinates of region g_i. The* RI *of g_i is measured by finding the nearest line segments to each pixel coordinate $\mathbf{p} \in b_i$ and computing their average distance by*

$$r_i = \frac{1}{|b_i|} \sum_{\mathbf{p} \in b_i} d(\mathbf{p}, \mathbf{s}_p), \tag{3.4}$$

where $\mathbf{s}_p$ is the nearest line segment to $\mathbf{p}$, and $d(\mathbf{p}, \mathbf{s}_p)$ is the shortest distance from $\mathbf{p}$ to line segment $\mathbf{s}_p$.

A smaller RI indicates a higher rectilinearity of region g_i. Fig. 3.1d shows an example of computing the RIs. If none of the regions have an RI smaller than a certain threshold, we determine that the view does not contain a distinguishable rectilinear structure.

3.5.1.3 Estimate RI Density

After computing the RI for each region, we use kernel density estimation to compute the density of the RIs. The density of the RIs can be computed by

$$\hat{p}(r) = \frac{1}{n_g h_{\text{opt}}} \sum_{i=1}^{n_g} K\left(\frac{r - r_i}{h_{\text{opt}}}\right) \tag{3.5}$$

where K is a normalized Gaussian kernel

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}, \tag{3.6}$$

and h_{opt} is the optimum kernel bandwidth estimated using leave-one-out cross-validation. The example in Fig. 3.3 shows a typical shape of the density estimate when the image contains a rectilinear building structure. A peak exists where the RI is small.

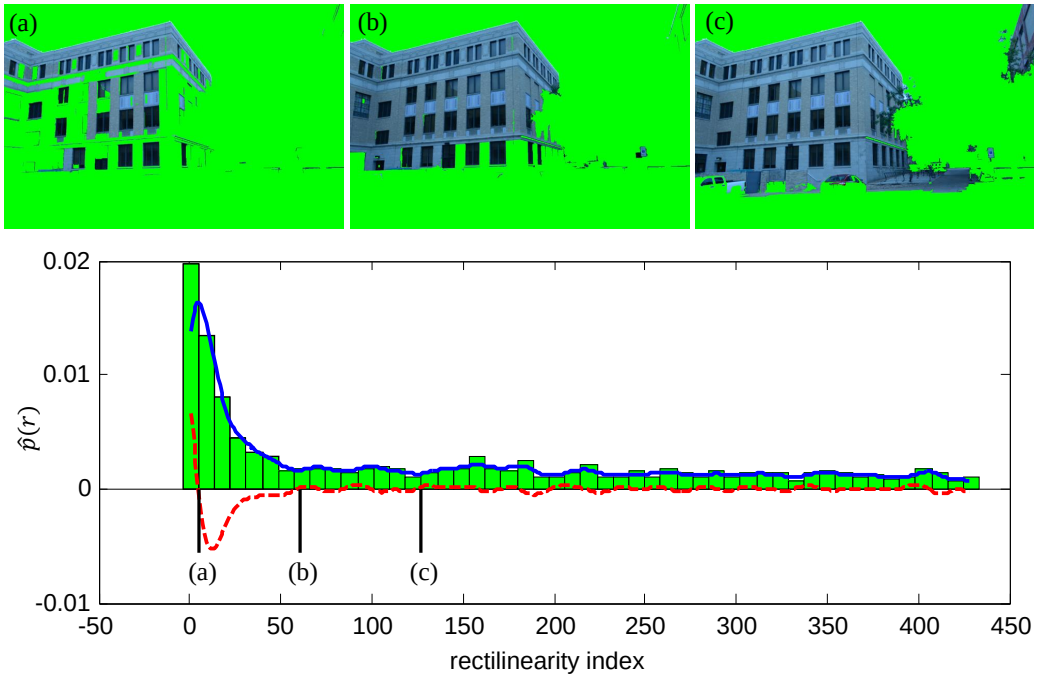


Figure 3.3: Thresholding RI density. The solid (blue) line is the estimated RI density, and the dotted (red) line is its derivative. Segmentation results are shown for (a) below selected threshold, (b) selected threshold, and (c) above selected threshold.

3.5.1.4 Thresholding RI Density

To extract regions g that belong to a PBF, we segment the peak of the RI density using a threshold. Instead of using a fixed threshold, we want to find the rectilinear regions based on the estimated distribution. Hence, we set the second zero-crossing of the first derivative of the kernel density estimate

$$\hat{p}'(r) = \frac{1}{n_g h_{\text{opt}}} \sum_{i=1}^{n_g} K'\left(\frac{r - r_i}{h_{\text{opt}}}\right) \tag{3.7}$$

as our threshold. An example result is shown in Fig. 3.3b.

3.5.1.5 Extract Building Feature Points

After the building region is segmented, we extract the feature points that lie on the building region.

3.5.2 Planar Facade Extraction

In the second step, our goal is to detect a set of corresponding PBFs $\{\phi_i\} \leftrightarrow \{\phi_i'\}$ by clustering feature points and line segments using homographies. Since a PBF should contain only one horizontal vanishing direction, the horizontal vanishing direction is an important cue to separate PBFs. Instead of applying J-linkage to the entire feature points that would result in ambiguous boundary problems, we apply J-linkage to each group of features that have the same horizontal vanishing direction. We use the following steps (Box 2.6-2.9 in Fig. 3.2) to separate PBFs.

3.5.2.1 Label Feature Points

We label each feature point $\mathbf{x}$ with a horizontal vanishing direction. Let $\mathbf{s}^h = (\mathbf{e}_1^h, \mathbf{e}_2^h)$ be a line segment associated to a horizontal vanishing point and $\mathbf{l}^h = \mathbf{e}_1^h \times \mathbf{e}_2^h$, where $\mathbf{e}_1^h$ is counterclockwise from $\mathbf{e}_2^h$ with respect to the zenith vanishing point $\mathbf{v}_z$. For each $\mathbf{s}^h$, we compute two vertical lines $\mathbf{l}_1^v = \mathbf{v}_z \times \mathbf{e}_1^h$ and $\mathbf{l}_2^v = \mathbf{v}_z \times \mathbf{e}_2^h$. Feature point $\mathbf{x}$ is then assigned with the vanishing direction of the nearest $\mathbf{s}^h$ that satisfies

$$\mathbf{x}^\mathsf{T}\mathbf{l}^h < 0 \wedge (\mathbf{x}^\mathsf{T}\mathbf{l}_1^v)(\mathbf{x}^\mathsf{T}\mathbf{l}_2^v) < 0. \tag{3.8}$$

Fig. 3.4a shows an example after feature points are labeled with their horizontal plane direction.

Figure 3.4: Extracting PBFs. (a) Feature points associated to vanishing direction of neighboring (color-coded) horizontal line segments. (b) Vertical line segments associated to vanishing direction of neighboring (color-coded) horizontal line segments. (c) Homography clustering applied to feature points with same vanishing direction. (d) Convex hull for each homography cluster.

3.5.2.2 Label Line Segments

We also label each vertical line segment $\mathbf{s}^v = (\mathbf{e}_1^v, \mathbf{e}_2^v)$ with a horizontal vanishing direction. As in the previous step, for each horizontal line segment $\mathbf{s}^h$, we compute lines $\mathbf{l}^h$, $\mathbf{l}_1^v$, and $\mathbf{l}_2^v$. We find the line segment $\mathbf{s}^h$ that has the smallest $\min(d(\mathbf{e}_1^v, \mathbf{l}_1^v) + d(\mathbf{e}_2^v, \mathbf{l}_1^v), d(\mathbf{e}_1^v, \mathbf{l}_2^v) + d(\mathbf{e}_2^v, \mathbf{l}_2^v))$ that satisfies

$$\mathbf{e}_1^{v\top}\mathbf{l}^h < 0 \wedge \mathbf{e}_2^{v\top}\mathbf{l}^h < 0. \tag{3.9}$$

Then, $\mathbf{s}^v$ is assigned with the vanishing direction of $\mathbf{s}^h$. See Fig. 3.4b for example.

3.5.2.3 Homography Clustering

After feature points are associated with a horizontal vanishing direction, we apply homography clustering using J-linkage on the feature points with the same horizontal vanishing direction. This avoids ambiguities between facade boundaries and separates the facades that are in different horizontal directions. Fig. 3.4c shows an example of the clustering on one horizontal vanishing direction.

3.5.2.4 Segment Planar Facades

Finally, a convex hull is computed for the set of feature points that belong to the same cluster obtained in the previous step. See Fig. 3.4d for illustration. The pixel region in this convex hull is set to G. The feature points and line segments that lie in G are denoted as X and S, respectively.

Thus, we have the three components X, S, and G for a PBF ϕ in an image I. The algorithm for the PBF segmentation and its expected running time are summarized below.

Algorithm 1 Planar Building Facade Segmentation

Input: $\{\mathbf{x}_i\}_{i=1}^{n_x}$, $\{\mathbf{s}_i\}_{i=1}^{n_s}$, $\{\mathbf{w}_i\}_{i=1}^{n_w}$, $\{g_i\}_{i=1}^{n_g}$
Output: $\{\phi_i\}_{i=1}^{n_f}$

1: remove ground line segments with $\{\mathbf{w}_i\}$	$\triangleright\ O(n_w n_s)$
2: compute region RI for all g	$\triangleright\ O(n_g n_s)$
3: estimate RI density using cross-validation	$\triangleright\ O(n_g^2)$
4: label $\mathbf{x}$ by horizontal direction	$\triangleright\ O(n_x)$
5: label vertical $\mathbf{s}$ by horizontal direction	$\triangleright\ O(n_s)$
6: $\{X_i\} \leftarrow$ cluster feature points	$\triangleright\ O(n_x^3)$
7: $\{\phi_i\} \leftarrow$ segment planar facade regions	$\triangleright\ O(n_x^2)$

Since n_w is usually small, it can be considered as a constant. Hence, the overall computational complexity of Algorithm 1 is $O(n_x^3 + n_s n_g + n_g^2)$.

3.6 3D PBF Estimation

After a set of corresponding PBFs $\{\phi_i\} \leftrightarrow \{\phi_i'\}$ is extracted from two views, we simultaneously estimate the camera pose and 3D PBF positions. The camera pose, $\mathtt{R}'$ and $\mathbf{t}'$, is initialized using the standard steps in [46]; the fundamental matrix $\mathtt{F}$ is fit to keypoint matches using RANSAC; the camera pose $\mathtt{R}'$ and $\mathbf{t}'$ are obtained by decomposing the essential matrix $\mathtt{E} = \mathtt{K}'^{\mathsf{T}}\mathtt{F}\mathtt{K}$. The 3D PBF positions $\{\Phi_i\}$ are initialized by triangulating the set of feature points and the set of line segment endpoints. The plane is initialized by fitting Π to the triangulated feature points and line segment endpoints.

To refine the initial estimates using geometric constraints, our goal is to solve

$$\underset{\mathtt{R}',\mathbf{t}',\{\Phi_i\}}{\arg\min}\, J_{\text{rep}} + J_{\text{ori}} + J_{\text{cop}}, \tag{3.10}$$

where J_{rep}, J_{ori}, and J_{cop} are the cost terms for the reprojection error, orientation constraint, and coplanarity constraint, respectively. Each cost term J_{rep}, J_{ori}, and J_{cop} is defined in the following subsections.

3.6.1 Reprojection Error

The reprojection error J_{rep} in (3.10) is minimized when the projections of the estimated 3D points and line segments are close to their image measurements. We compute the reprojection error by

$$J_{\text{rep}} = \frac{J_{\text{rep}}^x}{\lambda_{\text{rep}}^x} + \frac{J_{\text{rep}}^l}{\lambda_{\text{rep}}^l} + \frac{J_{\text{rep}}^e}{\lambda_{\text{rep}}^e}, \tag{3.11}$$

where J_{rep}^x, J_{rep}^l, and J_{rep}^e are the reprojection errors for feature points, lines, and line-segment endpoints, respectively. The λ values for each cost term will be described at the end of the section.

To compute J_{rep}^x, we use the standard reprojection error in [46]. Let X be the set of

feature points. The reprojection error over X is defined by

$$J^x_{\text{rep}} = \sum_{\mathbf{x}\in X} d(\mathbf{x},\hat{\mathbf{x}})^2 + d(\mathbf{x}',\hat{\mathbf{x}}')^2, \tag{3.12}$$

where $d(\mathbf{x},\hat{\mathbf{x}})$ is the distance between the observation $\mathbf{x}$ and estimation $\hat{\mathbf{x}}$ in image coordinates.

We use the line reprojection error in [102] to compute J^l_{rep}. The cost function for lines is

$$J^l_{\text{rep}} = \frac{1}{4}\sum_{\mathbf{s}\in S} \left(d(\hat{\mathbf{e}}_1,\mathbf{p}_1)d(\mathbf{p}_1,\mathbf{p}_3)\right)^2 + \left(d(\hat{\mathbf{e}}_2,\mathbf{p}_2)d(\mathbf{p}_2,\mathbf{p}_3)\right)^2 + \left(d(\hat{\mathbf{e}}'_1,\mathbf{p}'_1)d(\mathbf{p}'_1,\mathbf{p}'_3)\right)^2 + \left(d(\hat{\mathbf{e}}'_2,\mathbf{p}'_2)d(\mathbf{p}'_2,\mathbf{p}'_3)\right)^2, \tag{3.13}$$

where $\mathbf{p}_1$ and $\mathbf{p}_2$ are the projections of the estimated endpoints $\hat{\mathbf{e}}_1$ and $\hat{\mathbf{e}}_2$ on the line $\mathbf{e}_1 \times \mathbf{e}_2$. The point $\mathbf{p}_3$ is the intersection of the estimated line $\hat{\mathbf{e}}_1 \times \hat{\mathbf{e}}_2$ and the observed line $\mathbf{e}_1 \times \mathbf{e}_2$.

If the reprojection error for lines is used alone, the estimated line segments would "grow" or "shrink" during the estimation process. To avoid this, we also apply the reprojection error to line-segment endpoints by

$$J^e_{\text{rep}} = \sum_{\mathbf{s}\in S} d(\mathbf{e}_1,\hat{\mathbf{e}}_1)^2 + d(\mathbf{e}'_1,\hat{\mathbf{e}}'_1)^2 + d(\mathbf{e}_2,\hat{\mathbf{e}}_2)^2 + d(\mathbf{e}'_2,\hat{\mathbf{e}}'_2)^2. \tag{3.14}$$

3.6.2 Orientation Constraint

The 3D orientation of each line segment and plane can be computed since their associated vanishing direction is known. To constrain the overall orientation of the building

structure, we minimize the following cost function

$$J_{\text{ori}} = \frac{J^{v}_{\text{ori}}}{\lambda^{v}_{\text{ori}}} + \frac{J^{n}_{\text{ori}}}{\lambda^{n}_{\text{ori}}} + \frac{J^{p}_{\text{ori}}}{\lambda^{p}_{\text{ori}}}, \tag{3.15}$$

where J^{v}_{ori} and J^{n}_{ori} constrain the line segment orientation and J^{p}_{ori} constrains the plane orientation.

The first term in (3.15) makes each line segment $\mathbf{S}$ parallel to its associated vanishing direction by

$$J^{v}_{\text{ori}} = \sum_{i} \sum_{\mathbf{S} \in S} \|\hat{\mathbf{d}}_s \times \mathbf{d}_i\|^2, \tag{3.16}$$

where $\hat{\mathbf{d}}_s$ is the direction of the line segment $\mathbf{S}$ and $\mathbf{d}_i$ is the vanishing direction of the vanishing point $\mathbf{v}_i$. As in the line reprojection error, the above cost function represents the area between the line segment direction vector and the vanishing direction vector.

Each line segment direction $\hat{\mathbf{d}}_s$ is also enforced to be perpendicular to its associated plane normal $\hat{\mathbf{n}}$ by

$$J^{n}_{\text{ori}} = \sum_{i} \sum_{\mathbf{S} \in S} (\hat{\mathbf{d}}_s \cdot \hat{\mathbf{n}}_i)^2. \tag{3.17}$$

The above cost function prevents line segments becoming perpendicular to the plane due to the coplanarity constraint in (3.21), which is described in the next section.

Each plane normal $\hat{\mathbf{n}}_i$ is constrained so that it is perpendicular to the associated horizontal vanishing direction $\mathbf{d}_h$ and vertical vanishing direction $\mathbf{d}_z$ using

$$J^{p}_{\text{ori}} = \sum_{i} (\mathbf{d}_h \cdot \hat{\mathbf{n}}_i)^2 + (\mathbf{d}_z \cdot \hat{\mathbf{n}}_i)^2. \tag{3.18}$$

3.6.3 Coplanarity Constraint

A strong constraint to reduce the depth ambiguity of 3D points and line segments is the coplanarity constraint. The coplanarity constraint for feature points and line segments

Figure 3.5: Test image examples

is defined by

$$J_{\text{cop}} = \frac{J^x_{\text{cop}}}{\lambda^x_{\text{cop}}} + \frac{J^s_{\text{cop}}}{\lambda^s_{\text{cop}}}, \tag{3.19}$$

where J^x_{cop}, and J^s_{cop} are the cost terms for feature points and line-segment endpoints, respectively.

We compute J^x_{cop} by

$$J^x_{\text{cop}} = \sum_i \sum_{\mathbf{X} \in X_i} d_\perp(\hat{\mathbf{X}}, \hat{\Pi}_i)^2, \tag{3.20}$$

where $d_\perp(\hat{\mathbf{X}}, \hat{\Pi}_i)$ is the perpendicular distance from point $\hat{\mathbf{X}}$ to plane $\hat{\Pi}_i$.

For line segments, we enforce coplanarity by minimizing the area between the line segment and the plane by

$$J^s_{\text{cop}} = \sum_i \sum_{\mathbf{S} \in S} (\hat{\mathbf{E}} \cdot \hat{\Pi}_i)\, d(\mathbf{E}_1, \mathbf{E}_2). \tag{3.21}$$

The area between a line segment and plane Π_i becomes zero when they are coplanar.

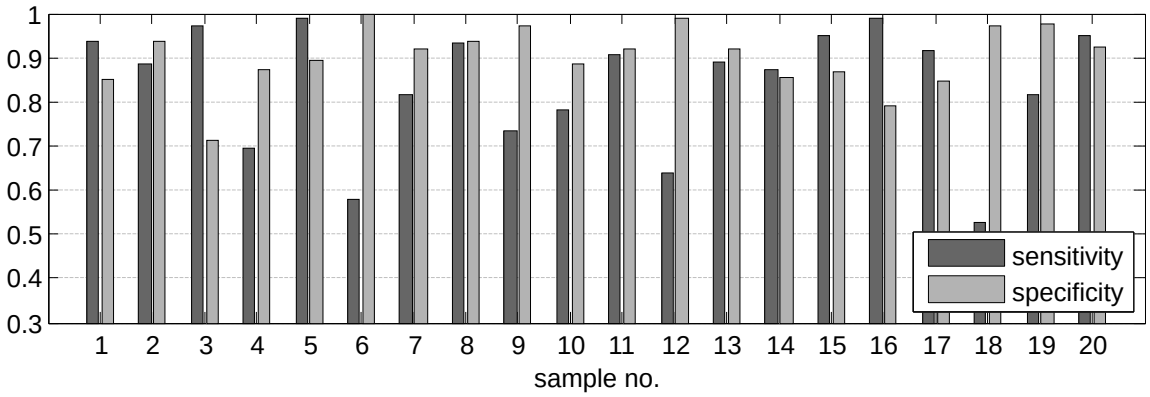


Figure 3.6: Sensitivity and specificity of PBF region detection

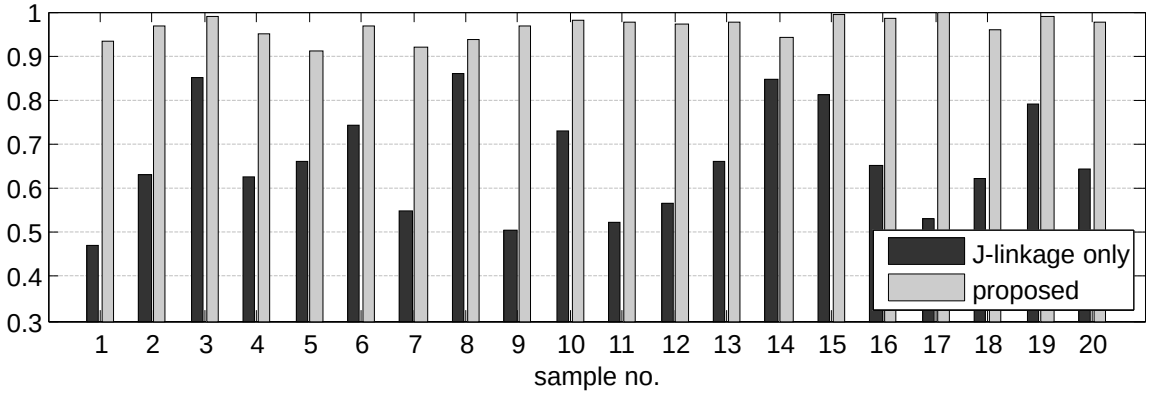


Figure 3.7: Precision comparison of PBF detection

We use the Levenberg-Marquardt algorithm to solve our non-linear optimization problem in (3.10). To balance the three cost terms in (3.10), as in [52], we set each weight λ as the initial value of its corresponding J so that the initial cost of $J_{\text{rep}} + J_{\text{ori}} + J_{\text{cop}} = 8$.

3.7 Experiments

To measure the performance of our proposed method, we have tested the method on real data. We have taken a pair of images of 20 different buildings on Texas A&M University campus (see Fig. 3.5). Two experiments have been conducted to test the performance

of our method: the PBF segmentation test and the PBF mapping test.

3.7.1 PBF Segmentation Test

We have measured the performance of each step in our PBF segmentation method described in Sec. 3.5. For the building region segmentation step (Sec. 3.5.1), we have counted the number of true-positive (TP), true-negative (TN), false-positive (FP), and false-negative (FN) feature points after the building region segmentation has been performed. The ground-truth was obtained by manually examining whether each feature point was on a building facade. Fig. 3.6 shows the sensitivity ($\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$) and specificity ($\mathrm{TN}/(\mathrm{FP}+\mathrm{TN})$) of our method.

Next, for the planar facade detection step (Sec. 3.5.2), we have computed the precision ($\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$). We have compared the precision of our proposed method with a base-line method, i.e., the J-linkage in [103] which uses J-linkage directly on the image without use of the horizontal vanishing direction information. It is worth noting that our method also uses J-linkage as a sub-step. The parameters for the J-linkage step are the same in both methods. Fig. 3.7 shows that our proposed method is always better. In fact, the average precision increase over the J-linkage method is 45.27%.

3.7.2 PBF Mapping Test

We have compared our proposed method with a base-line method which minimizes the standard reprojection error in structure estimation. Since the camera baseline distance is relatively small compared to the building distance, the camera positional error is not significant in either method. Hence, we present two measures, (1) the PBF depth error and (2) the PBF angular error, to illustrate the performance of our proposed method.

First, we measure the depth error of the mapped building. Each image has been taken so that the principal axis passes through the intersection of the two primary PBFs, i.e., the building corner. We set the baseline distance to be 1/60 of the distance to the building

corner. Suppose $\mathbf{l}_k$ is the intersection of the estimated 3D planes $\hat{\Pi}_i$ and $\hat{\Pi}_j$ of PBFs, and let Π_z be the plane passing through the camera center $\mathbf{C}$ with the normal vector $\mathbf{d}_x \times \mathbf{d}_z$, where $\mathbf{d}_x = [1,0,0]^\mathsf{T}$ and $\mathbf{d}_z = \mathtt{K}^{-1}\mathbf{v}_z$. Then, we can measure the depth error of a building corner $\mathbf{l}_k$ by

$$\varepsilon_{\text{depth}} = \left| \bar{d} - \frac{d_\perp(\hat{\mathbf{Q}}_1, \Pi_z) + d_\perp(\hat{\mathbf{Q}}_2, \Pi_z)}{2} \right| \tag{3.22}$$

where the ground-truth depth $\bar{d}$ is measured from a top-down view aerial map and a precision laser ranger (BOSCH GLR225) with 1 mm accuracy. If P is the set of feature points $X_i \cup X_j \cup E_i \cup E_j$ of Φ_i and Φ_j, then the points $\hat{\mathbf{Q}}_1$ and $\hat{\mathbf{Q}}_2$ are the intersections of $\mathbf{l}_k$ with the plane parallel to Π_z which passes the points

$$\max_{\mathbf{P}\in P}(\mathbf{P}^\mathsf{T}\mathbf{d}_z)\cdot\mathbf{d}_z \;\text{ and }\; \min_{\mathbf{P}\in P}(\mathbf{P}^\mathsf{T}\mathbf{d}_z)\cdot\mathbf{d}_z, \tag{3.23}$$

respectively. Fig. 3.8a shows that the overall depth error decreases after the refinement. In fact, the average depth error is reduced from 22.39 to 8.95, a 60% reduction.

Next, to measure how well the building structure is recovered, we measure the angular error of PBFs by

$$\varepsilon_{\text{angle}} = \left| \bar{\theta}_a - \hat{\theta}_a \right|, \tag{3.24}$$

where $\bar{\theta}_a$ is the ground-truth angle between the two primary PBFs measured from a top-down view aerial map and the same high precision laser ranger, and $\hat{\theta}_a$ is the estimated angle which is computed by

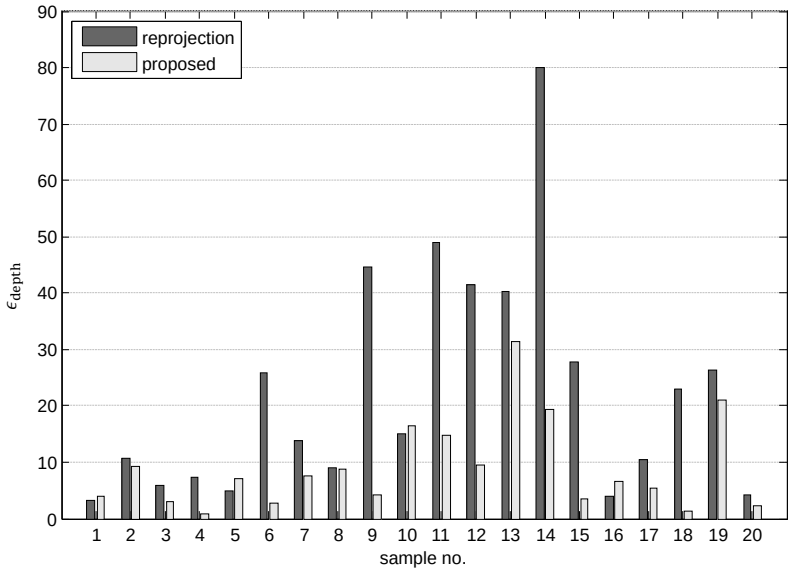
$$\hat{\theta}_a = \arccos\left(\frac{\hat{\mathbf{n}}_i^\mathsf{T}\hat{\mathbf{n}}_j}{\|\hat{\mathbf{n}}_i\|\|\hat{\mathbf{n}}_j\|} \right). \tag{3.25}$$

where $\mathbf{n}_i$ and $\mathbf{n}_j$ are plane normal vectors for $\hat{\Pi}_i$ and $\hat{\Pi}_j$, respectively. Fig. 3.8b shows that the pairwise angular errors are reduced using our proposed method. The average angular

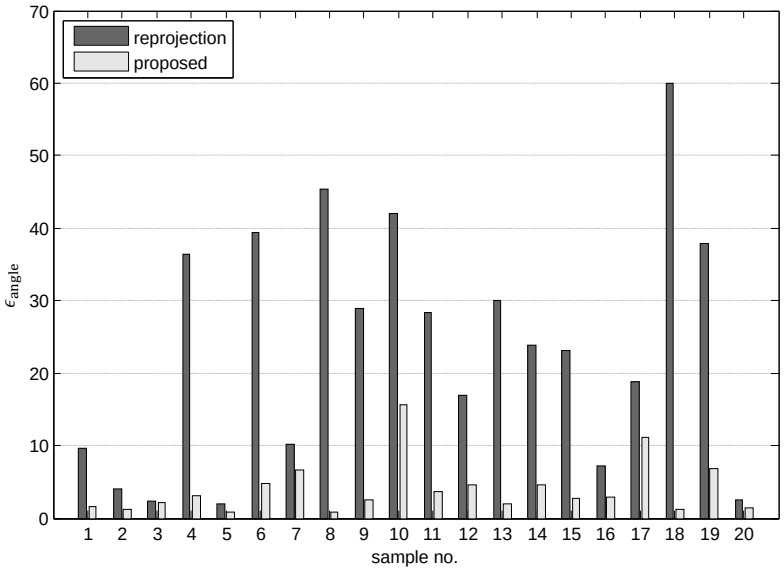
error reduction is 82.82%.

3.8 Conclusions

We reported a novel algorithm for PBF segmentation and 3D estimation. We proposed a new RI to distinguish building regions based on homogeneous region detection results. We combined J-linkage with vanishing point constraints to detect individual PBFs in a 2D scene. Then, we combined the reprojection errors, orientation constraints, and coplanarity constraints as cost functions in an optimization process to improve the 3D estimation of the building structure. We tested our method in physical experiments. We compared our algorithm with state-of-the-art J-linkage-based facade detection for segmentation and reprojection error-based 3D mapping. The results showed that our method increases precision in segmentation and reduces depth and angular errors in 3D estimation. Specially, our method reduced the angular error of the reprojection error-based 3D mapping by an average of 82.82%.



(a) PBF depth error



(b) PBF angular error

Figure 3.8: Comparison of the reprojection error-based 3D estimation and that of the proposed method

4. MAP FUSION FOR MONOCULAR VISION AND LIDAR INPUTS

4.1 Introduction

Knowing the accurate location of an indoor robot or a mobile device user is of great importance in the era of mobile computing. It is an enabling technology for many location-aware applications. Since GPS signals are often challenged in indoor or urban environments, many researchers focus on developing localization and mapping algorithms using onboard sensors. However, power, costs and physical limits of sensors often hinder the development. For example, the size of small mobile devices does not allow wide baseline distance for stereo cameras, and a monocular camera often suffers from scale drift. Also, regular cameras are always sensitive to lighting and feature distributions in the environment despite their low cost. A lidar is often considered as the most reliable mapping sensor, but it is too power hungry, bulky, and expensive for many mobile robots or devices. RGB-D cameras can be small but unreliable when strong sun light exists. No single sensor can offer an ideal solution to application needs.

One possible solution is to decouple the process of simultaneous localization and mapping (SLAM). After all, humans almost always use maps built by others. High accuracy 2D lidar mapping, which often consists of point clouds (see Fig. 4.1(b)), can be established by a scanning robot to provide prior knowledge for other cellphone users or robots only equipped with a low cost camera. A monocular vision-based SLAM algorithm often generates low-quality maps as illustrated in Fig. 4.1(a) due to scale drift. The output may even be disconnected maps because of the uneven distribution of features. If we can rectify the low quality map to the lidar-level accuracy (see Fig. 4.1(c)), mobile device users and robots can enjoy high localization accuracy by just using low cost cellphone cameras, which has a huge advantage in applications.

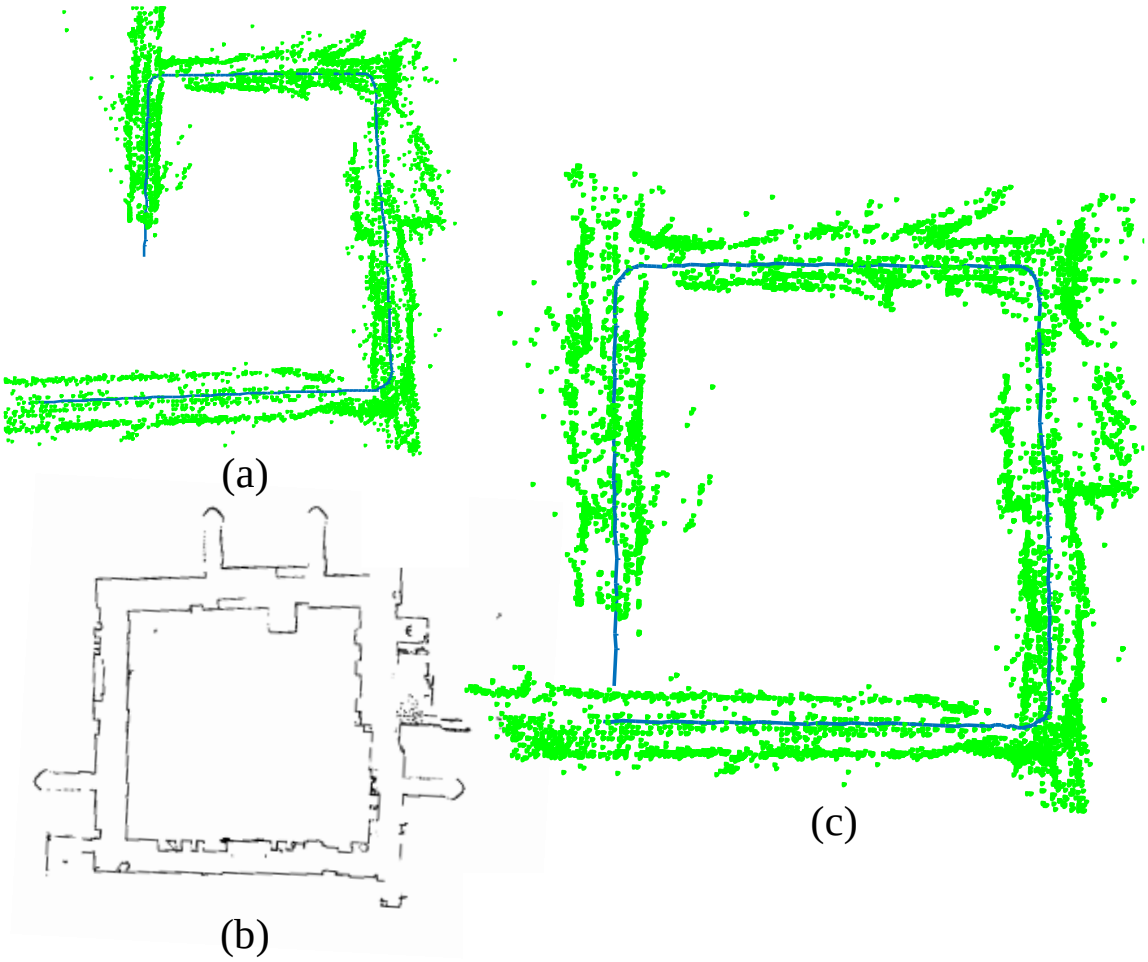


Figure 4.1: Map fusion. (a) A 3D monocular vision-based map with scale drift, (b) a 2D lidar map, and (c) the rectified map of (a) using our proposed method.

Since 2D lidar maps consist of 2D point clouds in the world coordinate system, and image data generate 2D feature points in an image coordinate system, they cannot be directly registered to each other. This requires us to fuse the spatial knowledge (i.e., landmarks of maps) asynchronously captured from different perspectives and platforms by heterogeneous sensors, which presents a new research problem. Building on high-level landmarks such as vertical planes that can be identified in both sensing modalities, we propose a two-step optimization approach to address this high dimension map fusion problem. We have implemented the system and tested it with real world data. The experimental results show that our algorithm significantly improves localization and mapping quality of visual SLAM by as much as 71.1% in mapping accuracy.

The rest of the chapter is organized as follows. After reviewing related work in Section 4.2, we define our map fusion/rectification problem in Section 4.3. We present our map fusion algorithm in Section 4.4. Experimental results are presented in Section 4.5 before we conclude the work in Section 4.7.

4.2 Related Work

The map fusion problem can be viewed as a post processing step in visual SLAM, which is a fast developing area. More specifically, this work builds upon recent works on visual SLAM including optimization techniques, different sensor and feature configurations, and collaborative map merging for multiple robots.

A SLAM algorithm simultaneously estimates camera/robot pose and landmark positions, which is a fundamental problem in robotics and computer vision. A full fledged SLAM framework consists of subproblems such as tracking, mapping, and loop closing. For efficiency, recent systems [57, 97, 93, 29, 85] implement a front-end and back-end system running in parallel, where the front-end performs tracking in real-time while the back-end refines both the resulting trajectory and the map, and closes loops occasionally.

To increase the speed of the back-end, different optimization techniques are proposed. Recent optimization methods take advantage of sparse matrix structures in the SLAM problem. In [62], an implementation of a hypergraph-based sparse non-linear optimization framework called g2o is presented. An optimization method known as iSAM2 [51] uses a Bayes tree to incrementally update the sparse matrix for an on-line SLAM system. Building on the g2o framework to exploit sparse matrix computation, our proposed map fusion can be viewed as a back-end system which runs asynchronously from a front-end tracking and mapping process.

SLAM can be performed with different exteroceptive sensors or their combinations including regular cameras, lidars, and RGB-D cameras. Regular camera-based SLAM is also referred as visual SLAM, where two main approaches exist: filtering and structure-from-motion (SFM) using an optimization approach. For filtering, the extended Kalman filters (EKF) or its variants have been used extensively [26, 13, 91, 12], whereas the dominating method for SFM is bundle adjustment (BA) [92, 60, 31]. The BA method is an optimization-based method which minimizes the reprojection error over multiple image frames. Strasdat et al. [98] have compared EKF against BA and pointed out that unless in high uncertainty situations, BA has a better performance and accuracy than EKF. Our proposed method uses a batch optimization to estimate the fused map. However, monocular vision-based SLAM is the most difficult problem due to the infamous scale drifting issue. Stereo vision can relieve the issue but is often limited by the baseline and power constraints of small devices.

Depth sensors such as RGB-D cameras or lidars [47, 28, 86] can help address the scale drifting issue in monocular vision. While RGB-D cameras are widely used for indoor environments, lidars are still the most favorable sensor due to longer sensing range, wide field of view, and robustness to sun light, making it suitable for both indoor and outdoor [42]. Kohlbrecher et al. [59] registers 2D lidar scans to a 2D occupancy grid map based on an

image registration technique. Zhang and Singh [106] use a 2-axis lidar and develop a real-time 3D mapping method with low-drift by separating the odometry and mapping task. In these methods, the lidar map is accurate enough so that post-processing is not required.

Combining both vision and lidar inputs to utilize the benefit of each sensor has been proposed as well. Newman et al. [87] use a 3D lidar to map buildings and use vision to detect loop closure. Zhang and Singh [107] use vision to handle rapid motion while the lidar warrants low-drift and robustness to lighting changes. However, these methods require that the vision and lidar data are synchronized and captured by the same robot. The resulting map is also limited to the hosting robot for the localization usage. Our method is intended to fuse the vision and lidar maps acquired by different robots and generate maps that can help different cameras or mobile device users.

Map merging has been studied for multi-robot systems in indoor environments. In [22], Dedeoglu and Sukhatme combines topological maps using landmarks detected by sonar, vision, and laser. Fox et al. [38] merges lidar maps with robots actively detecting each other to estimate their relative positions. Carpin [11] proposed a map merging technique based on Hough transforms to merge occupancy grid maps. Baudouin et al. [7] propose a method that merges robot paths with different scales generated by multi-modal vision sensors such as perspective, fish-eye, or omnidirectional cameras. These prior works shed light on our problem but they focus on maps building with the same or similar kind of sensors.

In a 2D lidar map, the most visible features are wall planes. Therefore, we adopt a visual SLAM method which has landmarks consisting of planes in addition to point features. Many studies make use of the fact that urban environments have planar structure [9]. Lu and Song introduce a multilayer feature graph (MFG) [78] for a visual SLAM approach using heterogeneous landmarks including planes. Taguchi et al. [100] use an RGB-D camera and propose a method that uses combinations of primitives, i.e., points and planes, for

faster pose estimation. More recently, Salas-Moreno et al. [94] have proposed a dense planar SLAM using an RGB-D camera. For our fusion method, we use the output from MFG since it provides planar structures with only vision.

4.3 Problem Definition

4.3.1 Background and Notations

Because a 2D lidar map cannot be directly registered to image feature points or lines, we plan to use high-level landmarks such as planes for the registration between lidar map and image features. One immediate choice is the 3D MFG [78], which is a sparse 3D feature structure built by examining geometric relationship among heterogeneous low-level features. For completeness, we briefly introduce it here. Let us denote the 3D MFG map by G. The map G is a graph where the nodes represent heterogeneous 3D landmarks and the edges represent the relationship between the landmarks. The landmarks include key points, line segments, lines, vanishing points, and planes while the edges represent constraints such as parallelism, collinearity, and coplanarity. (See Fig. 4.2 for illustration.) The map G is augmented with new landmarks at each key frame and refined by a local BA scheme in [78].

To describe the landmarks in G, we denote the image measurement and estimate of landmark $\mathbf{X}$ by $\tilde{\mathbf{x}}$ and $\hat{\mathbf{X}}$, respectively. Let I_j be the j-th key frame, and let the camera pose associated with I_j be denoted as $\mathbf{m}_j$. The measurement of a key point $\mathbf{P}_i \in \mathbb{R}^3$ in I_j is denoted by $\tilde{\mathbf{p}}_{ij} \in \mathbb{R}^2$, and the projection of $\hat{\mathbf{P}}_i$ in I_j is denoted by $\hat{\mathbf{p}}_{ij}$. Let $\tilde{\mathbf{s}}_{ijk}$ be the k-th observed line segment in I_j that should be collinear to the line $\mathbf{L}_i$. Let $\Pi = (\mathbf{n}, d)$ be a plane, where $\mathbf{n} \in \mathbb{R}^3$ is the plane normal, and $d/||\mathbf{n}||$ is the distance between the origin and Π.

It is worth noting that the choice of visual SLAM in our method is not limited to the MFG. In fact, any visual SLAM method that can provide plane estimation can use our

approach since we only use planes to register image landmarks with the 2D lidar map. Denote the 2D lidar map by L. In this work we use the 2D lidar map generated with the method in [59]. Similarly, any 2D lidar-based SLAM method works with our approach. The coordinate system/frame of L and G are denoted by $\{L\}$ and $\{G\}$, respectively.

4.3.2 Assumptions and Problem Definition

In our study, we assume the following:

A.1 An initial MFG map G and camera trajectory $\{\mathbf{m}_j\}$ is available from visual SLAM.

A.2 A lidar map L is available from a depth-based SLAM.

A.3 The set of corresponding wall corners between G and L are available. This can be achieved by 1) converting the Euclidean map into a topological map based on corners and connectivities, 2) aligning the maps using topological graph matching techniques [15], and 3) extract corners from the identified correspondence.

Our problem is defined as

Problem 2 *Given G, $\{\mathbf{m}_j\}$, $\{\tilde{\mathbf{p}}_{ij}\}$, $\{\tilde{\mathbf{s}}_{ijk}\}$, and L, estimate the rectified camera trajectory $\{\hat{\mathbf{m}}_j\}$ and MFG map $\hat{G}$.*

4.4 Map Fusion Algorithm

To solve the map fusion problem, we first extract vertical planes from 2D lidar map L and 3D MFG map G. The vertical planes will then be used as the reference to rectify G using a two-step optimization framework along with a penalty-barrier type method. We begin with vertical plane extraction.

4.4.1 Vertical Plane Extraction

In a 2D lidar map L, let $\mathbf{l}$ be a line that passes through the 2D point cloud that corresponds to a planar wall surface. Then, the intersection of two joining lines is defined as a

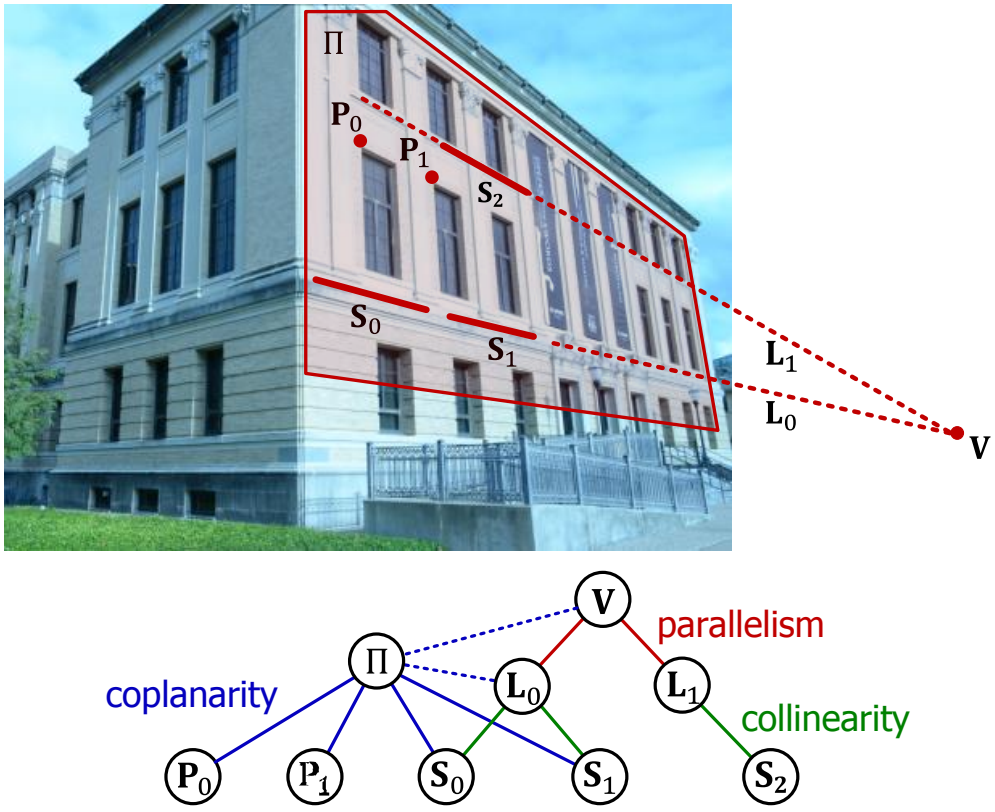


Figure 4.2: A glance at the MFG [78]. The top figure illustrates parallel, collinear, and coplanar features extracted from a scene. A subgraph of the MFG representing the relationship between extracted features is shown in the bottom.

wall corner in L. We denote the i-th wall corner in L as $\mathbf{c}_i^l$.

In a 3D map G, a wall corner coordinate is computed from the intersection of two joining vertical planes and a plane that is parallel to the ground plane. The vertical planes can be easily distinguished from other planes by verifying whether its normal is perpendicular to the vertical vanishing point. Without loss of generality, we assume that the x-z plane is parallel to the ground plane. Let the two joining vertical planes be $\mathbf{\Pi}_a = [a_1, a_2, a_3, a_4]^\mathsf{T}$ and $\mathbf{\Pi}_b = [b_1, b_2, b_3, b_4]^\mathsf{T}$, and let $\mathbf{\Pi}_g = [0, 1, 0, 0]^\mathsf{T}$ be the plane parallel to the ground plane. Then the wall corner $\mathbf{c} = [c_1, c_2, c_3]^\mathsf{T}$ in $\{G\}$ is computed by

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ 0 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} -a_4 \\ -b_4 \\ 0 \end{bmatrix}. \tag{4.1}$$

The 2D coordinates of the wall corner $\mathbf{c}$ lying on the horizontal plane $\mathbf{\Pi}_g$ is defined as $\mathbf{c}_j^g = [c_1, c_3]^\mathsf{T}$, where j is the index.

Given the set of 2D-2D corresponding wall corners $\{\mathbf{c}_i^l \leftrightarrow \mathbf{c}_j^g\}$ (see assumptions A.3 in Section 4.3.2), a non-reflective similarity transformation $\hat{T}$ is estimated by the least-squares method to provide an initial alignment frame which maps planes in world frame of L to frame of G. We will rectify G in this frame. Let $\mathbf{c}^r$ be the reference wall corner computed by

$$\mathbf{c}_i^r = \hat{T}\mathbf{c}_i^l. \tag{4.2}$$

Let $\mathbf{l}^r = [l_1, l_2, l_3]^\mathsf{T}$ be the line that passes two reference wall corners $\mathbf{c}_a^r$ and $\mathbf{c}_b^r$, i.e., $\mathbf{l}^r = \bar{\mathbf{c}}_a^r \times \bar{\mathbf{c}}_b^r$, where the $\bar{}$ symbol represents homogeneous coordinates. Then we can obtain a reference vertical plane in $\{G\}$ by

$$\mathbf{\Pi}^r = [l_1, 0, l_2, l_3]^\mathsf{T}. \tag{4.3}$$

In the next step, we use the set of vertical planes $\{\Pi_i^r\}$ to guide the estimation process for $\hat{G}$.

4.4.2 Rectification of MFG

Given the set of vertical planes $\{\Pi_i^r\}$, our goal is to rectify G by forcing the coplanar features to lie on the planes Π_i^r to which they belong. This can be viewed as constrained optimization with global bundle adjustment (GBA). The equality-constrained optimization problem that we want to solve is

$$\min_{\substack{\{\mathbf{m}_{i_m}\},\{\mathbf{P}_{i_p}\} \\ \{\mathbf{L}_{i_l}\},\{\mathbf{V}_{i_v}\},\{\Pi_{i_\pi}\}}} J_{im} = J_p + \lambda_l J_l + \lambda_v J_v \tag{4.4}$$

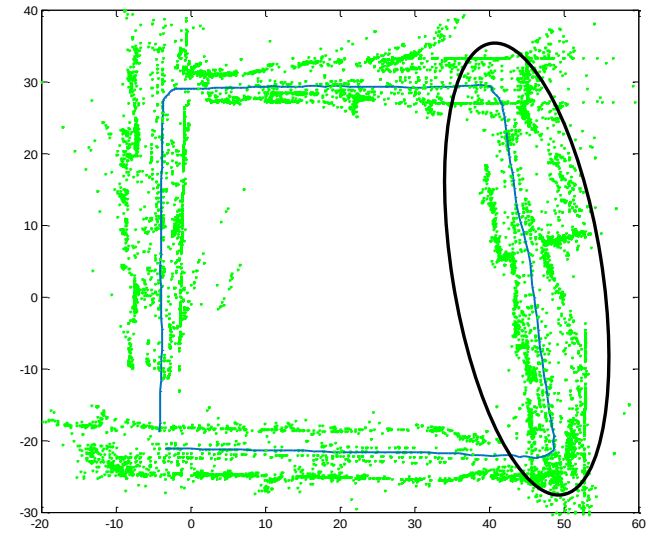
subject to,

$$\sum_i \sum_j d(\mathbf{P}_{ij}, \Pi_i^r) = 0, \tag{4.5}$$

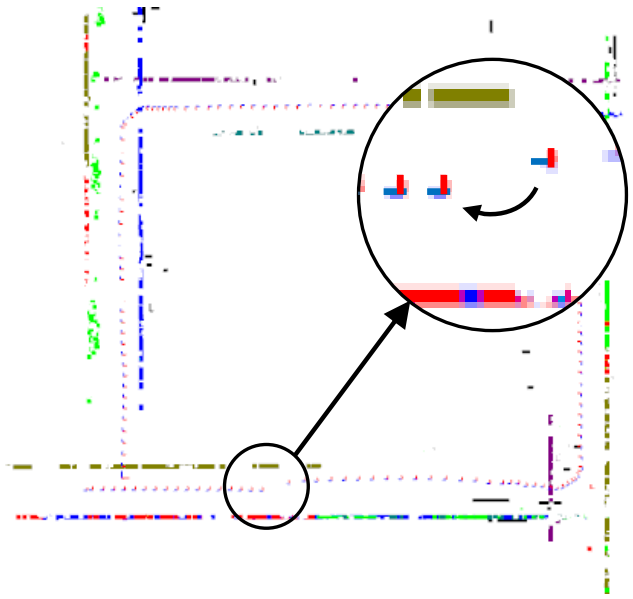
$$\sum_i \sum_j \sum_k d(\mathbf{A}_{ijk}, \Pi_i^r) + d(\mathbf{B}_{ijk}, \Pi_i^r) = 0, \tag{4.6}$$

where J_p, J_l, and J_v are the image domain cost for key points, lines, and vanishing points, respectively. The cost functions are the robustified version of the corresponding cost functions in [78]. For completeness, we detail them in section 4.6. The weight terms λ_l and λ_v are determined empirically. Eq. (4.5) refers to coplanar constraints for key points where $\mathbf{P}_{ij}$ is the j-th key point that belongs to plane Π_i^r. Eq. (4.6) refers to coplanar constraints for line segments where $\mathbf{A}_{ijk}$ and $\mathbf{B}_{ijk}$ are end-points of the k-th line segment that lies on $\mathbf{L}_j$ that is coplanar to Π_i^r. The function $d(\mathbf{P}, \Pi)$ denotes the perpendicular Euclidean distance between point $\mathbf{P}$ and plane Π.

Since the optimization problem (4.4) is a high dimensional nonlinear optimization problem with many decision variables and constraints, naively applying a nonlinear opti-



(a)



(b)

Figure 4.3: Examples of low-quality local minimum solutions during camera trajectory and map estimation when solving (4.4) by directly applying a nonlinear optimization solver. (a) A local minima solution where the estimated camera trajectory and map significantly deviates from the square-shaped environment. (b) Another local minima where the trajectory become discontinuous while the robot travels at a constant velocity.

mization solver often results in low quality local minima where either the map is distorted or the camera trajectory is discontinuous. Fig. 4.3 shows an example. To alleviate this problem, we rectify the 3D MFG G by solving a sequence of optimization problems described in the following. The approach combines a penalty-barrier type technique, which converts a constrained optimization to an unconstrained optimization with an iterative two-step optimization method to handle the high dimensionality.

4.4.2.1 Step 1: Optimization without Fixed Planes

We decide to solve our optimization problem using a recent sparse optimization solver named g2o [62]. G2o exploits hypergraph and sparse structures to achieve superior computation speed. Recently, g2o becomes a dominating solver for large scale bundle adjustment. However, it is unable to directly handle constrained optimization problems. Adopting the penalty-barrier methods in optimization, we convert the constrained problem in (4.4) to an unconstrained optimization problem by adding a penalty function J_π as the cost of violating coplanarity constraints (4.5) and (4.6),

$$\underset{\substack{\{\mathbf{m}_{i_m}\},\{\mathbf{P}_{i_m}\}\\ \{\mathbf{L}_{i_l}\},\{\mathbf{V}_{i_v}\},\{\Pi_{i_\pi}\}}}{\arg\min} \quad J_{im} + J_\pi, \tag{4.7}$$

where we seek the optimal camera trajectory $\{\mathbf{m}_{i_m}\}$, key points $\{\mathbf{P}_{i_m}\}$, lines $\{\mathbf{L}_{i_l}\}$, vanishing points $\{\mathbf{V}_{i_v}\}$, and planes $\{\Pi_{i_\pi}\}$. The coplanarity cost J_π in the 3D domain is defined by

$$J_\pi = \lambda_{\pi_p} J_{\pi_p} + \lambda_{\pi_l} J_{\pi_l}, \tag{4.8}$$

where J_{π_p} is the coplanarity cost for key points, and J_{π_l} is the coplanarity cost for lines. The weight terms λ_{π_p} and λ_{π_l} are increased progressively by a scalar factor w,

$$\lambda_{\pi_p} = w\lambda_{\pi_p} \text{ and } \lambda_{\pi_l} = w\lambda_{\pi_l}, \tag{4.9}$$

in each iteration. This is a typical penalty method in optimization that gradually guides the solution to satisfy the constraints. The coplanarity costs J_{π_p} and J_{π_l} are defined by

$$J_{\pi_p} = \sum_i \sum_j \rho(e_{ij}^{\pi_p}, \delta_{\pi_p}) \tag{4.10}$$

and

$$J_{\pi_l} = \sum_i \sum_j \rho(e_{ij}^{\pi_l}, \delta_{\pi_l}), \tag{4.11}$$

respectively, where i is the plane number, j is the index of the feature that lies on Π_i. The Huber function [49]

$$\rho(e, \delta) = \begin{cases} e^2 & \text{if } |e| < \delta \\ 2\delta e - \delta^2 & \text{otherwise} \end{cases} \tag{4.12}$$

makes the cost function more robust to outliers which have a residual larger than the threshold δ. The coplanarity error $e_{ij}^{\pi_p}$ for key point $\hat{\mathbf{P}}_j$ is computed by

$$e_{ij}^{\pi_p} = d(\hat{\mathbf{P}}_j, \hat{\Pi}_i), \tag{4.13}$$

and the coplanarity error $e_{ij}^{\pi_l}$ is computed by

$$e_{ij}^{\pi_l} = \frac{1}{n} \sum_{k=1}^{n} d(\hat{\mathbf{A}}_{ijk}, \hat{\Pi}_i) + d(\hat{\mathbf{B}}_{ijk}, \hat{\Pi}_i), \tag{4.14}$$

where n is number of line segments $\mathbf{S}_{ijk}$ that should be collinear with $\mathbf{L}_j$, and $\mathbf{A}_{ijk}$ and $\mathbf{B}_{ijk}$ are the end-points of $\mathbf{S}_{ijk}$. The output of the step yields a new estimated $\hat{G}$, which serves as the initial value for the next step, where we plan to tighten the optimization problem with fixed planes known from lidar data (4.3).

Algorithm 2 Map fusion with the two-step optimization

Input: $G, \{\mathbf{m}_i\}, \{\tilde{\mathbf{p}}_{ij}\}, \{\tilde{\mathbf{s}}_{ijk}\}, \{\Pi_i^r\}$
Output: Rectified $\{\hat{\mathbf{m}}_j\}$ and $\hat{G}$

while $(J_{k-1} - J_k)/J_k \geq \varepsilon$ **do**
  while $i < N$ **do**
    /* Optimization without fixed planes */
    Solve (4.7)
    Increase weights by (4.9)
    $i = i + 1$
  end while
  while $i < N$ **do**
    /* Optimization with fixed planes */
    Solve (4.15)
    Increase weights in (4.16)
    $i = i + 1$
  end while
end while

4.4.2.2 Step 2: Optimization with Fixed Planes

Since we fix planes, the optimization problem in (4.4) is transformed to a lower dimensional problem as follows,

$$\underset{\substack{\{\mathbf{m}_{im}\},\{\mathbf{P}_{ip}\},\\ \{\mathbf{L}_{il}\},\{\mathbf{V}_{iv}\}}}{\arg\min} \quad J_{im} + J_\pi^r. \tag{4.15}$$

Note that planes are removed from decision variables if compared to (4.4). The penalty function J_π^r

$$J_\pi^r = \lambda_{\pi_p} J_{\pi_p}^r + \lambda_{\pi_l} J_{\pi_l}^r, \tag{4.16}$$

follows a similar format to those in (4.8–4.14) except that the planes $\hat{\Pi}_i$ are replaced with known values Π_i^r in (4.3). During the process of optimization, we increase weighting on the penalty function using the same way as before.

The above two steps are repeated until $(J_{k-1} - J_k)/J_k < \varepsilon$ where J_k is the error $J_{im} + J_\pi^r$

at the k-th iteration, and ε is the error threshold. The overall algorithm is summarized in Algorithm 2.

Remark 1 *Algorihtm 2 can also be used to rectify piecewise visual SLAM maps. Due to uneven distribution of features, a visual SLAM method tends to fail in featureless environments (e.g. when its camera faces a large white wall). Such regions are not unusual in indoor environments and cause maps from visual SLAM to become disconnected. However, with our algorithm, we can relink those piecewise maps from the same or different robots / mobile device users using the 2D lidar data. All we need to do is to ensure wall corners exist in each disconnected map.*

4.5 Experiments

We have implemented our proposed algorithm in C++. To evaluate the performance of our proposed algorithm, we test our method on real data collected from an indoor environment containing many planar walls. As mentioned in the previous section, the 3D landmark map and 2D lidar map have been generated using the implementation of [78] and [59], respectively, which is publicly available.

To generate the 3D landmark map, we have used an indoor image sequence with a resolution of 640×360 taken by a camera with 65° HFOV. An example image is shown in Fig. 4.4(a). Due to low image resolution and lack of distinctive features on planar walls, the trajectory estimate becomes less accurate and contains significant scale drift. The ground-truth camera trajectory has been obtained by first placing artificial patterns on the wall with the 3D positions measured manually and then computing the camera position at the best frames using the perspective-n-point method with the manually specified 3D-2D point correspondences. The test data set consists of 12,000 images with a trajectory length of over 68 meters.

Figure 4.4: Sample image frame where visible artifical landmarks are used to compute the ground-truth camera position.

Lidar data have been acquired by a lidar mounted on a robot shown in Fig. 4.4(b). The lidar data have been collected by a Hokuyo UTM-30LX with a 30m and 270° scanning range at a scanning frequency of 25Hz. Although loop-closing has not been performed, the lidar map is more accurate when compared to the 3D landmark map due to less scale drift in the mapping.

4.5.1 Mapping Error

To assess the quality of the fused map, we compute the mapping error by measuring the error between the estimated wall corner positions and the reference lidar map wall corner positions. To measure the corner error, we first align the estimated trajectory to the ground-truth trajectory using Horn's method [48]. The method estimates the transformation $\mathtt{T}$ that minimizes

$$\mathtt{E}_k = \mathtt{P}_k^{-1}\mathtt{T}\hat{\mathtt{P}}_k, \tag{4.17}$$

where $\mathtt{P}_k \in \mathrm{SE}(3)$ and $\hat{\mathtt{P}}_k \in \mathrm{SE}(3)$ are the ground-truth and estimated camera pose, respectively. After the ground truth and estimated trajectories are aligned, we compute the Euclidean distance between the corresponding corner positions i by

$$e_i = d(\mathbf{c}_i^r, \mathbf{c}_i^g). \tag{4.18}$$

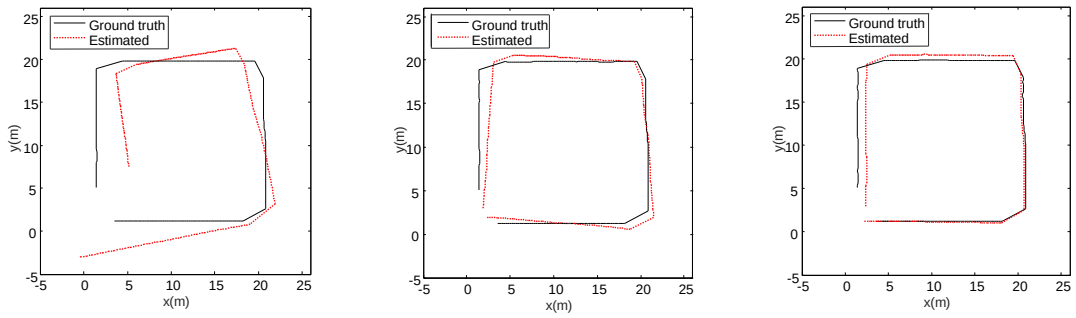
The mean and standard deviation (SD) of the mapping error at each optimization step are shown in Table 4.1. In Table 4.1, row vSLAM refers to the direct output of visual SLAM results using [78]. It also provides initial inputs to two algorithms that we are comparing: the naive approach and the proposed method. The naive approach refers to the method of simply fixing the estimated planes to the reference planes from lidar data during the whole optimization process. It is worth noting that the test data is challenging for visual SLAM due to regions with very few features.

The results show that the proposed approach significantly reduces mapping error in visual SLAM results in both mean and SD of wall corners. Comparing with vSLAM, the proposed method reduces mean error from 2.94 to 0.85 meters, which is a reduction of 71.1%. Moreover, the results show that the proposed method is able to further reduce error if compared to the naive approach. Mean error is reduced by 24.1% and SD is reduced by 19.7%.

4.5.2 Absolute Trajectory Error

Furthermore, the fusion process improves the estimated robot trajectory quality. We employ the absolute trajectory error (ATE) to measure the global consistency of the estimated trajectory. Using (4.17), the ATE is computed by

$$e = \sum_k ||\mathrm{trans}(\mathtt{E}_k)||, \tag{4.19}$$



(a) (b) (c)

Figure 4.5: Trajectory comparison. (Best viewed in color). (a) Initial trajectory estimate from vSLAM, (b) trajectory estimate from the naive method, and (c) trajectory estimate from the proposed method.

where trans(E_k) is the translation component. Table 4.2 shows the root-mean-square error (RMSE), RMSE normalized by length, SD, and max error for the ATE measured for our data. Fig. 4.5 shows the corresponding trajectories. The results show that the proposed approach significantly reduces ATE in visual SLAM results in all performance metrics. Again, our method outperforms the naive approach.

4.6 Objective Functions in (4.4)

The individual cost terms in (4.4) are described in the following.

4.6.1 Key Points

Assuming Gaussian noise in the measurement $\tilde{\mathbf{p}}_{ij}$, the error between $\tilde{\mathbf{p}}_{ij}$ and $\hat{\mathbf{p}}_{ij}$ is defined by

$$e_{ij}^{p}(\mathbf{m}_j) = (\tilde{\mathbf{p}}_{ij} - \hat{\mathbf{p}}_{ij})^{\mathsf{T}} \Sigma_p^{-1} (\tilde{\mathbf{p}}_{ij} - \hat{\mathbf{p}}_{ij}), \tag{4.20}$$

where $\Sigma_p = \sigma_p^2 \mathbf{I}$. Then,

$$J_p = \sum_j \sum_i \rho(e_{ij}^{p}(\mathbf{m}_j), \delta_p) \tag{4.21}$$

where i is the feature number, and j is the key frame index.

4.6.2 Collinear Line Segments

Let $\mathbf{L}_i := (\mathbf{Q}_i, \mathbf{D}_i)$ be a 3D line, where $\mathbf{Q}_i \in \mathbb{R}^3$ is a point on $\mathbf{L}_i$, and $\mathbf{D}_i \in \mathbb{R}^3$ is a vector that represents the direction of $\mathbf{L}_i$. The projection of $\hat{\mathbf{L}}_i$ in I_j is denoted by $\hat{\mathbf{l}}_{ij}$. Let $\tilde{\mathbf{s}}_{ijk} := (\tilde{\mathbf{p}}_{ijk}, \tilde{\mathbf{q}}_{ijk})$ be the k-th observed line segment in I_j that should be collinear to $\mathbf{L}_i$. Let $\hat{\mathbf{p}}'_{ijk}$ and $\hat{\mathbf{q}}'_{ijk}$ be the perpendicular foot of $\tilde{\mathbf{p}}_{ijk}$ and $\tilde{\mathbf{q}}_{ijk}$ on $\hat{\mathbf{l}}_{ij}$, respectively. Then the error between line segments $\tilde{\mathbf{s}}_{ijk}$ and $\hat{\mathbf{l}}_{ij}$ is defined by

$$e^l_{ij}(\mathbf{m}_j) = \sum_k (\tilde{\mathbf{p}}_{ijk} - \hat{\mathbf{p}}'_{ijk}) \Sigma_l^{-1} (\tilde{\mathbf{p}}_{ijk} - \hat{\mathbf{p}}'_{ijk}) + \sum_k (\tilde{\mathbf{q}}_{ijk} - \hat{\mathbf{q}}'_{ijk}) \Sigma_l^{-1} (\tilde{\mathbf{q}}_{ijk} - \hat{\mathbf{q}}'_{ijk}).$$

respectively, $\Sigma_l = \sigma_l^2 \mathbf{I}$. Then,

$$J_l = \sum_j \sum_i \rho(e^l_{ij}(\mathbf{m}_j), \delta_l) \tag{4.22}$$

where i is the feature number, and j is the key frame index.

4.6.3 Vanishing Points

Let $\mathbf{V}_i \in \mathbb{P}^3$ be a vanishing point. The observation of $\mathbf{V}_i$ in I_j is denoted by $\tilde{\mathbf{v}}_{ij} \in \mathbb{R}^2$, and the projection of $\hat{\mathbf{V}}_i$ in I_j is denoted by $\hat{\mathbf{v}}_{ij}$. Assuming Gaussian noise in the measurement, the error between $\tilde{\mathbf{v}}_{ij}$ and $\hat{\mathbf{v}}_{ij}$ is defined by

$$e^v_{ij}(\mathbf{m}_j) = (\tilde{\mathbf{v}}_{ij} - \hat{\mathbf{v}}_{ij})^\mathsf{T} \Sigma_{v_{ij}}^{-1} (\tilde{\mathbf{v}}_{ij} - \hat{\mathbf{v}}_{ij}), \tag{4.23}$$

Table 4.1: Mapping errors

	Mean (m)	SD (m)
vSLAM	2.94	1.93
Naive	1.12	0.66
Proposed	**0.85**	**0.53**

Table 4.2: Absolute trajectory errors

	RMSE (m)	SD (m)	Max (m)	$\frac{\text{RMSE}}{\text{length}}$
vSLAM	2.96	2.10	8.00	4.4%
Naive	0.78	0.56	2.69	1.2%
Proposed	**0.67**	**0.38**	**1.55**	**1.0%**

where $\Sigma_{v_{ij}} = \sigma_v^2 \mathbf{I}$. Then,

$$J_v = \sum_j \sum_i \rho(e_{ij}^v(\mathbf{m}_j), \delta_v). \tag{4.24}$$

where i is the feature number, and j is the key frame index.

4.7 Conclusions

We presented a method that can fuse two maps generated from different sensory modalities, i.e., monocular vision and a 2D lidar, with the hope to assist low-cost devices/robots to obtain high quality localization information. We formulated a new heterogeneous map fusion problem that takes data from a camera and a 2D lidar captured at different perspective and time. We extracted planes from both sensing modalities and used them as the anchoring information for map rectification. Since the fusion process is a large non-linear optimization problem, the solution can fall into an undesirable local minimum even with the anchoring planes available. We proposed a two-step optimization method assisted by a penalty function. We implemented the algorithm and tested with real data. The initial

results showed that the proposed algorithm significantly improves visual SLAM results in both mapping accuracy and trajectory errors. The approach was compared to the naive approach which simply applied a nonlinear optimization solver to the same problem. The proposed method also outperforms the naive approach in all performance metrics.

5. VISUAL PROGRAMMING FOR MOBILE ROBOTS USING HIGH-LEVEL LANDMARKS

5.1 Introduction

As service-oriented robots increase, they have to be programmed by users without much technical background. The average user will need easier and more flexible robot programming tools as opposed to the specialized programming languages used for industrial robots [8]. For mobile robots, the most fundamental task is to program robot navigation capabilities, which direct a robot to move from point A to point B in an environment in which prior maps often are not available. For a novice user, generating a path of waypoints on a sparse landmark map produced by prior simultaneous localization and mapping (SLAM) outputs is nontrivial due to the combination of human's poor spacial reasoning capability and lack of contextual information.

Here we propose a visual programming system that allows users to specify navigation tasks for mobile robots using high-level landmarks in a virtual reality (VR) environment constructed from the output of visual SLAM (vSLAM) (See Fig. 5.1). The VR graphical user interface (GUI) of our system allows users to take a virtual tour similar to Google Street View to familiarize themselves with the robot's working environment. Users can directly specify image regions of objects as high-level landmarks from the scene. With each specified landmark, the user creates task-level motion commands. In the preprocessing step, our system builds a roadmap by using the pose graph from the vSLAM outputs. Based on the roadmap, the high-level landmarks, and task-level motion commands, our system generates an output path for the robot to accomplish the navigation task.

We present data structures, architecture, interface, and algorithms for our system. Given n_s search-type motion commands, our system generates a path in $O(n_s(n_r \log n_r +$

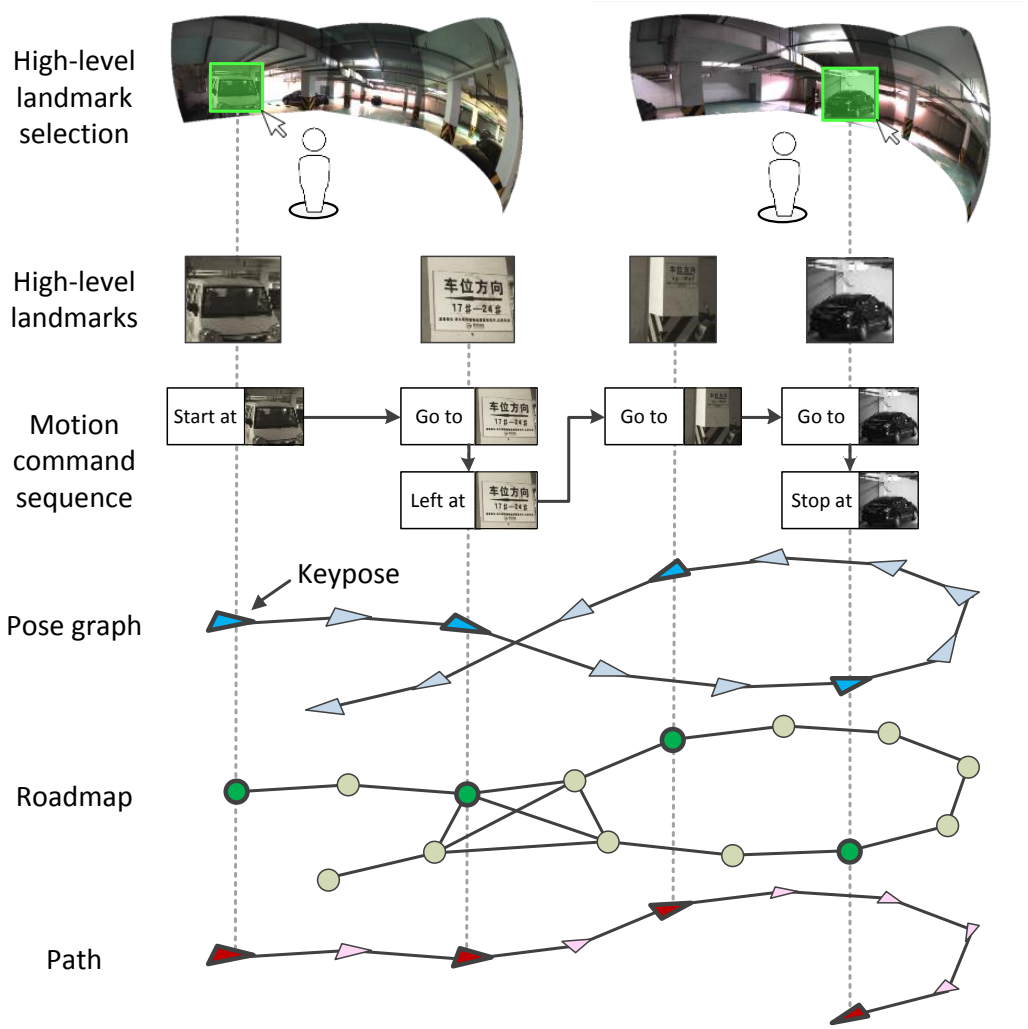


Figure 5.1: Mobile robot programming with our proposed system (Best viewed in color). Our system generates a robot path from a user-defined motion command sequence that uses high-level landmarks. This enables users to program mobile robots at the object level without dealing with low-level map coordinates.

$m_r)$) time, where n_r and m_r are the number of roadmap nodes and edges, respectively. We have implemented our system and tested it with real indoor data acquired by a mobile robot equipped with a synchronized camera array and a lidar sensor. The experimental results show that our system can generate robot paths that satisfy the user commands.

5.2 Related Work

Our proposed robot programming system relates to the areas of robot programming, motion planning, SLAM, and teleoperation.

In [8], Biggs and MacDonald provide a survey on robot programming which divides the methods into manual programming and automatic programming. In manual programming, the user directly programs the robot using either text-based [53] or graphical systems [18], whereas automatic programming refers to methods such as learning and programming by demonstration, a common technique used for manipulator robots [2, 80]. For mobile robots, Kanayama and Wu [53] develop a high-level programming language for text-based programming. Nicolescu and Mataric [89] develop an instructive system that programs a mobile robot using learning by demonstration. However, the effort to train a mobile robot in this approach will grow linear to space, and many training examples are required to increase flexibility. Our system can be viewed as a new task-level automatic programming using VR.

Robot motions can be generated by motion planning algorithms with classic methods such as [55, 63]. However, motion planning requires a complete scene representation which is often unavailable for service mobile robots in unstructured environments. Also, motion planning can be a difficult task for a robot and can benefit from human input. For example, a human-assisted motion planner is proposed in [25] where the user can steer the planner towards/away from certain regions. In [66], a remote human-in-the-loop gripper is presented where the user can assist motion planning by moving a virtual gripper in the

display and specifying waypoints. Our path generation step can be viewed as a motion planning problem with a given roadmap. However, inspired by the existing works, the main focus of our system is to translate user intention into the motion planning framework to automate the path generation process.

To display the robot's working environment, our system employs VR built by keyframes from vSLAM. We also use pose graph from the vSLAM as a starting point for the roadmap construction. SLAM can be performed using depth sensors and visual sensors, i.e., regular cameras. SLAM methods based on depth sensors such as lidar [87, 106] and RGB-D cameras [47, 28, 86] have low scale-drift issues, whereas visual sensors provide both geometric and appearance data. In vSLAM, the extended Kalman filter [26, 13, 91, 12] and bundle adjustment [92, 60, 31] are mainly used. In our work, we use a 2D lidar map generated by the method in [59] and use the pose graph and keyframes of the multi-layered feature graph (MFG) proposed by [78] which exploits the regularities of urban environment such as rectilinear structures.

Integrating trajectory following and obstacle avoidance capabilities [3], our system can be used as a supervisory control system in teleoperation. Fong and Thorpe [36] classify vehicle teleoperation interfaces into four categories: direct [64], multimodal/multisensor, supervisory control [82], and novel [30]. To allow a free-look of the environment, a visual teleoperation display is also proposed for unmanned vehicles using a spherical camera and an Oculus Rift [61]. It has been shown that VR can significantly improve telepresence. Our system falls into this category. However, directly controlling the robot can be difficult due to poor spacial reasoning of the user from the display. Our high-level landmark-based supervisory control can avoid this issue.

Our experience in vSLAM using mobile robots [70, 78] and teleoperation [43, 96] has prompted a need for a system that allows users to easily assist robot navigation in urban environments.

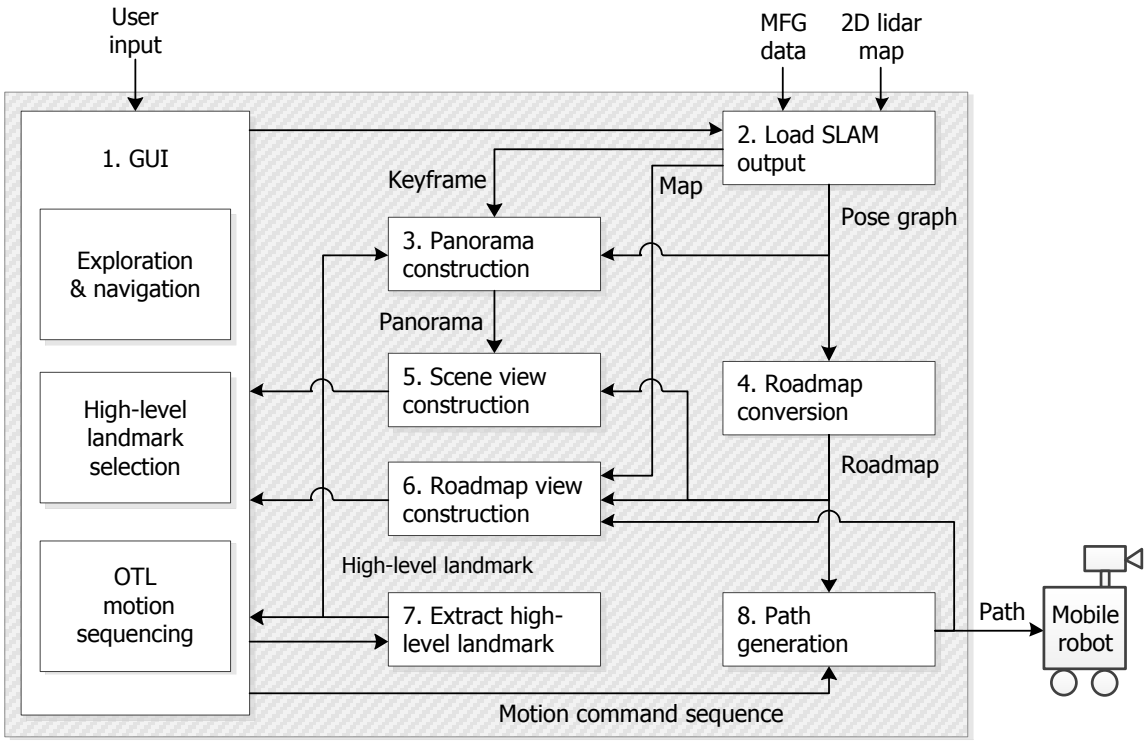


Figure 5.2: System diagram

5.3 System Design

5.3.1 Overall Structure

The system diagram of our visual programming system is depicted in Fig. 5.2. Using our system, a user explores and navigates through the robot's environment as a virtual tour and specifies high-level landmarks which are used for object-oriented task-level (OTL) motion sequencing. Given the OTL motion commands, our system generates a robot path.

From the GUI, the user first loads the MFG data and 2D lidar map to construct the robot's environment and roadmap. The environment is created using both keyframes and the pose graph of the MFG data, and the roadmap is created from the pose graph as a preprocessing step. The user navigates along the roadmap and keeps track of his current pose in the roadmap view, where the roadmap is overlaid on the 2D lidar map. From the scene view, the user selects high-level landmarks which are used for OTL motion sequencing. After the user completes motion sequencing, the system generates a path from the given motion commands. The robot path is then displayed to the user and used by a mobile robot.

The generated path is not limited to our robot but can be used among different type of mobile robots. When the path is generated off-line, the path can be used to program a mobile robot. However, as noted previously, our system can also be used as an interface for on-line robot control in a teleoperation scenario.

5.3.2 Data Structures

The data structures in the MFG data that are used in our system are listed as follows:

- *Keyframe* is a camera image denoted by I_i^k, where k denotes the key index, and i denotes the camera index in the synchronized camera array.

- *Keypose* is an estimated robot pose when keyframe I_i^k is captured. Let the camera

pose for I_i^k be defined by a rotation matrix $\mathtt{R}_i^k \in \mathrm{SO}(3)$ and a translation vector $\mathbf{t}_i^k \in \mathbb{R}^3$. Then the keypose is aligned with the reference camera pose in the camera array, i.e., $\mathtt{R}_0^k$ and $\mathbf{t}_0^k$.

- *Pose graph* is an undirected graph P, where nodes represent keyposes and edges represent their dependencies.

Additionally, the required data structures used to generate a robot path from user commands are illustrated in Fig. 5.1 and defined below.

- *High-level landmark* is a user-specified image region in I_i^k. The image region is defined by a bounding box and labeled by the user. However, if an image recognition algorithm is available, high-level landmarks can also be identified by the system.

- *Motion command* is a text command paired with a high-level landmark by the user. The text commands specify robot tasks, such as *start at*, *go to*, and *avoid*.

- *Motion command sequence* is a directed graph M, where a node m_i represents a motion command, and edge (m_i, m_j) represents the transition from m_i to a child node m_j. Each edge has a condition that should be satisfied to make the transition.

- *Roadmap* is an undirected graph R, where the k-th node r_k represents a waypoint $\mathbf{r}_k \in \mathbb{R}^3$. An edge (r_{k_1}, r_{k_2}) in R represents the Euclidean distance between r_{k_1} and r_{k_2}.

- *Path* is a continuous sequence S where the elements are robot positions in $\mathbb{R}^3$.

5.3.3 User Interface

In this section, we describe the GUI of our system (Fig. 5.2 Box 1) that facilitates robot programming using high-level landmarks. Our GUI, shown in Fig. 5.3, consists

primarily of four components: *scene view*, *roadmap view*, *motion command panel*, and *motion sequencing panel*. These components support mainly three user activities for robot programming: exploration and navigation, high-level landmark selection, and OTL motion sequencing. We detail them in the following sections.

5.3.3.1 Exploration and Navigation

When the user loads the MFG data and 2D lidar map (Fig. 5.2 Box 2), the scene view and roadmap view display the robot's environment in an interactive OpenGL scene. Both scene view and roadmap view assist the user with exploration and navigation and provide situation awareness.

From the scene view, the user views the scene as a panorama image that is constructed using the MFG keyframes (Fig. 5.2 Box 3). For immersive viewing experience, we use a spherical panorama, i.e., a panorama texture mapped on a spherical surface. The user can explore the environment using pan, tilt, and zoom controls in the scene view. The roadmap R is also overlaid in the scene view so that the user can see the navigable paths.

In the roadmap view, the roadmap R is overlaid on the 2D lidar map that is viewed top-down. The roadmap R is constructed from the pose graph P using the algorithm described in Sec. 5.3.4 (Fig. 5.2 Box 4). When the user navigates through the environment, the scene view camera moves along the nodes in R and the spherical panorama is created at the current position of the scene camera. The roadmap view allows translating and zooming for exploration of the environment.

5.3.3.2 High-level Landmark Selection

When the user explores the environment using the scene view, the user can specify high-level landmarks directly from the panorama image (Fig. 5.2 Box 7). To specify a high-level landmark, the user uses a mouse to specify a bounding box corner in the panorama image to extract objects of interest. When the user clicks the spherical

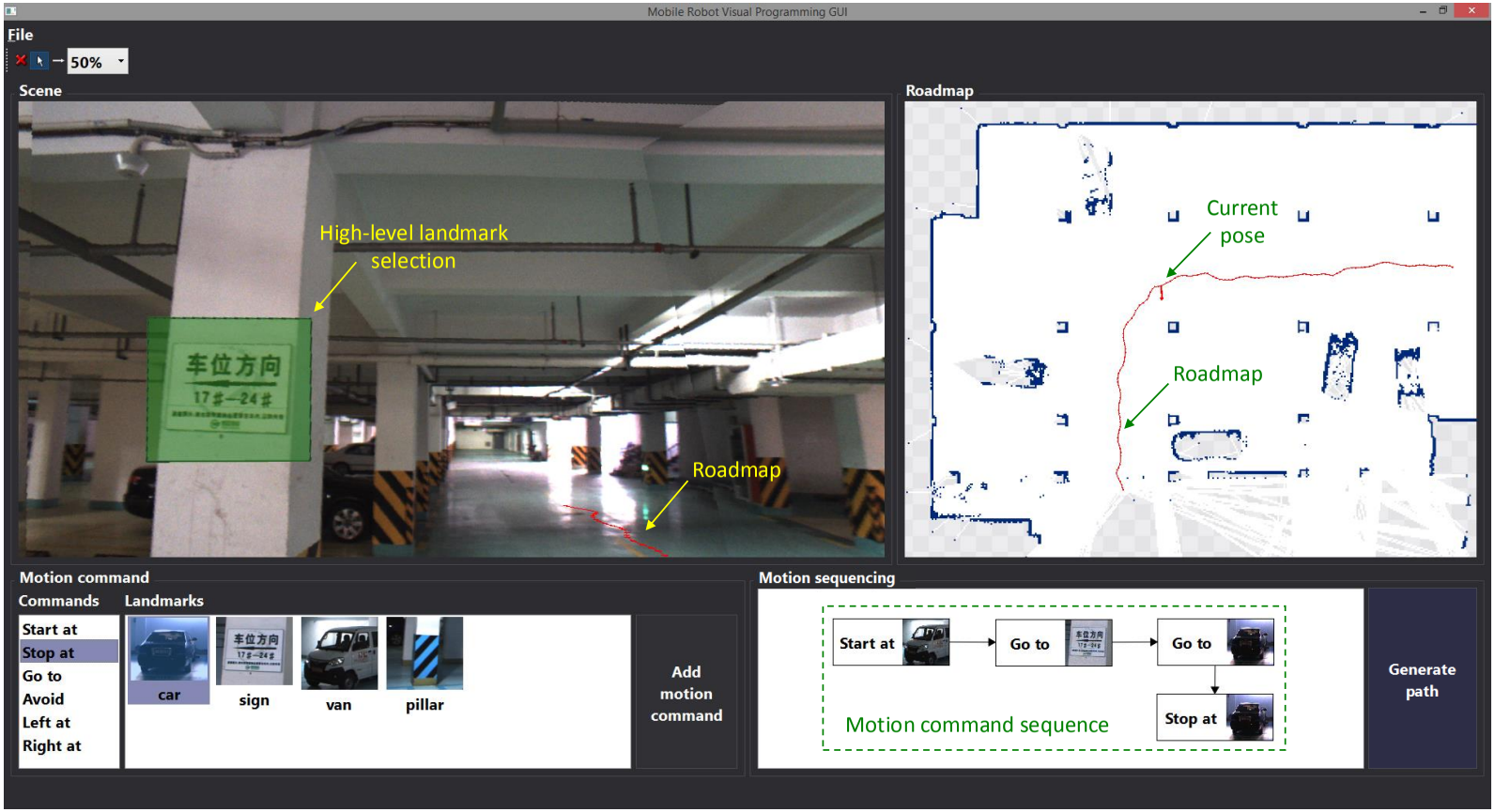


Figure 5.3: The GUI of our system. (Best viewed in color.) The GUI mainly consists of four components: the scene view (top-left), roadmap view (top-right), motion command panel (bottom-left), and motion sequencing panel (bottom-right).

panorama, an intersection point $\mathbf{X}$ between the sphere and the ray that originates from the mouse position is computed.

Let Π_i^k be the image plane of the camera image I_i^k, and let $\mathbf{C}_i^k$ be the camera center. The intersection point $\mathbf{I}$ between Π_i^k and the line $\mathbf{L} = \mathbf{C}_i^k + (\mathbf{X} - \mathbf{C}_i^k)t$ where $t \in \mathbb{R}$ is computed by

$$\mathbf{I} = \mathbf{C}_i^k + (\mathbf{X} - \mathbf{C}_i^k)((\mathbf{P} - \mathbf{C}_i^k) \cdot \mathbf{N}_i^k)/((\mathbf{X} - \mathbf{C}_i^k) \cdot \mathbf{N}_i^k), \tag{5.1}$$

where $\mathbf{P}$ is a point on Π_i^k and $\mathbf{N}_i^k$ is its normal. Then, the (x,y) image coordinates of $\mathbf{I}$, i.e., the bounding box corner, in image I_i^k is computed.

When the bounding box is defined, the high-level landmark is highlighted in the spherical panorama. The region inside the bounding box in I_i^k is colored and mapped to the panorama texture $T^k(u,v)$ by using the maps

$$m_i^x(u,v) = x, \quad m_i^y(u,v) = y, \tag{5.2}$$

which transforms $I_i^k(x,y)$ to $T^k(u,v)$ by

$$T^k(u,v) = I_i^k(m_i^x(u,v), m_i^y(u,v)), \tag{5.3}$$

where (u,v) is the texture coordinate and is related to the spherical coordinates by $\theta = v\pi$ and $\phi = \pi(2u-1)$, where θ and ϕ are the latitude and longitude angles of the coordinates of the sphere

$$\mathbf{S} = [r\sin(\theta)\cos(\phi), -r\cos(\theta), -r\sin(\phi)\sin(\theta)], \tag{5.4}$$

where r is the radius.

A label of the landmark can be added through the landmark list and is displayed in the scene view. The high-level landmarks can be saved as a file and can be loaded for later

usage.

5.3.3.3 OTL Motion Sequencing

The user performs motion sequencing using the motion command panel and the motion sequencing panel. The motion command panel contains two list views: the *command list* and *landmark list*. The user selects an item from each list view to form a motion command. Here, we only focus on six commands: *go to*, *start at*, *stop at*, *left at*, *right at*, and *avoid*. New commands can be added per task requirement.

When a high-level landmark is specified by the user, it is displayed as an iconized landmark in the landmark list in the motion command panel. The user selects an item from each list, i.e., a text command and high-level landmark, to form a motion command. When the user presses the *Add motion command* button, an orphan node corresponding to the user-defined motion command is created in the motion sequencing panel.

After the user creates motion commands, which are the building blocks of a motion command sequence, the user can arrange motion commands in the motion sequencing panel to create a motion command sequence M for higher-level tasking. Since high-level landmarks are displayed in the nodes, the user can create high-level commands in an object-oriented fashion. After an orphan node is created from the motion command panel, using a mouse, the user can connect the nodes with directed edges to create a motion sequence. The interface allows the user to add/remove nodes and edges in the graph. When an edge is selected, the user can add transition conditions. In our system, a transition condition is whether a motion command node is visited for a certain number of times (See Fig. 5.7d for example.) After the user constructs a sequence, the user presses the *Generate path* button, and the system will compile the motion command sequence M and generate a path S that satisfies the commands (Fig. 5.2 Box 8). The path S is overlaid on the roadmap view which allows the user to map the high-level commands into physical

Algorithm 3 Pose Graph To Roadmap Preprocessing

Input: Pose graph P, search radius d, number of nearest neighbors k to search
Output: Roadmap R

```
/* Initialize roadmap */
R ← P                                                    ▷ O(n_p + m_p)
/* Build kd-tree */
T ← KD-TREE-CREATE(V(R))                                 ▷ O(n_p log n_p)
/* Connect near keyposes */
for each vertex r ∈ V(R) do                              ▷ O(n_p)
    /* Find k-nearest neighbors within fixed radius */
    Q ← KD-TREE-SEARCH(T, r, k, d)                       ▷ O(k log n_p)
    for each vertex q ∈ Q do                             ▷ O(k)
        if edge (r, q) ∉ E(R) then
            R ← R ∪ {(r, q)}
        end if
    end for
end for
```

space at a glance. High-level landmarks associated with the motions are also displayed along the path in the roadmap view.

5.3.4 Algorithms

When the user loads the MFG data, the pose graph P is converted into the roadmap R as a preprocessing step using Algorithm 3 (Fig. 5.2 Box 4). When the user creates the motion sequence and presses the *Generate path* button the path is generated using Algorithm 4 (Fig. 5.2 Box 8). In the following, for a graph G, we denote the set of vertices and edges by $V(G)$ and $E(G)$, respectively.

In Algorithm 3, roadmap R is first initialized by copying the nodes and edges from P. The value of each node $r_k \in V(R)$ is set as the keypose position $\mathbf{r}_k = -\mathrm{R}_0^{k\mathsf{T}}\mathbf{t}_0^k$, and the value of each edge $(r_a, r_b) \in E(R)$ is set by $distance[(r_a, r_b)] = \|\mathbf{r}_a - \mathbf{r}_b\|$. We build a kd-tree with the roadmap node values $\mathbf{r}_k$. Then, for each $\mathbf{r}_k$, we perform a k-nearest neighbor search within a fixed radius and obtain the set of nodes Q. We add an edge from node r_k to each

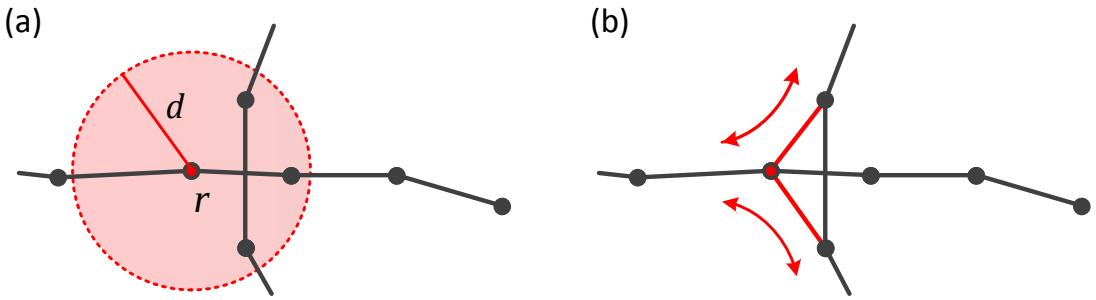


Figure 5.4: Pose graph to roadmap. (a) Nearest neighbors search region. (b) Newly added edges and navigable paths.

node in Q if it is not already a neighboring node. This is illustrated in Fig. 5.4. If we let $n_p = |V(P)|$ and $m_p = |E(P)|$, the time complexity of Algorithm 3 is $O(kn_p \log n_p + m_p)$.

Algorithm 4 generates a path S from a motion command sequence M. In a graph point of view, there are two types of motion commands in our system: *graph-search-type commands* and *graph-editing-type commands*. The former includes *start at*, *stop at*, and *go to*, and performs a search in the current roadmap R_c. The latter includes *avoid*, *left at*, and *right at* which modifies edge weights in R_c. For graph-search-type commands, a shortest path search using Dijkstra's algorithm is performed. For the graph-editing-type commands, the edges extracted from EXTRACT-EDGES, which depends on the specified command, are modified. For example, see Fig. 5.5. Let d_r be the maximum degree of R, and let d_m be the maximum outdegree of M. If we let $n_r = |V(R)|$ and $m_r = |E(R)|$, the graph-search-type commands run in $O(n_r \log n_r + m_r)$, whereas the graph-editing-type command runs in $O(d_r)$. Let the number of graph-search-type commands be n_s and let the number of graph-editing-type commands be n_e. Then the worst case time complexity of Algorithm 4 is $O(n_s(n_r \log n_r + m_r + d_m) + n_e d_r)$. If we consider that d_m, d_r, n_s, and n_e are usually small compared to n_r and m_r the time complexity becomes $O(n_s(n_r \log n_r + m_r))$.

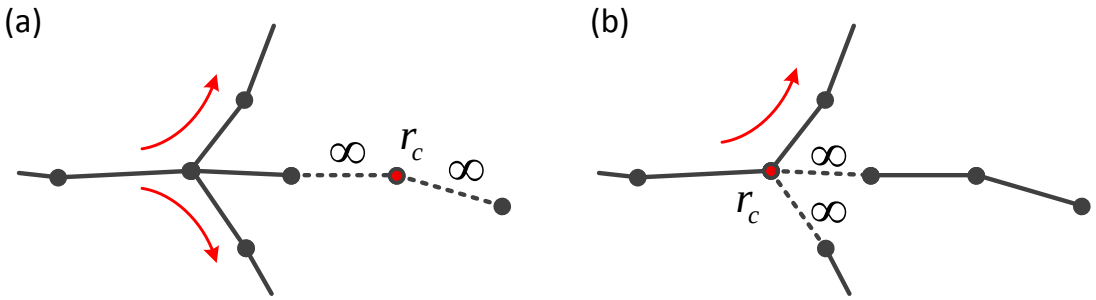


Figure 5.5: Modified edge weights and navigable paths using graph-editing-type commands (a) *avoid* and (b) *left at*.

5.4 Experiments

We have implemented our system in C++. To evaluate the effectiveness of our system, we have tested our system using real world data. We use a set of motion command sequences as input and verify that the generated output paths are correct. We have also tested the runtime of our algorithm and show numerical test results.

5.4.1 Data Set

Our ground mobile robot that was used to collect data is built on an X80Pro platform. The wheel-based platform has two 12V motors and can move at a maximum speed of 75cm/sec. A SICK TiM571 lidar which has a 25m-depth range, a 270°-angular coverage range, and a scanning frequency of 15Hz is mounted on the platform. On top of the robot, six Point Grey USB3 2.1MP color cameras (BFLY-U3-23S6C-C) are mounted. The cameras are synchronized by an external trigger and capture 960×600 resolution images at a frequency of 1Hz. This frame rate is suitable for our application due to slow robot movement. Here, we present data collected from two environments: office and garage. The size of the environment is shown in Fig. 5.6, where each chessboard square in the background of the 2D lidar map is $1m^2$. The office data contains 45 keyposes, and the garage data contains 181 keyposes in the pose graph.

5.4.2 Roadmap Construction and Path Generation

We have tested our system using a set of motion command sequences to verify whether our system can generate correct motion paths. For each dataset, we first load the MFG data and 2D lidar map into our system. Then we provide the system with different motion command sequences for testing.

When the MFG data is loaded, our system constructs a roadmap from the pose graph using Algorithm 3. Figs. 5.6a and 5.6e show the constructed roadmaps for the office and garage data, respectively. Note that in Fig. 5.6a, the original sequential pose graph is converted into a loop which allows the user to navigate in different directions.

Next, we use a set of motion command sequences (MCSs) to test Algorithm 4. Fig. 5.7 shows the high-level landmarks and motion command sequences that are used for testing: MCS #1 and MCS #2 are used for the office data, and MCS #3 is used for the garage data. Fig. 5.6b shows the generated path from MCS #1 which contains only graph-search-type commands. The resulting path is the shortest path on the roadmap. We use MCS #2 to test a combination of graph-search-type and graph-editing-type commands. Figs. 5.6c and 5.6d show the resulting partial paths where a longer route is taken due to the *avoid* command. Finally, we use MCS #3 to test the transition conditions used in a repetitive motion task. Figs. 5.6f, 5.6g, and 5.6h show that the generated partial paths. Due to the transition conditions, the pillar, sign, and object high-level landmarks are visited twice.

5.4.3 Runtime Test

We test the runtime of our algorithm under different parameter settings. The testing computer is a PC laptop with a 1.9GHz Intel Core i7 CPU and 8GB RAM. The operating system is a 64-bit Windows 8. We test both algorithms in Sec. 5.3.4 on the garage data to see how they perform depending on the size of the input pose graph and roadmap. Fig. 5.8a shows the runtime of Algorithm 3 respect to the pose graph size n_p. For each k nearest

neighbor settings, we compute the average of 1000 runs. Fig. 5.8b shows the runtime of Algorithm 4 respect to the roadmap size n_r. We increase the number of graph-search-type commands n_s from 1 to 5 in a motion command sequence. The results show an average of 100 runs for each n_s. Trends in both Figs. 5.8a and 5.8b conform to our complexity analysis.

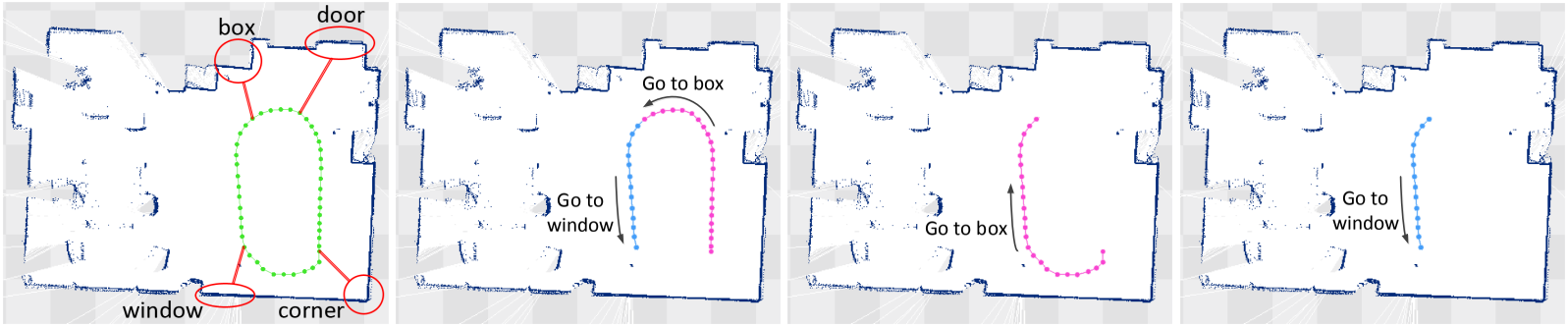
5.5 Conclusions

We proposed a new system that allows users to visually program mobile robots. The system used a VR environment constructed from keyframes in vSLAM results to allow users to experience robot working environment and define high-level landmarks associate with OTL motion commands. Our algorithms converted the OTL motion commands to weight point paths to guild robots without overtaxing on human spatial reasoning capability. We presented the data structures, system architecture, user interface, and algorithms. We tested our system using real world data and showed that the generated paths satisfied the motion task requirements.

Algorithm 4 Path Generation

Input: Motion command sequence M, roadmap R, current node $r_c \in R$
Output: Path S

1: $m_{curr} \leftarrow head[M]$
2: **while** $m_{curr} \neq$ NIL **do**
3: /* Handle motion command */
4: $c \leftarrow command[m_{curr}]$
5: $r_t \leftarrow target[m_{curr}]$
6: **if** $c =$ START **or** $c =$ STOP **or** $c =$ GOTO **then**
7: /*graph-search-types */
8: $T \leftarrow$ DIJKSTRA(R_c, r_c, r_t) ▷ $O(n_r \log n_r + m_r)$
9: $S \leftarrow S \cup T$
10: $r_c \leftarrow r_t$
11: /* Restore graph edges */
12: $R_c \leftarrow R$
13: **else if** $c =$ AVOID **or** $c =$ LEFT **or** $c =$ RIGHT **then**
14: /*graph-edit-types */
15: $E \leftarrow$ EXTRACT-EDGES(R_c, r_c, c, S) ▷ $O(d_r)$
16: **for** each $e \in E$ **do** ▷ $O(d_r)$
17: $distance[e] \leftarrow \infty$
18: **end for**
19: **end if**
20: /* Handle transition conditions */
21: $m_{next} \leftarrow$ NIL
22: $k \leftarrow 0$
23: **for** each edge $e \in outedge[m_{curr}]$ **do** ▷ $O(d_m)$
24: **if** $condition[e] =$ TRUE **then**
25: $m_{next} \leftarrow head[e]$
26: $k \leftarrow k + 1$
27: **end if**
28: **end for**
29: **if** $k > 1$ **then**
30: **error** "invalid motion command sequence"
31: **end if**
32: $m_{curr} \leftarrow m_{next}$
33: **end while**



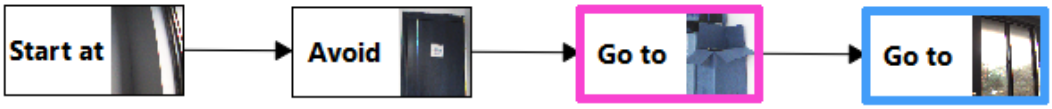
(a) Roadmap and landmarks (b) Path segments from MCS #1 (c) Path segment from MCS #2 (d) Path segment from MCS #2

(e) Roadmap and landmarks (f) Path segments from MCS #3 (g) Path segment from MCS #3 (h) Path segments from MCS #3

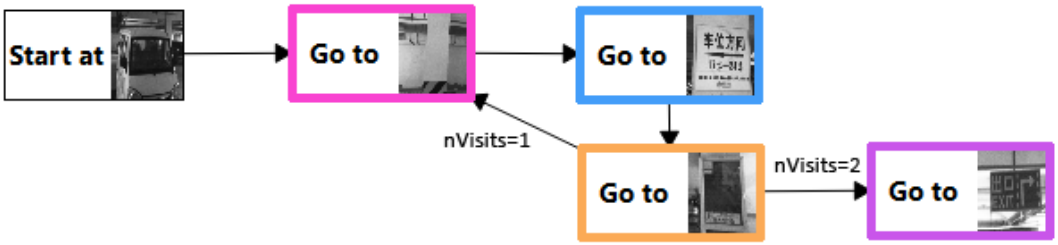
Figure 5.6: Roadmap and paths (Best viewed in color). The color corresponds to the colors in Fig. 5.7.

(a) High-level landmarks in Figs. 5.6a and 5.6e

(b) MCS #1: Graph-search-type commands only

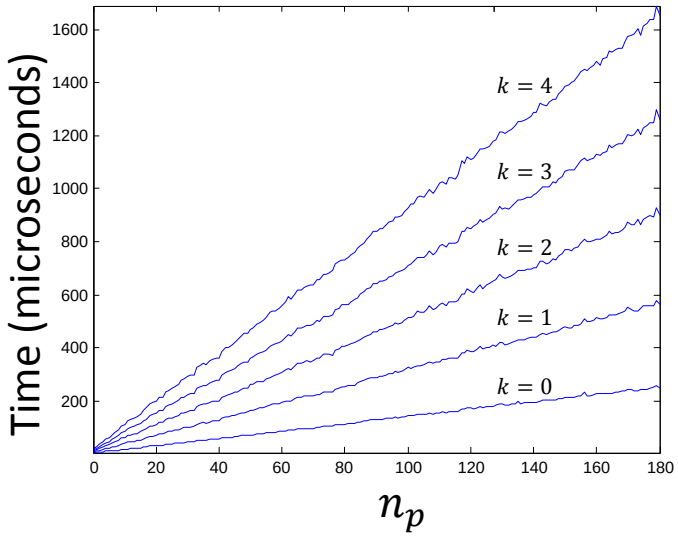


(c) MCS #2: Graph-search-type commands with graph-editing-type commands

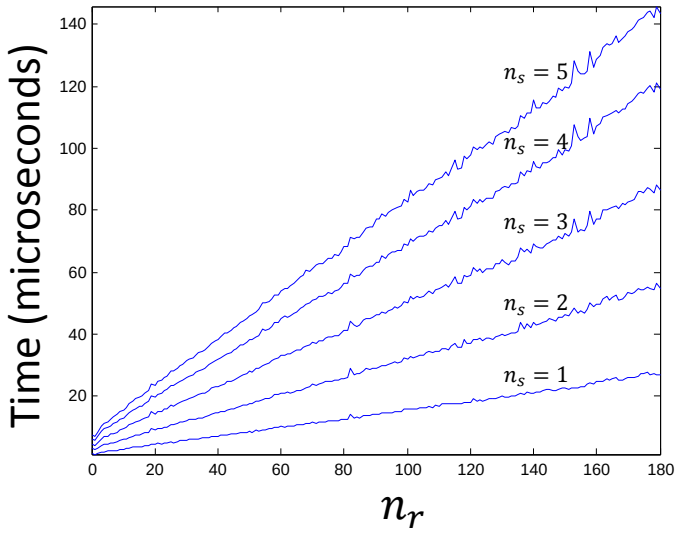


(d) MCS #3: Graph-search-type commands with a cycle

Figure 5.7: Sample motion command sequences (MCSs) for testing (Best viewed in color). The color of the motion command nodes correspond to the colors of their resulting path segments in Fig. 5.6.



(a)



(b)

Figure 5.8: Runtime results

6. CONCLUSIONS AND FUTURE WORK

In our work we developed both system and algorithms that utilizes geometric and appearance information to assist robot navigation in urban areas. First, we developed SILK, an appearance-based image registration method, which estimates geometric transformation between two images and takes into account of the image resolution. The method fully exploits appearance information and is especially useful when registering low-resolution image regions in video. Next, we proposed a segmentation and mapping method for PBFs, an important landmark in urban environments. Our method showed that appearance information can be used effectively to distinguish man-made object from natural objects and multiple geometric constraints can be used to obtain accurate position information. By using the vertical planes in urban areas, we have also proposed an algorithm that can rectify visual SLAM results using a 2D lidar map. This method enables benefiting from both the accuracy of lidar and the lower cost in cameras in a distributed robot system, where a single robot with a lidar sensor can assist localization of other robots with a camera. Finally, we develop a novel system that enables mobile robot programming where a user can define high-level landmarks directly from appearance data. By using appearance information, the user can specify high-level commands and assign tasks in a more intuitive manner.

During our studies, we have identified problems that we felt important to address. We list these problems in the following as future work.

In chapter 2, our proposed method, SILK, estimates similarity transformations, i.e., translation, rotation, and scale, between images. Since the Gaussian function, used in (2.8), behaves well under affine transformations, extending SILK to work for affine transformation is trivial. However, it is unclear how to extend the method for projective transformation at this moment. Also, future work will involve extending our method to use

a coarse-to-fine strategy to avoid local minima and develop performance metrics of the proposed registration method.

When estimating the 3D structure of PBFs in chapter 3, the weights in the objective function (3.10) are used to balance the cost of each geometric constraints, e.g. reprojection error and coplanarity error. Since these costs are in different units, e.g. the reprojection error is computed in the image domain while the coplanarity error is computed in 3D, finding the right balance for the weights of geometric constraints can be difficult. In our approach, each weight is determined by the initial value of its corresponding geometric cost and is used to normalize each geometric cost so that each cost does not overwhelm the others. However, the initial value of each cost term will not accurately capture the unit difference between geometric constraints. Hence, we will explore geometric costs that can be computed in the same domain. Another issue is that the weights in objective function (3.10) are fixed during the entire optimization process. Since this can affect the optimization performance, we will explore the effects of weight updating during the optimization process using penalty methods in non-linear programming.

In chapter 4, the visual SLAM map and lidar map is matched by manually specifying corresponding wall corners. We will improve wall corner correspondence by developing a topological graph matching algorithm. Also, a shortcoming of the proposed method is the reliance on vertical plane distribution. Although most indoor environments are dominated by vertical planes, this method would fail in an environment with very few vertical planes. We are interested in extracting other signatures such as object boundary points to handle the problem.

In our current visual programming system, the user commands are limited to motion commands that are related to the robot's position. To meet different application needs, an extended set of commands need to be developed so that the commands can be used to control the robot's orientation. We would also like to extend our method so that the

optimal motion sequencing is not restricted to the exact pose graph locations.

High-level scene representation for mobile robots has yet to be explored fully. Additional to the problems mentioned above, our long term goal is aimed toward the development of higher-level scene representation that uses landmarks with semantic meaning. This can be achieved with the help of recent advances in statistical learning methods. The outcome of new high-level scene representations will allow mobile robot navigation systems to be more robust and enable communication and sharing of spatial knowledge among robots and humans.

REFERENCES

[1] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden. 1984, Pyramid methods in image processing. *RCA Engineer*, 29(6):33–41, 1984.

[2] Sonya Alexandrova, Maya Cakmak, Kaijen Hsiao, and Leila Takayama. Robot programming by demonstration with interactive action visualizations. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.

[3] Ronald C. Arkin. *An Behavior-based Robotics*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[4] C. Baillard and A. Zisserman. A plane-sweep strategy for the 3D reconstruction of buildings from multiple images. In *ISPRS Journal of Photogrammetry and Remote Sensing*, pages 56–62, 2000.

[5] Simon Baker and Takeo Kanade. Limits on super-resolution and how to break them. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:1167–1183, 2002.

[6] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *Int. J. Comput. Vision*, 56(3):221–255, February 2004.

[7] L. Baudouin, Y. Mezouar, O. Ait-Aider, and H. Araujo. Multi-modal sensors path merging. In *13th International Conference on Intelligent Autonomous System*, Padova, Italy, July 2014.

[8] Geoffrey Biggs and Bruce Macdonald. A survey of robot programming systems. In *in Proceedings of the Australasian Conference on Robotics and Automation, CSIRO*, page 27, 2003.

[9] Andras Bodis-Szomoru, Hayko Riemenschneider, and Luc Van Gool. Fast, approximate piecewise-planar modeling based on sparse structure-from-motion and superpixels. June 2014.

[10] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[11] S. Carpin. Merging maps via hough transform. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1878–1883, Sept 2008.

[12] Javier Civera, Oscar G Grasa, Andrew J Davison, and JMM Montiel. 1-point ransac for extended kalman filtering: Application to real-time structure from motion and visual odometry. *Journal of Field Robotics*, 27(5):609–631, 2010.

[13] Laura A Clemente, Andrew J Davison, Ian D Reid, José Neira, and Juan D Tardós. Mapping large loops with a single hand-held camera. In *Robotics: Science and Systems*, volume 2, page 11, 2007.

[14] Alejo Concha and Javier Civera. Using superpixels in monocular slam. In *Robotics and Automation, 2014 IEEE International Conference on*, 2014.

[15] D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 2004.

[16] G.H. Costa and J.C.M. Bermudez. Are registration errors always bad for super-resolution? In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 1, pages I–569 –I–572, april 2007.

[17] John L. Crassidis and John L. Junkins. *Optimal Estimation of Dynamic Systems (Chapman & Hall/Crc Applied Mathematics & Nonlinear Science)*. Chapman & Hall/CRC, April 2004.

[18] Wenrui Dai and M. Kampker. User oriented integration of sensor operations in a offline programming system for welding robots. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 1563–1567 vol.2, 2000.

[19] A.J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, volume 2, pages 1403–1410, Oct 2003.

[20] A.J. Davison and N. Kita. 3d simultaneous localisation and map-building using active vision for a robot moving on undulating terrain. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–384–I–391, 2001.

[21] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(6):1052–1067, June 2007.

[22] Gksel Dedeoglu and GauravS. Sukhatme. Landmark-based matching algorithm for cooperative mapping by autonomous robots. In *Distributed Autonomous Robotic Systems 4*, pages 251–260. Springer Japan, 2000.

[23] Göksel Dedeoğlu, Simon Baker, and Takeo Kanade. Resolution-aware fitting of active appearance models to low resolution images. In *Proceedings of the 9th European conference on Computer Vision - Volume Part II*, ECCV'06, pages 83–97, Berlin, Heidelberg, 2006. Springer-Verlag.

[24] J. A. Delmerico, P. David, and J. J. Corso. Building facade detection, segmentation, and parameter estimation for mobile robot localization and guidance. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1632–1639, 2011.

[25] Jory Denny, Read Sandström, Nicole Julian, and Nancy M. Amato. A region-based strategy for collaborative roadmap construction. In *Algorithmic Foundations of Robotics XI - Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics, WAFR 2014, 3-5 August 2014, Boğaziçi University, İstanbul, Turkey*, pages 125–141, 2014.

[26] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *Robotics & Automation Magazine, IEEE*, 13(2):99–110, 2006.

[27] Ethan Eade and Tom Drummond. Edge landmarks in monocular {SLAM}. *Image and Vision Computing*, 27(5):588 – 596, 2009.

[28] Felix Endres, Jürgen Hess, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard. 3-d mapping with an rgb-d camera. *IEEE Transactions on Robotics*, 30(1):177–187, 2014.

[29] Jakob Engel, Thomas Schops, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *Proceedings of the European Conference on Computer Vision*, Zurich, Switzerland, September 2014.

[30] C. Escolano, J.M. Antelis, and J. Minguez. A telepresence mobile robot controlled with a noninvasive brain-computer interface. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 42(3):793–804, June 2012.

[31] A. Eudes, M. Lhuillier, S. Naudet-Collette, and M. Dhome. Fast odometry integration in local bundle adjustment-based visual slam. In *Proceedings of the 20th International Conference on Pattern Recognition*, pages 290–293, Aug 2010.

[32] B. Fan, F. Wu, and Z. Hu. Robust line matching through line-point invariants. *Pattern Recognition*, 45(2):794–805, February 2012.

[33] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, September 2004.

[34] Renan U. Ferreira, Edson M. Hung, and Ricardo L. de Queiroz. Video super-resolution based on local invariant features matching. In *IEEE International Conference on Image Processing (ICIP)*, Orlando, USA, September 2012.

[35] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

[36] Terrence W Fong and Chuck Thorpe. Vehicle teleoperation interfaces. *Autonomous Robots*, 11(1):09–18, July 2001.

[37] D. F. Fouhey, D. Scharstein, and A. J. Briggs. Multiple plane detection in image pairs using J-linkage. In *Proceedings of the International Conference on Pattern Recognition*, pages 336–339, 2010.

[38] D. Fox, J. Ko, K. Konolige, B. Limketkai, D. Schulz, and B. Stewart. Distributed multirobot exploration and mapping. *Proceedings of the IEEE*, 94(7):1325–1339, July 2006.

[39] R. G. von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall. LSD: a line segment detector. *Image Processing On Line*, 2012.

[40] Andrew P. Gee, Denis Chekhlov, Walterio Mayol, and Andrew Calway. Discovering planes and collapsing the state space in visual slam. In *Proceedings of the 18th British Machine Vision Conference*, September 2007.

[41] A.P. Gee, D. Chekhlov, A. Calway, and W. Mayol-Cuevas. Discovering higher level structure in visual slam. *Robotics, IEEE Transactions on*, 24(5):980–990, Oct 2008.

[42] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[43] K. Goldberg, Dezhen Song, and A. Levandowski. Collaborative teleoperation using networked spatial dynamic voting. *Proceedings of the IEEE*, 91(3):430–439, Mar 2003.

[44] G. Griffin, A. Holub, and P. Perona. The Caltech-256. Technical report, California Institute of Technology, 2007.

[45] O. Haines, J. Martnez-Carranza, and A. Calway. Visual mapping using learned structural priors. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2227–2232, 2013.

[46] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[47] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663, 2012.

[48] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987.

[49] Peter J. Huber. Robust estimation of a location parameter. *Annals of Mathematical Statistics*, 35(1):73–101, March 1964.

[50] Michal Irani and Shmuel Peleg. Improving resolution by image registration. *CVGIP: Graph. Models Image Process.*, 53(3):231–239, April 1991.

[51] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. isam2: Incremental smoothing and mapping with fluid relinearization and incremental vari-

able reordering. In *IEEE International Conference on Robotics and Automation*, pages 3281–3288, May 2011.

[52] K. Kanatani. Calibration of ultrawide fisheye lens cameras by eigenvalue minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(4):813–822, 2013.

[53] Y.J. Kanayama and C.T. Wu. It's time to make mobile robots programmable. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 1, pages 329–334 vol.1, 2000.

[54] R. Kaushik and J. Xiao. Accelerated patch-based planar clustering of noisy range images in indoor environments for robot mapping. *Robotics and Autonomous Systems*, 60(4):584–598, April 2012.

[55] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, Aug 1996.

[56] D. Keren, S. Peleg, and R. Brada. Image sequence enhancement using sub-pixel displacements. In *Computer Vision and Pattern Recognition, 1988. Proceedings CVPR '88., Computer Society Conference on*, pages 742 –746, jun 1988.

[57] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR. 6th IEEE and ACM International Symposium on*, pages 225–234, Nov 2007.

[58] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, November 2007.

[59] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics*, November 2011.

[60] Kurt Konolige and Motilal Agrawal. Frameslam: From bundle adjustment to real-time visual mapping. *IEEE Transactions on Robotics*, 24(5):1066–1077, 2008.

[61] K. Kruckel, F. Nolden, A. Ferrein, and I. Scholl. Intuitive visual teleoperation for ugvs using free-look augmented reality displays. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4412–4417, May 2015.

[62] R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3607–3613, Shanghai, China, May 2011.

[63] Steven M. Lavalle, James J. Kuffner, and Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

[64] Dongjun Lee, O. Martinez-Palafox, and M.W. Spong. Bilateral teleoperation of a wheeled mobile robot over delayed communication network. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 3298–3303, May 2006.

[65] T. Lee, S. Lim, S. Lee, S. An, and S. Oh. Indoor mapping using planes extracted from noisy RGB-D sensors. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1727–1733, 2012.

[66] A. Leeper, Kaijen Hsiao, M. Ciocarlie, L. Takayama, and D. Gossow. Strategies for human-in-the-loop robotic grasping. In *Human-Robot Interaction (HRI), 2012 7th*

ACM/IEEE International Conference on, pages 1–8, March 2012.

[67] T. Lemaire and S. Lacroix. Monocular-vision based slam using line segments. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2791–2796, April 2007.

[68] J. J. Leonard and H. F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, Jun 1991.

[69] H. Li, D. Song, Y. Lu, and J. Liu. A two-view based multilayer feature graph for robot navigation. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3580–3587, 2012.

[70] Wen Li and Dezhen Song. *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, chapter Featureless Motion Vector-Based Simultaneous Localization, Planar Surface Extraction, and Moving Obstacle Tracking, pages 245–261. Springer International Publishing, 2015.

[71] Y. Li and S. Birchfield. Image-based segmentation of indoor corridor floors for a mobile robot. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 837–843, 2010.

[72] Zhouchen Lin and Heung-Yeung Shum. Fundamental limits of reconstruction-based superresolution algorithms under local translation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(1):83–97, January 2004.

[73] Tony Lindeberg. *Scale-Space*. John Wiley & Sons, Inc., 2007.

[74] Hongche Liu, Tsai-Hong Hong, Martin Herman, Ted Camus, and Rama Chellappa. Accuracy vs efficiency trade-offs in optical flow algorithms. *Comput. Vis. Image*

Underst., 72(3):271–286, December 1998.

[75] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[76] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150 –1157 vol.2, 1999.

[77] Y. Lu, D. Song, Y. Xu, A.G.A. Perera, and S. Oh. Automatic building exterior mapping using multilayer feature graphs. In *IEEE International Conference on Automation Science and Engineering*, pages 162–167, 2013.

[78] Yan Lu and Dezhen Song. Visual navigation using heterogeneous landmarks and unsupervised geometric constraints. *IEEE Transactions on Robotics*, 31(3):736–749, June 2015.

[79] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision (darpa). In *Proceedings of the 1981 DARPA Image Understanding Workshop*, pages 121–130, April 1981.

[80] S. Manschitz, J. Kober, M. Gienger, and J. Peters. Learning to sequence movement primitives from demonstrations. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 4414–4421, Sept 2014.

[81] Jose Martinez-Carranza and Andrew Calway. Unifying planar and point mapping in monocular slam. In *Proceedings of the British Machine Vision Conference*, pages 43.1–43.11. BMVA Press, 2010.

[82] C. Masone, A. Franchi, H.H. Bulthoff, and P.R. Giordano. Interactive planning of persistent trajectories for human-assisted navigation of mobile robots. In *Intelligent*

Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pages 2641–2648, Oct 2012.

[83] N. D. Molton, A. J. Davison, and I. D. Reid. Locally planar patch features for real-time structure from motion. In *Proc. British Machine Vision Conference*. BMVC, 2004.

[84] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 116–121, Mar 1985.

[85] R. Mur-Artal, J.M.M Montiel, and J.D. Tardos. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, PP(99):1–17, 2015.

[86] Richard A. Newcombe, Dieter Fox, and Steven M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *The IEEE Conference on Computer Vision and Pattern Recognition*, June 2015.

[87] P. Newman, D. Cole, and K. Ho. Outdoor slam using visual appearance and laser ranging. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1180–1187, May 2006.

[88] Nhat Xuan Nguyen. *Numerical Algorithms for Image Super-resolution*. PhD thesis, Stanford University, Scientific Computing and Computational Mathematics, July 2000.

[89] Monica N. Nicolescu and Maja J. Mataric. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, pages 241–248, New York, NY, USA, 2003. ACM.

[90] Christopher C. Paige and Michael A. Saunders. Lsqr: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Softw.*, 8(1):43–71, March 1982.

[91] Lina M Paz, Pedro Piniés, Juan D Tardós, and José Neira. Large-scale 6-dof slam with stereo-in-hand. *Robotics, IEEE Transactions on*, 24(5):946–957, 2008.

[92] Eric Royer, Maxime Lhuillier, Michel Dhome, and Jean-Marc Lavest. Monocular vision for mobile robot localization and autonomous navigation. *International Journal of Computer Vision*, 74(3):237–260, 2007.

[93] Renato F. Salas-Moreno, Richard A. Newcombe, Hauke Strasdat, Paul H.J. Kelly, and Andrew J. Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[94] R.F. Salas-Moreno, B. Glocken, P.H.J. Kelly, and A.J. Davison. Dense planar slam. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 157–164, Sept 2014.

[95] Paul Smith, Ian Reid, and Andrew Davison. Real-time monocular SLAM with straight lines. In *Proc. British Machine Vision Conference*, pages 17–26, Edinburgh, 2006.

[96] D. Song and K. Goldberg. Sharecam part 1: interface, system architecture, and implementation of a collaboratively controlled robotic webcam. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, pages 1080–1086 vol.2, Oct 2003.

[97] H. Strasdat, J. M. M. Montiel, and A. Davison. Scale drift-aware large scale monocular slam. In *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June

2010.

[98] Hauke Strasdat, JMM Montiel, and Andrew J Davison. Real-time monocular slam: Why filter? In *2010 IEEE International Conference on Robotics and Automation*, pages 2657–2664, 2010.

[99] Y. Taguchi, Y.-D. Jian, S. Ramalingam, and C. Feng. Point-plane slam for hand-held 3D sensors. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2013.

[100] Y. Taguchi, Yong-Dian Jian, S. Ramalingam, and Chen Feng. Point-plane slam for hand-held 3d sensors. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5182–5189, May 2013.

[101] C. Taylor and A. Cowley. Parsing indoor scenes using RGB-D imagery. In *Proceedings of Robotics: Science and Systems*, July 2012.

[102] C. J. Taylor and D. J. Kriegman. Structure and motion from line segments in multiple images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(11):1021–1032, November 1995.

[103] R. Toldo and A. Fusiello. Robust multiple structures estimation with J-linkage. In *Proceedings of the European Conference of Computer Vision*, pages 537–547, 2008.

[104] E. Tretyak, O. Barinova, P. Kohli, and V. Lempitsky. Geometric image parsing in man-made environments. *International Journal of Computer Vision*, 97(3):305–321, May 2012.

[105] Michalis Vrigkas, Christophoros Nikou, and Lisimachos P. Kondi. On the improvement of image registration for high accuracy super-resolution. In *ICASSP*, pages 981–984. IEEE, 2011.

[106] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems Conference*, July 2014.

[107] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2015.

[108] Liang Zhao, Shoudong Huang, Lei Yan, and Gamini Dissanayake. A new feature parametrization for monocular slam using line features. *Robotica*, FirstView:1–24, 3 2014.

[109] J. Zhou and B. Li. Homography-based ground detection for a mobile robot platform using a single camera. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 4100–4105, 2006.

[110] Barbara Zitová and Jan Flusser. Image registration methods: a survey. *Image and Vision Computing*, 21:977–1000, 2003.