Online Lidar and Vision based Ego-motion Estimation and Mapping

Ji Zhang

CMU-RI-TR-17-04

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Robotics.

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

February 2017

Thesis Committee:
Sanjiv Singh, Chair
Martial Hebert
Michael Kaess
Larry Matthies, JPL

Abstract

In many real-world applications, ego-motion estimation and mapping must be conducted online. In the robotics world, especially, real-time motion estimates are important for control of autonomous vehicles, while online generated maps are crucial for obstacle avoidance and path planning. Further, the complete map of a traversed environment can be taken as an input for further processing such as scene segmentation, 3D reasoning, and virtual reality.

To date, fusing a large amount of data from a variety of sensors in real-time remains a nontrivial problem. The problem is particularly hard if is to be solved in 3D, accurately, robustly, and in a small form factor. This thesis proposes to tackle the problem by leveraging range, vision, and inertial sensing in a coarse-to-fine manor, through multi-layer processing. In a modularized processing pipeline, modules taking light computation execute at high frequencies to gain robustness w.r.t. high-rate, rapid motion. Modules consuming heavy processing run at low frequencies to ensure accuracy in resulting motion estimates and maps.

Further, the modularized processing pipeline is capable of handling sensor degradation by automatic reconfiguration bypassing failure modules. Vision-based methods typically fail in low-light or texture-less scenes. Likewise, lidar-based methods are problematic in symmetric or extruded environments such as a long and straight corridor. When such degradation occurs, the proposed pipeline automatically determines a degraded subspace in the problem state space, and solves the problem partially in the well-conditioned subspace. Consequently, the final solution is formed by combination of the "healthy" parts from each module.

The proposed ego-motion estimation and mapping methods have been validated in extensive experiments ranging from car-mounted, hand-carried, to drone-attached setups. Experiments are conducted in various environments covering structured urban areas as well as unstructured natural scenes. Results indicate that the methods can carry out high-precision estimation over a long distance of travel as well as robustness w.r.t. high-speed, aggressive motion and environmental degradation.

Acknowledgments

I would like to express the deepest appreciation to my committee chair, Dr. Sanjiv Singh, and committee members for their support and guidance in the preparation and improvement of this document and the underlying research.

I would like to thank my wife and parents for their encouragement and patience.

In addition, a thank you to my colleagues for their and invaluable help over the journey my PhD study. These include but not limit to Marcel Bergerman, Michael Kaess, George Kantor, Sebastian Scherer, Stephen Nuske, Daniel Huber, David Wettergreen, Kevin Dowling, Dave Duggins, Steven Huber, Lucas Nogueira, Volker Grabe, Ethan Abramson, Stephan Roth, Brad Hamner, Lyle Chamberlain, Benjamin Grocholsky, Paul Bartlett, David Murphy, Spencer Spiker, Kimber Rugg, Patrick DeFranco, Rauhit Ashar, Luke Yoder, Silvio Maeta, Chuck Whittaker, Zheng Fang, Yu Zhang, Yu Song, Shichao Yang, Lantao Liu, Daniel Maturana, Geetesh Dubey, Sezal Jain, Sankalp Arora, Sanjiban Choudhury, Mike Uenoyama, Srinivasan Vijayarangan, David Butterworth, Rushat Chadha, and Vivek Velivela.

Contents

Chapter 1

Introduction

1.1 Motivation

Creation of three dimensional representations of the world is a common need across a large number of applications (see Fig. 1.1(a) for an example). The state of the art comes from the usage of laser scanning. Lidars are particularly suited to 3D reconstruction because the laser range accuracy can be sub-centimeter and does not vary with respect to the distance measured. It is also affected little by ambient light or the surface normal of most objects. Laser beams can be made highly coherent and hence can provide high spatial resolution. Finally, because laser measurements can be made at a high rate (up to hundreds of KHz), sampling can provide high temporal resolution as well.

The downside of lidar mapping is in registration of the data. Since accurate reconstruction requires high precision in the pose of the lidar when the measurements are taken, industrial solutions in 3D reconstruction use one of two methods. First, data is taken from mechanisms that stay stationary except for the motion of the actuator that directs the laser beam (see Fig. 1.1(b) for an example). Lidar data is collected from different locations and merged offline. The method is time consuming and easy to create occlusions on the resulting map since scans are collected from a limited number of locations.

(a) (b) (c)

Figure 1.1: An example lidar map and the survey lidars used in creation of the map. (a) A lidar map of an urban area. (b) A Faro scanner attached to a tripod. (c) A Riegl scanner mounted to a driving vehicle. The vehicle is equipped with a high-accuracy GPS/INS system to provide motion estimates for lidar data registration.

(a) (b)

Figure 1.2: Two examples where real-time motion estimates and maps are needed simultaneously for navigation. (a) is an autonomous drone for bridge mapping and inspection. (b) is a self-driving orchard vehicle for transportation, mowing, and spraying. In both cases, high-accuracy GPS/INS systems are impractical due to the weight and cost. GPS signals are also unreliable underneath the trees in the orchard or structures of the bridge.

The other method is to use independent positioning systems (such as a high-accuracy GPS/INS system) to register data from moving lidars (see Fig. 1.1(c) for an example). However, high-accuracy GPS/INS systems are often practically impossible due to the weight, cost, or availability of GPS signals. Two examples are shown in Fig. 1.2. In the left figure, an autonomous drone flies around and through bridges for mapping and inspection. High-accuracy GPS/INS systems are too heavy to carry due to the limited flight payload. In the right figure, a self-driving utility vehicle operates in orchards for transportation, mowing, and spraying. Sensors used on the vehicle are cost-sensitive. In both cases, GPS signals are unreliable underneath the structures of the bridge or trees in the orchard.

This thesis explores the capability to create high-accuracy maps from a moving platform without precise and independent positioning being available. It is commonly understood that obtaining such maps without independent positioning requires solving for the positioning and mapping simultaneously. However, it is unobvious how to realize this accurately in 3D, in real-time, robustly, and in a small form factor.

In essence, this thesis seeks a way to combine range, vision, and inertial sensing to compute ego-motion and build a 3D reconstruction of the environment. While the 3D reconstruction can be further improved by post-processing using loop closure [50], this thesis will concentrate on real-time methods capable of providing high precision motion estimates and maps. The online and real-time attributes make the resulting system suitable for autonomous navigation: the motion estimates are important for vehicle control, and the maps are crucial for obstacle avoidance and path planning. Further, the complete map of the traversed environment can be taken as an input for further processing such as scene segmentation, 3D reasoning, and virtual reality.

1.2 Problem Statement

The thesis proposes to solve a state estimation problem by fusing data from a variety of sensors to estimate the ego-motion in real-time and develop a 3D reconstruction of the traversed environment. Major difficulty of solving the aforementioned problem rises from several aspects. Especially, the proposed study involves different sensor mortality while sensors of each type

must be characterized distinctively. Lidars are good at measuring range. Range measurements from lidars are accurate and insensitive to ambient light or surface materials. However, lidar measurements are made continuously over time, resulting in a unique timestamp associated with each laser point. When the lidar moves in 6-DOF, high-frequency motion estimates must be made to remove the distortion caused by the lidar extrinsic motion, as to register the laser points on a coherent map. Very often, state estimation is solved with a large number of variables to approximate the continuous motion over time.

Camera can capture visual patterns from the environment. With global shutter cameras, all pixels in an image share the same timestamp, significantly reducing the difficultly of the state estimation problem. However, because of its instinctive nature, cameras are sensitive to lighting condition changes. Further, texture-less environments such as a homogeneously colored surface can cause difficulty to vision-based methods.

In the proposed thesis work, I seek a novel path where data from various sensors are fused. Instead of combining all sensor data in a large, full-blown problem, I parse the processing into multiple modules and solve sequentially in a coarse-to-fine manner. Each module in the system explores the advantages of a particular sensor, solves a sub-problem, and generates results for the following modules to process. The modules are arranged in an order where the demand of processing decreases – modules executing at high frequencies enable the estimation of high-rate, rapid motion; modules consuming more processing run at low frequencies to ensure accurate ego-motion estimation and 3D reconstruction.

The modularized processing pipeline also improves robustness, by selecting "healthy" sensor modes when forming the final solution. Here, "healthy" means the sensor data contains sufficient information to carry out the desired estimation. For example, when a camera is in a low-light or texture-less scene, useful information becomes sparse causing estimation failures. Likewise, in a symmetric or extruded environment such as a long and straight corridor, lidar-based methods can produce ambiguous poses sliding along the corridor. In these cases, the proposed method automatically determines a degraded subspace in the problem state space. When degradation happens, the problem is solved partially in the well-conditioned subspace. The result is that only the "healthy" parts are combined to produce the final solution.

1.3 Summary of Contributions

To solve the ego-motion estimation and mapping problem, the thesis includes a number of high-level contributions:

- A modularized processing pipeline to fuse data from range, vision, and inertial sensors for ego-motion estimation and mapping through muli-layer optimization. The resulting system achieves high accuracy and low drift.
- The proposed processing pipeline is dynamically reconfigurable, which fully or partially bypasses failure modules and combines the rest system to deal with sensor degradation. It can handle environmental degradation and aggressive motion.
- A method embedded in the processing pipeline determines sensor degradation. It sepa-

rates well-conditioned subspace from degenerate subspace in the underlying problem state space, solves the problem only in the well-conditioned subspace, and therefore actively eliminates sensor noise in the degraded subspace.

- In vision processing module, the method registers range measurements on a depthmap and associates depth information to visual features. Depth information provides additional constraints in ego-motion estimation and fixes scale ambiguity.
- Also in vision processing module, the method involves visual features from three categories – features with depth from the range measurements or depthmap, features with depth triangulated using the estimated motion, and features without depth information, to maximumly utilize features' information.
- In lidar processing module, the method uses a two-level voxel representation to efficiently retrieve registered laser points for scan matching.
- Also in lidar processing module, the method conducts scan matching on multiple CPU threads. Each CPU thread process an individual scan while multiple scans are processed in parallel to carry out the onboard processing in real-time.

The thesis may be summarized in the statement:

Coarse-to-fine fusion of sensor data for ego-motion estimation and mapping offers high accuracy, robustness, and reduced computational complexity in data processing.

1.4 Document Outline

The remainder of the thesis is documented as follows:

Chapter 2 summarizes existing literature in related fields.

Chapter 3 presents vision-based ego-motion estimation assisted by range measurements.

Chapter 4 addresses lidar-based ego-motion estimation and mapping.

Chapter 5 details complete processing pipeline combining range, vision, and inertial sensing.

Chapter 6 further compares the three methods.

Chapter 7 discusses estimation robustness w.r.t. sensor degradation.

Chapter 8 explores usage of the proposed processing pipeline in aerial applications.

Chapter 9 summarizes completed work and suggests possible future work.

Chapter 2

Related Work

2.1 Vision-based Approaches

2.1.1 Stereo Camera

Camera projects 3D environments onto 2D imagery. Through the process of imaging, depth information does not retain. When using cameras for ego-motion estimation, a common way is to employ multiple cameras and utilize the geometry between the cameras to retrieve the depth information. Typically, two cameras are involved to form a stereo pair [15, 35, 45, 62, 70]. When matching visual features between two cameras, depth of the features can be computed through triangular geometry. The baseline between the cameras is a known variable and functions as a reference. For example, Paz et al.'s method estimates the ego-motion of a hand-held stereo camera where depth information is retreaved for features close to the camera [76]. Konolige, at al's stereo visual odometry recovers the camera motion from bundle adjustment [56]. The method is integrated with an IMU to handle orientation drift and therefore is able to work over a long distance of travel in off-road environments.

Stereo visual odometry can be further extended to use three or more cameras [75]. When overlap exists between the cameras in the field of view, depth information can be retrieved in the intersection areas. However, without any overlap, the multi-camera system is equivalent to a monocular camera with a large field of view [87].

Except that several advantages exist in stereo and multi-camera systems, usage of these system also comes at their own cost. If the system has a small baseline, the cameras will reduce to a monocular camera in an open area where features are far away. This is typically the case where visual odometry is used on an aircraft flying at a high altitude. On the other hand, if the cameras are separated significantly, inter-camera calibration becomes difficult and accuracy is hard to ensure. The use of multiple cameras also reduces the field of view because usually only features in the intersection areas are processed.

2.1.2 Monocular Camera

The simplest setup from a hardware's perspective is to use a monocular camera [13, 24, 25, 29, 54, 68, 78]. If the camera motion is unconstrained and is used alone without assistance

from other sensors, in general, the ego-motion estimation cannot solve the scale ambiguity. If certain constraint or assumption about the camera motion can be established, scale ambiguity can possibly be fixed. For example, with a front-looking camera mounted to a ground vehicle, Kitt et al.'s method determines scale by using a nonholonomic motion model and assuming the vehicle drives on a planar ground surface [53]. Nourani-Vatani and Borges adopt a nonholonomic motion model along with a downward-looking camera to estimate the planar motion of a ground vehicle [72]. Since the method only estimates the ego-motion in 3-DOF, an INS system is used to obtain the vehicle inclination. Scaramuzza et al.'s approach [83] employs an omnidirectional camera , where a nonholonomic motion model and steering encoder readings are used as problem constraints. This approach can recover motion at a low computational cost with a single visual feature, and shows significantly improved accuracy compared to the cases with unconstrained motion. Scaramuzza also shows that a monocular camera placed with an offset to the vehicle rotation center can solve scale when the vehicle is turning [82]. During straight driving, however, the formulation degenerates and scale is unsolvable.

On the other hand, if certain knowledge about the environment is available, it can also help fix the scale ambiguity. For example, Artieda et al's visual SLAM method uses a front-looking camera mounted on an aerial vehicle [4]. Scale is solved by associating certain visual features with known 3D coordinates. Conte and Doherty's visual navigation system works for flights at high altitudes such that the ground can be considered flat and level [14]. The vehicle motion is computed using planar homography with imagery of the ground. The method also matches against geo-referenced aerial images to fix estimation drift. Caballero et al.'s visual odometry method also uses planar homography assuming flat ground [10]. However, the method does not require the ground to be level and computes the ground inclination. Scale is solved by the aircraft hight above the ground measured with a range sensor.

Another choice is to use an IMU with a camera [57]. The IMU acceleration readings help determine scale. The state of the art along this direction tightly couples inertial measurements from an IMU with camera imagery in ego-motion estimation [48, 58]. Alternatively, Weiss et al. loosely couples an IMU with an independent visual odometry method, parallel tracking and mapping (PTAM) [54], in a Kalman filter. The system simultaneously estimates the IMU biases and extrinsic parameters between the IMU and camera.

2.1.3 RGB-D Camera

The introduction of RGB-D cameras has drawn great attention in the research comunity [20, 26, 67, 96]. RGB-D cameras deliver visual images with depth information associated to image pixels, providing an easy way to determine scale. Huang et al. use tracked visual features with depth from an RGB-D camera to estimate the ego-motion [47]. The method eliminates visual features if the corresponding depth is unavailable from depth images. Henry et al. [39] employ both 2D and 3D feature matching metricses in the ego-motion estimation. The method employs a joint optimization minimizing combined 2D and 3D feature matching errors. On the other hand, dense tracking is another popular direction in the RGB-D visual odometry literature [52, 86]. These methods minimize the photometric error using a dense 3D model of the environment provided from the depth images. The technology is applied to monocular visual odometry as well, using dense [68] and semi-dense [24, 25] 3D models.

Table 2.1: Comparsion of vision-based approaches.

Camera	Additional Constrains	Solve Scale	Existing Literature	**Ours**
Stereo	N/A	Yes	[15, 35, 45, 56, 62, 70]	×
Monocular	N/A	No	[13, 25, 29, 54, 68, 78]	
	Nonholonomic constraint of a ground vehicle	Yes	[53, 72, 82, 83]	
	Hight above ground of an aerial vehicle	Yes	[10, 14]	
	Integration with an IMU	Yes	[48, 57, 58, 95]	×
	Integration with a lidar	Yes	N/A	×
RGB-D	N/A	Yes	[39, 46, 47, 52, 86]	×

2.1.4 Summary

The vision-based methods are summarized in Table 2.1. The methods proposed in this thesis fall in four categories – it utilizes visual images with additionally provided depth information, and couples an IMU with a camera. In contrast to existing RGB-D visual odometry methods [39, 46, 47, 52, 86], the proposed method can handle sparse depth information by involving both features with and without depth. Further, the method maintains a depthmap and associates depth to visual features from the depthmap. Hence, it is not limited to RGB-D cameras but can utilize depth information from different sources. The method supports depth from lidar range measurements as well as stereo camera reconstructions.

2.2 Lidar-based Approaches

2.2.1 Scanning from Standing

Lidar measures distance by illuminating object with laser and analyzing the reflected light [11]. For most lidars, at an instant time, only a single or a limited number of laser beams are projected into the environment. The sensor realizes its complete field of view by rotating the laser beams and reporting the rotation angles. As a result, laser points are perceived continuously over time. Each laser point is associated with a unique timestamp.

If the lidar is stationary without any external motion during the course of scanning – rotation of the laser beams is the only motion involved, the laser points can be projected based on the rotation angles of the laser beams to produce a scan of the environment. If the process contains multiple scans conducted in such a way at different locations, each scan can be converted into a common coordinate system using a rigid body transform. In fact, many industrial surveys are conducted this way, with artificial markers setup in the environment to provide the transforms. Variations of the iterative closest point (ICP) method [77, 81] and the normal distribution transform (NDT) method [6, 92] can also be use to align the scans if coarse transforms among the scans are provided. The standard ICP method iteratively finds point-to-point correspondences and minimizes the Euclidean distances in point pairs. Its variants adopt different types of correspondences such as point-to-plane correspondences and several methods for nonlinear optimiza-

tion. The NDT method considers scans as being consisted of point clusters following normal distributions and correlates bewteen point clusters for scan matching. Additionally, methods [23, 27, 43, 85] can be used to match lidar scans with relaxed requirements on the initial guess. Need to mention, if a vehicle carrying a lidar stops whenever the lidar collects data, the case falls in the same category. Also, if odometry information is available on the vehicle, it can be used to provide coarse transforms to initialize scan matching.

2.2.2 Scanning from Moving

In the case that the lidar moves during the course of scanning, such as mounted to a driving vehicle or carried by a person walking in the environment to be mapped, the problem becomes difficult. This is because the lidar external motion is registered in the scans – a scan cannot be consider as a rigid body but motion distortion is contained. If the lidar scan rate is fast compared to its external motion, the motion distortion can be neglectable [55]. The scan matching methods in the previous section are still valid. On the other hand, if the scan rate is mediate, a two step method can possibaly compensate for the motion distortion [41, 65]: a scan matching step is followed by a distortion removal step using velocity estimated in the first step, assuming constant velocity during a scan. The hardest case is that the scan rate is slow, where motion distortion can be severe. This is especially the case of a 2-axis lidar since the rotation around one axis is typically much slower than the other axis. Often, other sensors are used to provide velocity measurements, with which, the motion distortion can be removed [73]. For example, in Scherer et al's work, laser points are registered by ego-motion estimation from the visual-inertial odometry on an aerial vehicle [84]. Also taking visual odometry estimation, Droeschel et al's method [19] and Holz and Behnke's method [40] further match lidar scans to refine the motion. When multiple sensors such as wheel encoders and IMUs are available, the problem is typically solved as a simultaneous localization and mapping (SLAM) problem [21, 22], using Kalman filers [64], particle filters [88], or factor-graph optimization [49, 51].

Ego-motion estimation is possible with a slow scanning lidar itself. In this case, ego-motion estimation and motion distortion correction become one problem. A method by Barfoot et al. is to create visual images from laser intensity returns and match visual features [5] to recover the motion of a ground vehicle [2, 3, 18, 90]. The vehicle motion is modeled with constant velocity in [3, 18] and Gaussian processes in [2, 90]. The method in [3, 18] is similar to ours in that both use a linear motion model in ego-motion estimation. However, methods [2, 3, 18, 90] adopt visual features and require dense scan data. The scan matching in this thesis extracts gemoetrical features and therefore does not require scans to be dense.

Another approach is from Bosse and Zlot [7, 8, 103]. They use a 2-axis lidar to obtain maps registered by matching geometric structures of local point clusters [7]. Further, they use multiple 2-axis lidars to map an underground mine [103]. This method incorporates an IMU and uses loop closures to create large maps. Proposed by the same authors, Zebedee is a mapping device composed of a 2D lidar and an IMU attached to a hand-bar through a spring [8]. Mapping is conducted by hand-nodding the device. The ego-motion is computed by a batch optimization method that processes segmented data with boundary constraints between the segments. In this method, IMU measurements are used to register laser points and the optimization-based scan matching corrects the IMU biases. In essence, Bosse and Zlot's methods require batch processing

Table 2.2: Comparison of lidar-based approaches.

Lidar	Scan rate	Distortion	Approaches	Real-time	Feature
Standing	Any	N/A	ICP [77, 81], NDT [6, 92]	Yes/No	Geometrical
Moving	Fast	Neglectable	ICP [77, 81], NDT [6, 92]	Yes	Geometrical
	Mediate	Small	Two-step method [41, 65]	Yes	Geometrical
	Slow	Large	Registered by external sensoring [84]	Yes	N/A
			Barfoot et al. [2, 3, 18, 90]	Yes	Visual
			Bosse and Zlot [7, 8, 103]	No	Geometrical
			Ours	Yes	Geometrical

to develop accurate maps. Hence, the methods are unsuitable for applications where the ego-motion estimation and mapping are needed in real-time.

2.2.3 Summary

The lidar-based methods are summarized in Table 2.2. The method proposed in this thesis employs milti-layer optimization in processing lidar data – the first layer estimates velocities of the sensor and removes motion distortion in the scans, while the second layer further matches the scans to build a map of the traversed environment. The method can work with slow scanning lidars. Also, the real-time generated ego-motion estimates and maps are useful in autonomous navigation systems for vehicle control and path planning.

Chapter 3

Depth Enhanced Monocular Odometry

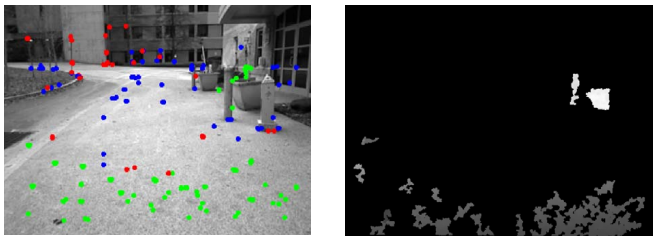
Visual odometry can be augmented by depth information such as provided by RGB-D cameras, or from lidars associated with cameras. However, such depth information can be limited by the sensors, leaving large areas in the visual images where depth is unavailable. Here, I propose a method to utilize the depth, even if sparsely available, in recovery of camera motion. In addition, the method utilizes depth by structure from motion using the previously estimated motion, and salient visual features for which depth is unavailable. Therefore, the method is able to extend RGB-D visual odometry to large scale, open environments where depth often cannot be sufficiently acquired. The method is validated in three sensor setups, one using an RGB-D camera, and two using combinations of a camera and a 3D lidar. Evaluated on the KITTI odometry benchmark, the method produces an average position error as 1.14% of the distance traveled.

3.1 Introduction

Visual odometry is the process of egomotion estimation given a sequence of camera imagery. Typically, monocular imagery is insufficient to compute the egomotion because motion along the camera optical axis can cause little motion of visual features and therefore the estimation problem can be degenerate. With a single camera [25, 29, 54, 68], if assuming unconstrained motion, rotation can be recovered but translation is up to scale. This situation can be mitigated by using extra information such as knowledge of a non-holonomic motion constraint [83], or measurements from an IMU integrated with the visual odometry [95]. However, the results are dependent on the quality of the extra information involved.

It is possible to obtain scale by using multiple cameras simultaneously [15, 56]. However, this comes at its own cost–reduced effective field of view and a limitation on the range that can be accurately recovered from the multiple cameras. If a small baseline is used, depth is uncertain for features far away from the camera. But, if the cameras are separated significantly, inter-camera calibration becomes difficult and accuracy can be hard to ensure. When used in scenes where a large difference exists between near and far objects, depth can only be obtained in certain parts of the images. Effectively, only near features are used in recovery of the motion.

This chapter proposes a method, namely DEMO, that can effectively utilize depth information along with the imagery. When used in scenes where a large difference exists between near



(a) (b)

Figure 3.1: (a) Features tracked at an image frame. The green dots represent features whose depth comes from the depth map, the blue dots represent features whose depth is determined by triangulation using the previously estimated motion of the camera, and the red dots represent features without depth. The proposed method uses all three types of features in determining motion. (b) A depth image from an RGB-D camera corresponding to (a), where depth information is only available in the vicinity of the camera. The gray and white pixels represent close and far objects with respect to the camera, and the black pixels are areas that depth is unavailable.

and far objects, depth can only be obtained in certain parts of the images. The method handles exactly this scenario – enhance motion estimation through usage of any depth information available. Hence, I extend RGB-D visual odometry to large scale, open environments where depth often cannot be sufficiently acquired. The method maintains and registers a depth map using the estimated motion of the camera. Visual features are associated with depth from the depth map, or by structure from motion using the previously estimated motion. With depth associated, I derive objective functions that minimize the distances from the features to their lateral projections onto the rays representing the features tracked at a consecutive frame. This makes the method extra sensitive to features at large angles off the camera periapical axis. Salient visual features for which depth is unavailable are also used, which provide different constraints in solving the problem. Further, the method contains a bundle adjustment step which refines the motion estimates in parallel by processing a sequence of images, in a batch optimization.

The proposed method is not limited to RGB-D cameras. It can be adapted to various types of cameras as long as depth information can be acquired and associated. I have collected experimental data using an RGB-D camera and a custom-built sensor consisting a camera and 3D lidar (a 2-axis laser scanner). I have also evaluated the method using the well-known KITTI benchmark datasets [33, 36], which contain carefully registered data from a number of sensors and ground truth from a high-accuracy GPS/INS. The method reported here uses images from a single camera in a stereo pair and laser scans from a high rate lidar.

3.2 Notations and Task Description

The visual odometry problem addressed in this chapter is to estimate the motion of a camera using monocular images with assistance of depth information. I assume that the camera is well modeled as a pinhole camera [38], the camera intrinsic parameters are known from pre-calibration, and the lens distortion is removed. As a convention in this chapter, I use right superscript k, $k \in Z^+$ to indicate image frames. Define camera coordinate system, $\{C\}$, as,

- $\{C\}$ is a 3D coordinate system with its origin at the camera optical center. The x-axis points to the left, the y-axis points upward, and the z-axis points forward coinciding with the camera principal axis.

I would like to utilize features with and without depth. Let $\mathcal{I}$ be a set of feature points. For a feature i, $i \in \mathcal{I}$, that is associated with depth, its coordinates in $\{C^k\}$ are denoted as $\boldsymbol{X}_i^k$, where $\boldsymbol{X}_i^k = [x_i^k,\ y_i^k,\ z_i^k]^T$. For a feature with unknown depth, I normalize the coordinates such that its z-coordinate is one. Let $\bar{\boldsymbol{X}}_i^k$ be the normalized term of the feature, $\bar{\boldsymbol{X}}_i^k = [\bar{x}_i^k,\ \bar{y}_i^k,\ 1]^T$. Intuitively, imagine a plane that is parallel to the $x - y$ plane of $\{C^k\}$ and at a unit length in front of the coordinate origin, and $\bar{\boldsymbol{X}}_i^k$ is the point projected onto the plane. With notations defined, the visual odometry problem can be described as

Problem: Given a sequence of image frames k, $k \in Z^+$, and features, $\mathcal{I}$, compute the motion of the camera between each two consecutive frames, k and $k-1$, using $\boldsymbol{X}_i^k$, if the depth is available, and $\bar{\boldsymbol{X}}_i^k$, if the depth is unknown, $i \in \mathcal{I}$.

3.3 System Overview

3.3.1 Sensor Hardware

The proposed method is validated on, but not limited to, three different sensor systems. The first two sensors shown in Fig. 3.2 are used to acquire author-collected data, while the third one uses configuration of the KITTI benchmark datasets. Through the chapter, I will use data from the first two sensors to illustrate the method. Fig. 3.2(a) is an Xtion Pro Live RGB-D camera. The camera is capable of providing RGB and depth images at 30Hz, with 640×480 resolution and 58° horizontal field of view. Fig. 3.2(b) shows a custom-built camera and 3D lidar. The camera can provide RGB images up to 60Hz, with 744×480 resolution and 83° horizontal field of view. The 3D lidar is based on a Hokuyo UTM-30LX laser scanner, which has 180° field of view with 0.25° resolution and 40 lines/sec scanning rate. The laser scanner is actuated by a motor for rotational motion to realize 3D scanning.

(a) (b)

Figure 3.2: Two sensors involved in the evaluation. (a) An Xtion Pro Live RGB-D camera. The camera is capable of providing 30Hz RGB and depth images, with 640×480 resolution and 58° HFV. (b) A custom-built camera and 3D lidar. The camera provides up to 60Hz RGB images with 744×480 resolution and 83° HFV. The 3D lidar consists of a Hokuyo UTM-30LX laser scanner rotated by a motor to realize 3D scan. The laser scanner has 180° FOV with 0.25° resolution and 40 lines/sec scanning rate.

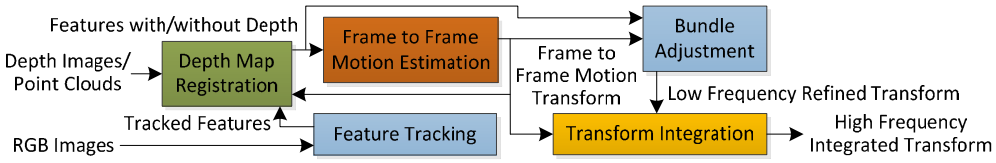


Figure 3.3: Block diagram of the visual odometry software system.

3.3.2 Software System Overview

Fig. 3.3 shows a diagram of the software system. First, visual features are tracked by the feature tracking block. Depth images from RGB-D cameras or point clouds from lidars are registered by the depth map registration block, using the estimated motion. The block also associates depth for the visual features. The frame to frame motion estimation block takes the features as the input, and its output is refined by the bundle adjustment block using sequences of images. The bundle adjustment runs at a low frequency (around 0.25-1.0Hz). The transform integration block combines the high frequency frame to frame motion with the low frequency refined motion, and generates integrated motion transforms at the same frequency as the frame to frame motion transforms. Section 3.4 and 3.5 present each block in detail.

3.4 Frame to Frame Motion Estimation

3.4.1 Mathematical Derivation

Let us start with mathematical derivation for the frame to frame motion estimation. The corresponding algorithm is discussed in the next section. Recall that $\boldsymbol{X}_i^{k-1}$ and $\boldsymbol{X}_i^k$ are the coordinates of a tracked feature in $\{C^{k-1}\}$ and $\{C^k\}$. Define $\mathbf{R}$ and $\boldsymbol{T}$ as the 3×3 rotation matrix and 3×1 translation vector of the camera between the two frames, I model the camera motion as rigid body transformation,

$$\boldsymbol{X}_i^k = \mathbf{R}\boldsymbol{X}_i^{k-1} + \boldsymbol{T}. \tag{3.1}$$

Next, I will work on features with known depth. Acquiring depth for the features will be discussed in the later part of this chapter. Here, note that depth is only associated for one of the two frames – frame $k-1$. This is because the depth maps are registered in the camera coordinates by the estimated motion. By the time of frame k, the depth map at frame $k-1$ is available and the depth of $\boldsymbol{X}_i^{k-1}$ can be associated. However, the depth map is unavailable to frame k as the last frame to frame motion has not been computed. This is especially true for the sensor configuration in Fig. 3.2(b). Since laser points are perceived continuously over time, estimated motion is needed to register the points on the depth map. Further, the observation is that depth maps change little between consecutive frames and using that of one frame is sufficient. Recall that $\bar{\boldsymbol{X}}_i^k$ is the normalized term of $\boldsymbol{X}_i^k$, where the z-coordinate is one, (3.1) is rewritten as

$$z_i^k \bar{\boldsymbol{X}}_i^k = \mathbf{R}\boldsymbol{X}_i^{k-1} + \boldsymbol{T}, \tag{3.2}$$

where $\mathbf{R}$ and $\boldsymbol{T}$ form an $\mathbf{SE}(3)$ transformation [66]. Eq. (3.2) contains three rows. Combining the 1st and 2nd rows with the 3rd row, respectively, I eliminate z_i^k as the unknown depth. This

gives us two equations as follows,

$$(\boldsymbol{R}_1 - \bar{x}_i^k \boldsymbol{R}_3)\boldsymbol{X}_i^{k-1} + T_1 - \bar{x}_i^k T_3 = 0, \tag{3.3}$$

$$(\boldsymbol{R}_2 - \bar{y}_i^k \boldsymbol{R}_3)\boldsymbol{X}_i^{k-1} + T_2 - \bar{y}_i^k T_3 = 0, \tag{3.4}$$

where $\boldsymbol{R}_h$ and T_h, $h \in \{1, 2, 3\}$ are the h-th rows of $\mathbf{R}$ and $\boldsymbol{T}$, respectively.

For a feature with unknown depth, (3.1) is rewritten as the following. Here, $\bar{\boldsymbol{X}}_i^{k-1}$ is the normalized term of $\boldsymbol{X}_i^{k-1}$,

$$z_i^k \bar{\boldsymbol{X}}_i^k = z_i^{k-1} \mathbf{R} \bar{\boldsymbol{X}}_i^{k-1} + \boldsymbol{T}. \tag{3.5}$$

Eq. (3.5) also contains three rows. Combining all rows to eliminate both z_i^k and z_i^{k-1}, I obtain,

$$[-\bar{y}_i^k T_3 + T_2,\ \bar{x}_i^k T_3 - T_1,\ -\bar{x}_i^k T_2 + \bar{y}_i^k T_1]\mathbf{R}\bar{\boldsymbol{X}}_i^{k-1} = 0. \tag{3.6}$$

So far, I have modeled the frame to frame motion for features with and without depth separately. Now, I will solve the motion using both types of features. Define θ as a 3×1 vector, $\theta = [\theta_x,\ \theta_y,\ \theta_z]^T$, where the normalized term $\theta / ||\theta||$ represents direction of the rotation axis and $||\theta||$ is the rotation angle between frames k and $k-1$. The rotation matrix $\mathbf{R}$ can be expressed by exponential map through Rodrigues formula [66],

$$\mathbf{R} = e^{\hat{\theta}} = \begin{cases} \mathbf{I} + \frac{\hat{\theta}}{||\theta||}\sin||\theta|| + \frac{\hat{\theta}^2}{||\theta||^2}(1 - \cos||\theta||) & \text{if } \theta \text{ is not a small angle,} \\ \mathbf{I} + \hat{\theta} + \frac{1}{2}\hat{\theta}^2 & \text{otherwise,} \end{cases} \tag{3.7}$$

where $\hat{\theta}$ is the skew symmetric matrix of θ. Here, note that when θ is a small angle, take the second-order Taylor expansion of the Rodrigues formula in (3.7) to avoid the problem of dividing by zero.

Substituting (3.7) into (3.3)-(3.4), I can derive two equations for a feature with depth, and substituting (3.7) into (3.6), I can derive one equation for a feature with unknown depth. Each equation is a function of θ and $\boldsymbol{T}$. Suppose a total of m and n features with known and unknown depth. Stacking the equations, I obtain a nonlinear function,

$$\boldsymbol{f}([\boldsymbol{T};\ \theta]) = \epsilon, \tag{3.8}$$

where $\boldsymbol{f}$ has $2m + n$ rows, ϵ is a $(2m + n) \times 1$ vector containing the residuals, and $[\boldsymbol{T};\ \theta]$ is the vertical joining of the vectors. Compute the Jacobian matrix of $\boldsymbol{f}$ with respect to $[\boldsymbol{T};\ \theta]$, denoted as $\mathbf{J}$, where $\mathbf{J} = \partial f / \partial [\boldsymbol{T};\ \theta]$. (3.8) can be solved by the Levenberg-Marquardt (LM) method [38],

$$[\boldsymbol{T};\ \theta]^T \leftarrow [\boldsymbol{T};\ \theta]^T - (\mathbf{J}^T\mathbf{J} + \lambda \mathrm{diag}(\mathbf{J}^{\mathrm{T}}\mathbf{J}))^{-1}\mathbf{J}^{\mathrm{T}}\epsilon. \tag{3.9}$$

Here, λ is a scale factor determined by the LM method.

The proposed method is a sparse feature based method. Features are tracked regardless of depth information being available. The tracked features are then formulated into different equations according to the depth information. In comparison, direct methods such as [52, 86] employ overall image warping in the depth available areas, eliminating the need to process individual features. However, as discussed in [29], sparse feature based methods can produce better accuracy. The method [29], therefore, performs direct image matching to generate initial guess followed by sparse motion estimation to warrant accuracy.

3.4.2 Motion Estimation Algorithm

The frame to frame motion estimation algorithm is presented in Algorithm 1. As I have mentioned that the proposed method only uses depth information associated at frame $k-1$, all features taken as the input at frame k are without depth, as $\bar{\boldsymbol{X}}_i^k$. Features at frame $k-1$ are separated into $\boldsymbol{X}_i^{k-1}$ and $\bar{\boldsymbol{X}}_i^{k-1}$ for those with and without depth. On line 8, a feature with depth contributes two equations to the nonlinear function (3.8), and on line 12, a feature with unknown depth contributes one equation.

A constant velocity model is used for initialization of θ and $\boldsymbol{T}$ on line 4. Optionally, the terms are initialized to zero at the first frame. The algorithm is adapted to a robust fitting framework [1]. On line 16, the algorithm assigns a bisquare weight for each feature based on their residuals in (3.8). Features that have larger residuals are assigned with smaller weights, and features with residuals larger than a threshold are considered as outliers and assigned with zero weights. On line 17, θ and $\boldsymbol{T}$ are updated for one iteration. The nonlinear optimization terminates if convergence is found, or the maximum iteration number is met. The nonlinear optimization also terminates if the ratio of the inlier features is lower than a threshold, which lead to neglecting of the current frame. Finally, the algorithm returns the motion estimation θ and $\boldsymbol{T}$.

3.4.3 Feature Depth Association

In this section, I will discuss how to associate depth to the visual features. As illustrated in Fig. 3.4, a depth map is registered by the estimated motion of the camera. The depth map is projected to the last image frame whose transform to the previous frame is established. Here, I use frame $k-1$ to keep the same convention with the previous sections.

New points are added to the depth map upon receiving from depth images or point clouds. Only points in front of the camera are kept, and points that are received a certain time ago are forgotten. Then, the depth map is downsized to maintain a constant point density. I would like to keep an even angular interval among the points viewed from the origin of $\{C^{k-1}\}$, or the optical center of the camera at frame $k-1$. Here, we choose angular interval over Euclidean distance interval with the consideration that an image pixel represents an angular interval projected into the 3D environment. We use the same format for the depth map.

The map points are converted into a spherical coordinate system coinciding with $\{C^{k-1}\}$. A point is represented by its radial distance, azimuthal angle, and polar angle. When downsizing, only the two angular coordinates are considered, and the points are evenly distributed with respect to the angular coordinates. This results in a denser point distribution that is closer to the camera, and vice versa. An example of a registered depth map is shown in Fig. 3.5, color coded by elevation. The point clouds are collected by the lidar in Fig. 3.2(b) pointing to a wall.

To associate depth to the visual features, I store the depth map in a 2D KD-tree [16] based on the two angular coordinates. As illustrated in Fig. 3.6, this equals to projecting points on the depth map onto a sphere with a unit distance to the camera center. For each feature i, $i \in \mathcal{I}$, as illustrated by the orange point, I find three points from the KD-tree that are the closest to the feature. The three points form a local planar patch in the 3D environment, and the 3D coordinates of the feature are found by projecting onto the planar patch. Denote $\hat{\boldsymbol{X}}_j^{k-1}$, $j \in \{1, 2, 3\}$ as the Euclidean coordinates of the three points in $\{C^{k-1}\}$, and recall that $\boldsymbol{X}_i^{k-1}$ is the coordinates of

Algorithm 1: Frame to Frame Motion Estimation

input : $\bar{\boldsymbol{X}}_i^k$, $\boldsymbol{X}_i^{k-1}$ or $\bar{\boldsymbol{X}}_i^{k-1}$, $i \in \mathcal{I}$
output : θ, $\boldsymbol{T}$
begin
  $\theta, \boldsymbol{T} \leftarrow$ initialization based on a constant velocity model or zero at the first frame;
  for a number of iterations **do**
    for each $i \in \mathcal{I}$ **do**
      if i is depth associated **then**
        Derive (3.3)-(3.4) using $\bar{\boldsymbol{X}}_i^k$ and $\boldsymbol{X}_i^{k-1}$ as functions of θ and $\boldsymbol{T}$,
        $(3.3) : (\boldsymbol{R}_1 - \bar{x}_i^k \boldsymbol{R}_3)\boldsymbol{X}_i^{k-1} + T_1 - \bar{x}_i^k T_3 = 0,$
        $(3.4) : (\boldsymbol{R}_2 - \bar{y}_i^k \boldsymbol{R}_3)\boldsymbol{X}_i^{k-1} + T_2 - \bar{y}_i^k T_3 = 0;$
        Stack (3.3)-(3.4) into $(3.8) : \boldsymbol{f}([\boldsymbol{T};\ \theta]) = \epsilon;$
      end
      else
        Derive (3.6) using $\bar{\boldsymbol{X}}_i^k$ and $\bar{\boldsymbol{X}}_i^{k-1}$ as a function of θ and $\boldsymbol{T}$,
        $(3.6) : [-\bar{y}_i^k T_3 + T_2,\ \bar{x}_i^k T_3 - T_1,\ -\bar{x}_i^k T_2 + \bar{y}_i^k T_1] \qquad \mathbf{R}\bar{\boldsymbol{X}}_i^{k-1} = 0;$
        Stack (3.6) into $(3.8) : \boldsymbol{f}([\boldsymbol{T};\ \theta]) = \epsilon;$
      end
    end
    Compute a bisquare weight for each feature based on the residuals in $(3.8) : \boldsymbol{f}([\boldsymbol{T};\ \theta]) = \epsilon;$
    Update θ, $\boldsymbol{T}$ for one iteration based on,
    $(3.9) : [\boldsymbol{T};\ \theta]^T \leftarrow [\boldsymbol{T};\ \theta]^T - (\mathbf{J}^T\mathbf{J} + \lambda \mathrm{diag}(\mathbf{J}^{\mathrm{T}}\mathbf{J}))^{-1}\mathbf{J}^{\mathrm{T}}\epsilon$
    if the inlier ratio is smaller than a threshold **then**
      Report the frame is skipped and return;
    end
    if the nonlinear optimization converges **then**
      Break;
    end
  end
  Return θ, $\boldsymbol{T}$;
end

feature i in $\{C^{k-1}\}$. The depth is computed by solving a function as follows,

$$(\boldsymbol{X}_i^{k-1} - \hat{\boldsymbol{X}}_1^{k-1})((\hat{\boldsymbol{X}}_1^{k-1} - \hat{\boldsymbol{X}}_2^{k-1}) \times (\hat{\boldsymbol{X}}_1^{k-1} - \hat{\boldsymbol{X}}_3^{k-1})) = 0. \tag{3.10}$$

Further, if the depth is unavailable from the depth map for some features but they are tracked for longer than a certain distance in the Euclidean space, I triangulate the features using the image sequences where the features are tracked. This uses a similar procedure as [29, 94], where the depth of a feature is updated based on a Bayesian probabilistic model each time the feature is tracked for one more frame. Fig. 3.7 gives an example of depth associated features, corresponding to Fig. 3.1. The white dots are points on the depth map, only available within a limited range. The green dots represent features whose depth are provided by the depth map, and the blue dots are from structure from motion.

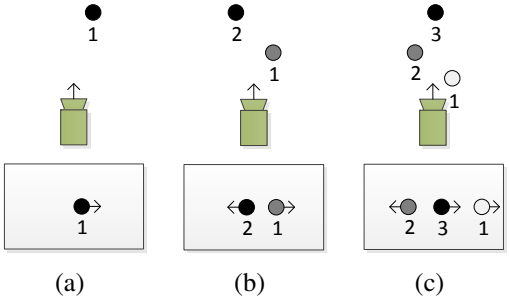


Figure 3.4: Illustration of depth map registration. (a)-(c) represent three image frames. In each figure, I show the camera frame on the top and the corresponding image on the bottom. Three points are registered labeled with 1-3. The illumination of a point indicates its distance to the camera. The camera moves forward. The registered points change illumination during the camera motion, leave the camera field of view in the end and are removed from the depth map.

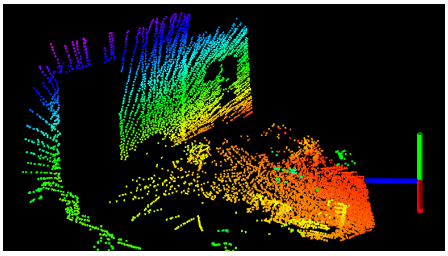


Figure 3.5: An example of the depth map with point clouds perceived by the lidar in Fig. 3.2(b). The points are color coded by elevation. The camera points to a wall during data collection.

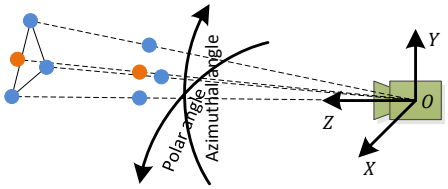


Figure 3.6: Illustration of depth association. Points on the depth map are represented by spherical coordinate representation. The points are stored in a 2D KD-tree based on the azimuthal angle and polar angle as illustrated in the figure. This equals to projecting the points onto a sphere with a unit distance to the camera center. For each feature (indicated by the orange point), I find the three closest points from the KD-tree. The depth of the feature is then interpolated from the three points assuming a local planar patch formed by the three points in the 3D environment.

3.5 Bundle Adjustment

The camera frame to frame motion estimated in the previous section is refined by a bundle adjustment, which takes a sequence of images and performs a batch optimization. As a trade-off between accuracy and processing time, I choose one image out of every five images as the bundle adjustment input. The image sequence contains a number of eight images (taken from 40 original images). This allows the batch optimization to finish before the next image sequence is

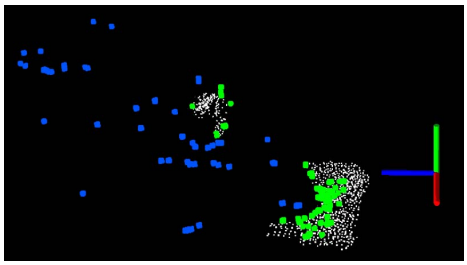


Figure 3.7: Features projected into the 3D environment corresponding to Fig. 3.1. The depth map (white dots) is from depth images collected by the RGB-D camera in Fig. 3.2(a), only available within a limited range. The green dots are features whose depth is provided by the depth map, and the blue dots are from triangulation using the previously estimated motion.

accumulated and ready for processing. The bundle adjustment process is illustrated in Fig. 3.8, using the iSAM [50] open source library. I choose iSAM over other libraries because it supports user-defined camera models, and can conveniently handle both features with and without depth.

In this step, outlier features are already eliminated in the frame to frame motion estimation. All selected inlier features are used in the bundle adjustment. While bundle adjustment itself is time-consuming, skipping outlier removal helps reduce processing time. In the image sequence, the first fame is fixed. The bundle adjustment refines every frame thereafter and the 3D feature poses along the sequence.

Here, define another representation of the features, $\tilde{\boldsymbol{X}}_i^k = [\bar{x}_i^k,\ \bar{y}_i^k,\ z_i^k]^T$, where the x- and y- entries contain the normalized coordinates, and the z-entry contains the depth. For features without depth, z_i^k is set at a default value. Let $\mathcal{J}$ be the set of image frames in the sequence, and let l be the first frame in the set. Upon initialization, all features appear in the sequence are projected into $\{C^l\}$, denoted as $\tilde{\boldsymbol{X}}_i^l$, $i \in \mathcal{I}$. Define $\mathcal{T}_l^j$ as the transform projecting $\tilde{\boldsymbol{X}}_i^l$ from $\{C^l\}$ to $\{C^j\}$, where j is a different frame in the sequence, $j \in \mathcal{J} \backslash \{l\}$. The bundle adjustment minimizes the following function by adjusting the motion transform between each two consecutive frames

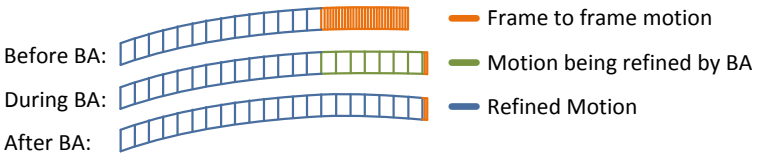


Figure 3.8: Illustration of bundle adjustment. The orange segment represents frame to frame motion, the green segment represents the motion being refined by bundle adjustment, and the blue segment is refined motion. Each vertical bar indicates an image frame. On the top row, the system accumulates frame to frame motion (orange) in front of the refined motion (blue). When enough image frames are accumulated, on the second row, bundle adjustment is executed refining the green segment. After the bundle adjustment, on the third row, the green segment becomes refined motion. The system keeps accumulating frame to frame motion (orange) for the next bundle adjustment to execute.

and the coordinates of $\tilde{\boldsymbol{X}}_i^l$,

$$\min \sum_{i,j} (\mathcal{T}_l^j(\tilde{\boldsymbol{X}}_i^l) - \tilde{\boldsymbol{X}}_i^j)^T \Omega_i^j (\mathcal{T}_l^j(\tilde{\boldsymbol{X}}_i^l) - \tilde{\boldsymbol{X}}_i^j),\ i \in \mathcal{I},\ j \in \mathcal{J} \setminus \{l\}.$$

Here, $\tilde{\boldsymbol{X}}_i^j$ represents the observation of feature i at frame j, and Ω_i^j is its information matrix. The first two entries on the diagonal of Ω_i^j are given constant values representing the feature tracking errors. If the depth is from the depth map which is perceived by a time-of-fly sensor such as the lidar in Fig. 3.2(b), the 3rd entry is set at a fixed value determined by the measurement noise of the sensor. If the depth map is reconstructed by triangulation such as using the RGB-D camera in Fig. 3.2(a), or if the depth is from structure from motion using the estimated motion, the 3rd entry is set to be inversely proportional to the square of the depth. A zero value is used for features with unknown depth. The information matrices associated with the frame to frame poses are set to small and negligible values, meaning that poses from the frame to frame motion estimation only provide initial guess to the bundle adjustment.

The bundle adjustment publishes refined motion transforms at a low frequency. With the camera frame rate between 10-40Hz, the bundle adjustment runs at 0.25-1.0Hz. As illustrated in Fig. 3.8, a transform integration step takes the bundle adjustment output (blue segment) and combines it with the high frequency frame to frame motion estimates (orange segment). The result is integrated motion transforms at the frame to frame motion frequency.

3.6 Experiments

The visual odometry is tested with author-collected data and the KITTI benchmark datasets. It tracks Harris corners [38] by the Kanade Lucas Tomasi (KLT) method [61]. The algorithms run on a laptop computer with 2.5GHz cores and 6GB memory, using around three cores for computation. The feature tracking and bundle adjustment take one core each, and the frame to frame motion estimation and depth map registration together consume another core.

3.6.1 Tests with Author-collected Datasets

I first conduct tests with author-collected datasets using the two sensors in Fig. 3.2. The data is collected from four types of environments shown in Fig. 3.9: a conference room, a large lobby, a clustered road, and a flat lawn. The difficulty increases over the tests as the environments are opener and depth information changes from dense to sparse. Two images are present from each dataset, on the 2nd and 4th rows in Fig. 3.9. The red areas indicate coverage of depth maps, from the RGB-D camera (right figure) and the lidar (left figure). Here, note that the depth map registers depth images from the RGB-D camera or point clouds from the lidar at multiple frames, and usually contains more information than that from a single frame. With the RGB-D camera, the average amount of imaged area covered by the depth map reduces from 94% to 23% over the tests. The lidar has a longer detection range and can provide more depth information in open environments. The depth coverage changes from 89% to 47% of the images.

The camera frame rate is set at 30Hz for both sensors. To evenly distribute the features within the images, I separate an image into 3×5 identical subregions. Each subregion provides

maximally 30 features, giving maximally 450 features in total. The method is compared to two popular RGB-D visual odometry methods. Fovis estimates the motion of the camera by tracking image features, and depth is associated to the features from the depth images [47]. DVO is a dense tracking method that minimizes the photometric error within the overall images [52]. Both methods use data from the RGB-D camera. Our method is separated into two versions, using the two sensors in Fig. 3.2, respectively. The resulting trajectories are presented on the 1st and 3rd rows in Fig. 3.9, and the accuracy is compared in Table 3.1, using errors in 3D coordinates. Here, the camera starts and stops at the same position, and the gap between the two ends of a trajectory compared to the length of the trajectory is considered the relative position error.

From these results, I conclude that all four methods function similarly when depth information is sufficient (in the room environment), while the relative error of DVO is slightly lower than the other methods. However, as the depth information becomes sparser, the performance of Fovis and DVO reduces significantly. During the last two tests, Fovis frequently pauses without giving odometry output due to insufficient number of inlier features. DVO continuously generates output but drifts heavily. This is because both methods use only imaged areas where depth is available, leaving large amount of areas in the visual images being unused. On the other hand, the two versions of our method are able to maintain accuracy in the tests, except that the relative error of the RGB-D camera version is relatively large in the lawn environment, because the depth is too sparse during the turning on top of Fig. 3.9(h).

3.6.2 Tests with KITTI Benchmark Datasets

The proposed method is further tested with the KITTI datasets. The datasets are logged with sensors mounted on the top of a passenger vehicle, in road driving scenarios. As shown in Fig. 3.10, the vehicle is equipped with color stereo cameras, monochrome stereo cameras, a 360° Velodyne laser scanner, and a high accuracy GPS/INS for ground truth. Both image and laser data are logged at 10Hz. The image resolution is around 1230×370 pixels, with 81° horizontal field of view. Our method uses the imagery from the left monochrome camera and the laser data, and tracks maximally 2400 features from 3×10 identical subregions in the images.

The datasets contain 11 tests with the GPS/INS ground truth provided. The data covers mainly three types of environments: "urban" with buildings around, "country" on small roads with vegetations in the scene, and "highway" where roads are wide and the surrounding environment is relatively clean. Fig. 3.11 presents sample results from the three environments. On the top row, the results of the proposed method are compared to the ground truth, with and without using the bundle adjustment. On the middle and bottom rows, an image and the corresponding laser point cloud is presented from each of the three datasets, respectively. The points are color coded by depth. The complete test results with the 11 datasets are listed in Table 3.2. The three tests from left to right in Fig. 3.11 are respectively datasets 0, 3, and 1 in the table. Here, the accuracy is measured by averaging relative translation and rotation errors using segments at 100m, 200m, ..., 800m lengthes, based on 3D coordinates.

Results are analyzed using the overall 11 datasets in Fig. 3.12. On the first row, translation errors are shown as percentages of the traveled distances. Errors are plotted with respect to the lengthes of the data segments in Fig. 3.12(a), the linear speed of the vehicle in Fig. 3.12(b), and the angular speed in Fig. 3.12(c). On the second row, the corresponding rotation errors

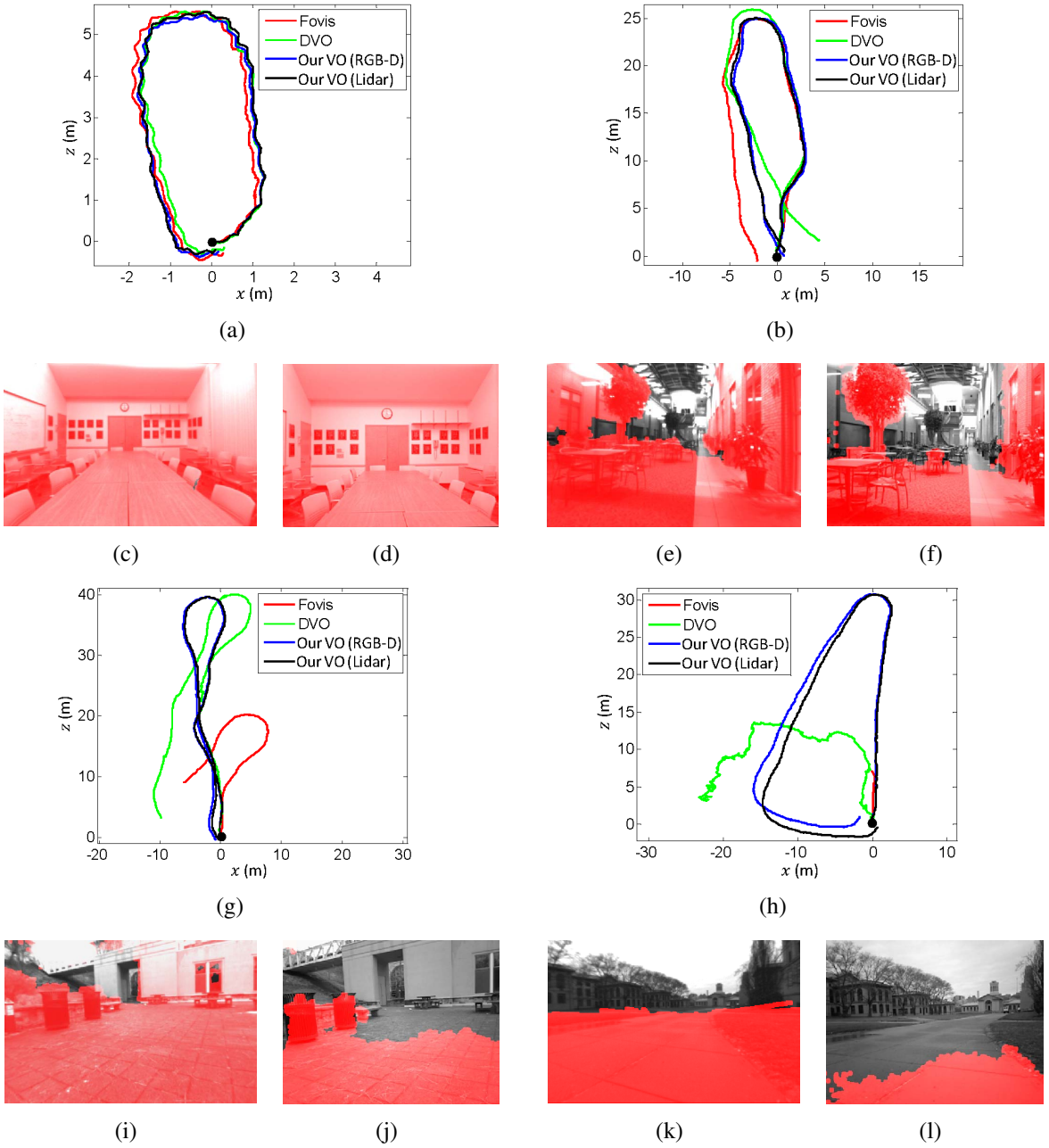


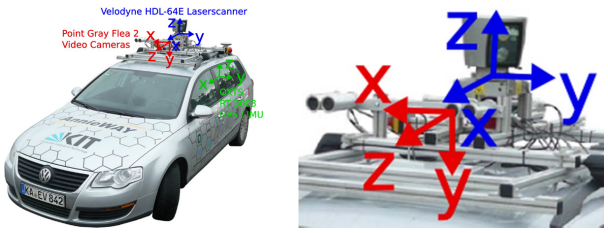
Figure 3.9: Comparison of four methods using author-collected datasets: Fovis, DVO, and two versions of our method using depth from an RGB-D camera and a lidar. The environments are selected respectively from a conference room ((a), (c), and (d)), a large lobby ((b), (e), and (f)), a clustered road ((g), (i), and (j)), and a flat lawn ((h), (k), and (l)). Two images are present from each dataset. The red areas indicate availability of depth maps, from the RGB-D camera (right figure) and the lidar (left figure). The depth information is sparser from each test to the next as the environment becomes opener, resulting in the performance of Fovis and DVO reduces significantly. Our methods relatively keep the accuracy in the tests.

Table 3.1: Results using author-collected data. The error is measured at the end of a trajectory as a % of the distance traveled

Environment	Distance	Relative position error			
		Fovis	DVO	Our VO (RGB-D)	Our VO (Lidar)
Room	16m	2.72%	1.87%	2.14%	2.06%
Lobby	56m	5.56%	8.36%	1.84%	1.79%
Road	87m	13.04%	13.60%	1.53%	0.79%
Lawn	86m	failed	failed	3.72%	1.73%

are present with respect the three terms. And on the third row, distributions of the data used in calculation of the errors are shown. Looking at the left column, one can obversely see that the translation and rotation errors are decreasing functions of the segment lengthes. I explain this as an effect of high frequency noise in the visual odometry output and ground truth. Using the middle row, one perceives a trend that the translation error is an increasing function of the linear speed, however, the rotation error is an decreasing function. The exception is in the left-most part of Fig. 3.12(b), which can possibly be caused by sparsity of data as indicated in the corresponding part of Fig. 3.12(h). It is understandable that the translation error increases as the vehicle drivers faster. The explanation of the fact that the rotation error decreases accordingly is that most rotations happen when the vehicle drives slowly, and the vehicle drives mostly straight at high speeds. In the right column, one sees both translation and rotation errors as increasing functions of the angular speed (again, with an exception in the left most part of Fig. 3.12(c), possibly due to sparsity of data).

Further, to inspect the effect of the bundle adjustment, accuracy of the results are compared in the three environments. The 11 datasets are manually separated into segments and labeled with an environment type. For each environment, the visual odometry is tested with and without the bundle adjustment. Fig. 3.13 shows the distributions of the relative errors. Overall, the bundle adjustment helps reduce the mean errors by 0.3%-0.7%, and seems to be more effective in urban and country scenes than on highways, because the feature quality is lower in the highway scenes.



(a) (b)

Figure 3.10: (a) Vehicle used by the KITTI benchmark for data logging. The vehicle is mounted with color stereo cameras, monochrome stereo cameras, a Velodyne lidar, and a high accuracy GPS/INS for ground truth acquisition. Our method uses data from a single camera and the Velodyne lidar for motion estimation. (b) shows a zoomed in view of the sensors.

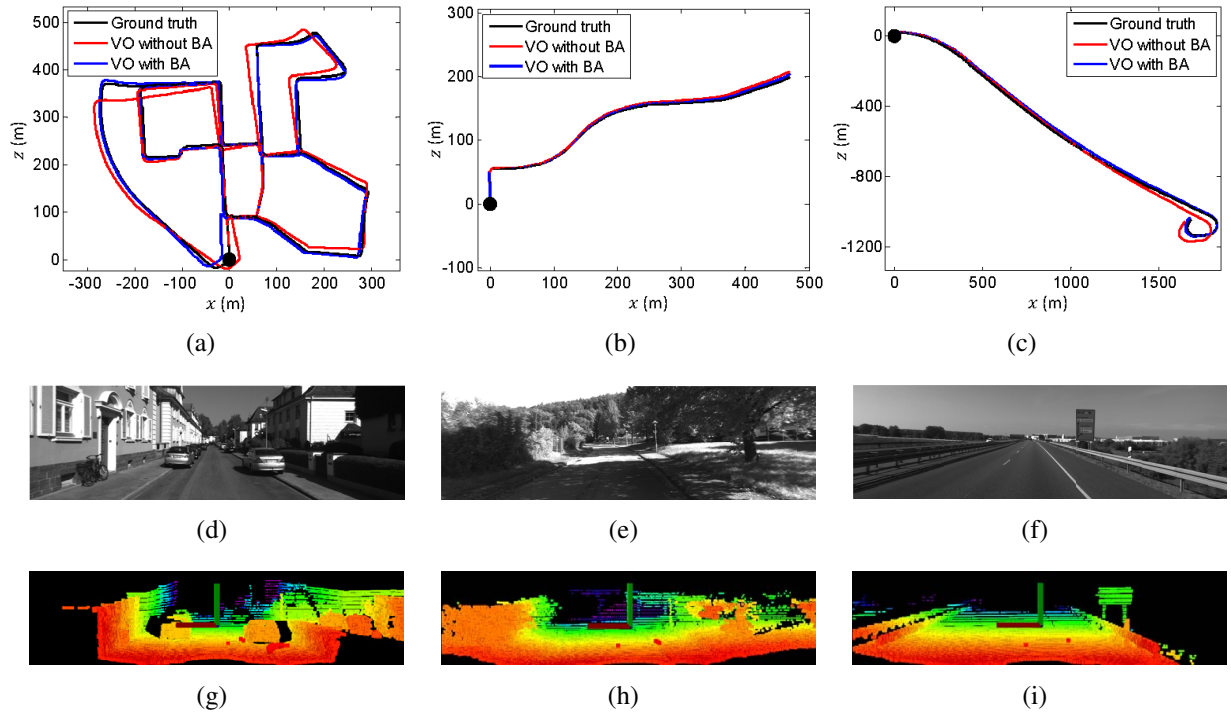


Figure 3.11: Sample results of the proposed method using the KITTI datasets. The datasets are chosen from three types of environments: urban, country, and highway from left to right. In (a)-(c), results are compared with and without the bundle adjustment, to the GPS/INS ground truth. The black dots are the starting points. An image is shown from each dataset to illustrate the three environments, in (d)-(f), and the corresponding laser point cloud in (g)-(i). The points are coded by depth, where red color indicates near objects and blue color indicates far objects.

Table 3.2: Configurations and results of the KITTI datasets. The translation error is calculated using segments of a trajectory at 100m, 200m, ..., 800m lengthes, as an averaged % of the segment lengthes based on 3D coordinates.

Data no.	Configuration		Error	
	Distance	Environment	Trans.	Rotation
0	3714m	Urban	1.05%	0.0029°/s
1	4268m	Highway	1.87%	0.0026°/s
2	5075m	Urban + Country	0.93%	0.0038°/s
3	563m	Country	0.99%	0.0043°/s
4	397m	Country	1.23%	0.0048°/s
5	2223m	Urban	1.04%	0.0034°/s
6	1239m	Urban	0.96%	0.0029°/s
7	695m	Urban	1.16%	0.0043°/s
8	3225m	Urban + Country	1.24%	0.0047°/s
9	1717m	Urban + Country	1.17%	0.0035°/s
10	919m	Urban + Country	1.14%	0.0064°/s

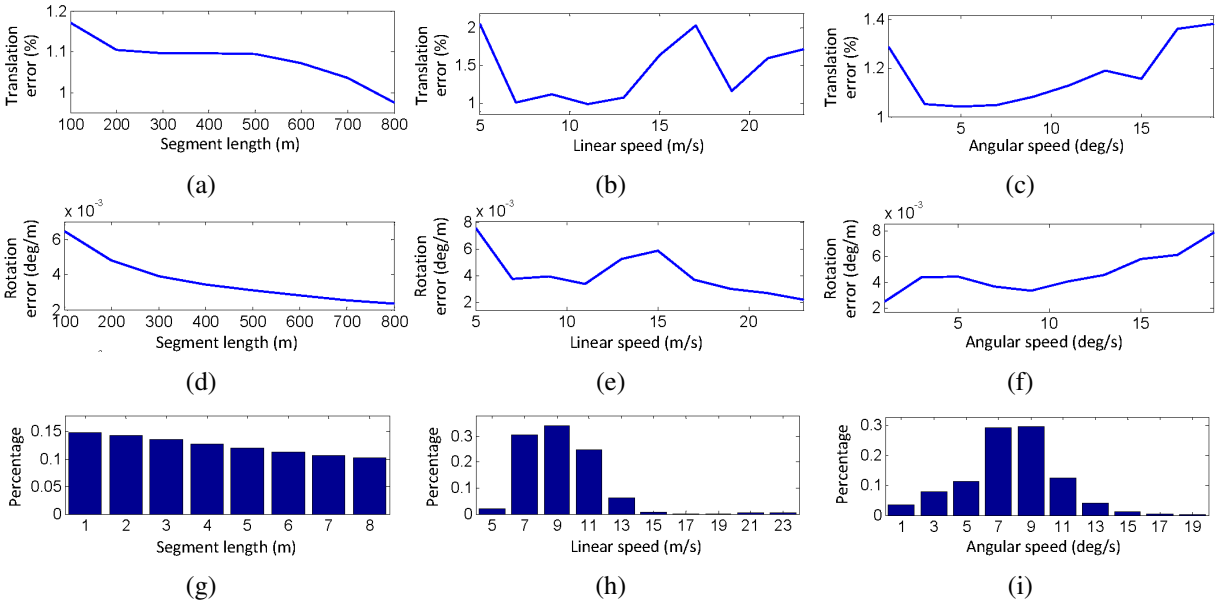


Figure 3.12: Analysis of accuracy with the KITTI datasets. On the top and middle rows, translation and rotation errors are shown, by averaging errors in the 11 datasets listed in Table 3.2. The translation errors are represented as fractions of the distance traveled, using data segments in 100m, 200m,..., 800m lengthes based on 3D coordinates. On the bottom row, distributions of the data used in calculation of the errors are present. In the left, middle, and right columns, the errors are shown with respect to the segment length, linear speed, and angular speed, respectively.

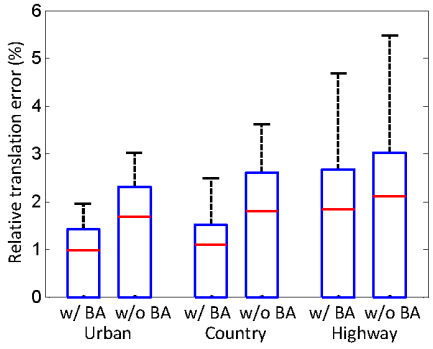


Figure 3.13: Comparison of relative translation errors in urban, country, and highway environments, tested with the KITTI datasets. In each environment, I compare errors with and without the bundle adjustment. The black, blue, and red lines indicate 100%, 75%, and median errors.

3.7 Conclusion

The scenario of insufficiency in depth information is common for RGB-D cameras and lidars which have limited ranges. Without sufficient depth, solving the visual odometry is hard. The method handles the problem by exploring both visual features whose depth is available and unknown. The depth is associated to the features in two ways, from a depth map and by triangulation using the previously estimated motion. Further, a bundle adjustment is implemented

which refines the frame to frame motion estimates. The method is tested with author-collected data using two sensors and the KITTI benchmark datasets.

Chapter 4

Lidar Odometry and Mapping

This chapter proposes a real-time method for low-drift odometry and mapping using range measurements from a 3D laser scanner moving in 6-DOF. The problem is hard because the range measurements are received at different times, and errors in motion estimation (especially without an external reference such as GPS) cause mis-registration of the resulting point cloud. To date, coherent 3D maps have been built by off-line batch methods, often using loop closure to correct for drift over time. The proposed method achieves both low-drift in motion estimation and low-computational complexity. The key idea that makes this level of performance possible is the division of the complex problem of Simultaneous Localization and Mapping, which seeks to optimize a large number of variables simultaneously, into two algorithms. One algorithm performs odometry at a high-frequency but at low fidelity to estimate velocity of the laser scanner. The other algorithm runs at an order of magnitude lower frequency for fine matching and registration of the point cloud. Combination of the two algorithms allows map creation in real-time. The method has been evaluated by indoor and outdoor experiments as well as well-known publicly available datasets. The method produces an average position error as 0.78% of the distance traveled on the KITTI odometry benchmark.

4.1 Introduction

3D Mapping remains a popular technology. The main issue with laser ranging in which the laser moves has to do with registration of the resulting point cloud. If the only motion is the pointing of a laser beam with known internal kinematics of the lidar from a fixed base, this registration is obtained simply. However, if the sensor base moves, as in many applications of interest, laser point registration has to do with both the internal kinematics and external motion. The second one has to contain knowledge of how the sensor is located and oriented for every range measurement. Since lasers can measure distance up to several hundred thousand times per second, high-rate pose estimation is a significant issue. A common way to solve this problem is to use an independent method of pose estimation (such as with an accurate GPS/INS system) to register the range data into a coherent point cloud in reference to a fixed coordinate frame. When independent measurements relative to a fixed coordinate frame are unavailable, the general technique used is to register points using some sort of odometry estimation, e.g. using combinations of

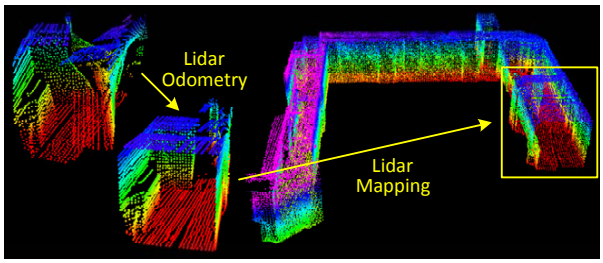


Figure 4.1: The method aims at motion estimation and mapping using a moving 3D lidar. Since the laser points are received at different times, distortion is present in the point cloud due to motion of the lidar (shown in the left lidar cloud). The proposed method decomposes the problem by two algorithms running in parallel. An odometry algorithm estimates velocity of the lidar and corrects distortion in the point cloud, then, a mapping algorithm matches and registers the point cloud to create a map. Combination of the two algorithms ensures real-time performance.

wheel motion, gyros, and by tracking features in range or visual images.

Here, let us consider the case of creating maps using low-drift odometry with a mechanically scanned laser ranging device (optionally augmented with low-grade inertial measurements) moving in 6-DOF. A key advantage of only using laser ranging is that it is not sensitive to ambient lighting or optical texture in the scene. New developments in laser scanners have reduced the size and weight of such devices to the level that they can be attached to mobile robots (including flying, walking or rolling) and even to people who move around in an environment to be mapped. Since the work seeks to push the odometry toward the lowest possible drift in real-time, I don't consider issues related to loop closure. Indeed, while loop closure could help further cancel the drift, one finds that in many practical cases, loop closure is unnecessary.

The method, namely LOAM, achieves both low-drift in motion estimation in 6-DOF and low-computational complexity. The key idea that makes this level of performance possible is the division of the typically complex problem of simultaneous localization and mapping, which seeks to optimize a large number of variables simultaneously, into two algorithms. One algorithm performs odometry at a high-frequency but at low fidelity to estimate velocity of the laser scanner moving through the environment. Although not necessary, if an IMU is available, it can provide a motion prior and help account for gross, high-frequency motion. A second algorithm runs at an order of magnitude lower frequency for fine matching and registration of the point cloud. Specifically, both algorithms extract feature points located on edges and planar surfaces and match the feature points to edge-line segments and planar surface patches, respectively. In the odometry algorithm, correspondences of the feature points are found by ensuring fast computation, while in the mapping algorithm, by ensuring accuracy.

In the method, an easier problem is solved first as online velocity estimation, after which, mapping is conducted as batch optimization to produce high-precision motion estimation and maps. The parallel algorithm structure ensures feasibility of the problem to be solved in real-time. Further, since motion estimation is conducted at a higher frequency, mapping is given plenty of time to enforce accuracy. When staggered to run at an order of magnitude slower than the odometry algorithm, the mapping algorithm incorporates a large number of feature points and uses sufficiently many iterations to converge.

4.2 Notations and Task Description

The problem addressed in this chapter is to perform ego-motion estimation with point clouds perceived by a 3D lidar, and build a map for the traversed environment. Assume that the lidar is intrinsically calibrated with the lidar internal kinematics precisely known (the intrinsic calibration makes 3D projection of the laser points possible). Also assume that the angular and linear velocities of the lidar are smooth and continuous over time, without abrupt changes. The second assumption will be released by usage of an IMU, in Section 4.6.2 and 4.6.3.

As a convention in this chapter, Iet us use right uppercase superscription to indicate the coordinate systems. Define a sweep as the lidar completes one time of scan coverage. Let us use right subscription k, $k \in Z^+$ to indicate the sweeps, and $\mathcal{P}_k$ to indicate the point cloud perceived during sweep k. Let us define two coordinate systems as follows.

- Lidar coordinate system $\{L\}$ is a 3D coordinate system with its origin at the geometric center of the lidar (see Fig. 4.2). Here, I use the convention of cameras. The x-axis is pointing to the left, the y-axis is pointing upward, and the z-axis is pointing forward. Denote a point i received during sweep k as $\boldsymbol{X}^L_{(k,i)}$. Further, let us use $\boldsymbol{T}^L_k(t)$ to denote the transform projecting a point received at time t to the beginning of the sweep k.
- World coordinate system $\{W\}$ is a 3D coordinate system coinciding with $\{L\}$ at the initial pose. Denote a point i in $\{W\}$ as $\boldsymbol{X}^W_{(k,i)}$ and denote $\boldsymbol{T}^W_k(t)$ as the transform projecting a point received at time t to $\{W\}$.

With assumptions and notations made, the lidar odometry and mapping problem is,

Problem: Given a sequence of lidar cloud $\mathcal{P}_k$, $k \in Z^+$, compute ego-motion of the lidar in the world, $\boldsymbol{T}^W_k(t)$, and build a map with $\mathcal{P}_k$ for the traversed environment.

4.3 System Overview

4.3.1 Lidar Hardware

The study of this chapter is validated on a sensor systems: a back-and-forth spin lidar, a continuously-spinning lidar, a Velodyne HDL-32 lidar, and the sensor system used by the KITTI benchmark [33, 36]. Let us use the first lidar hardware as an example to illustrate the method, therefore I introduce the lidar hardware in the front of the chapter to help readers understand the method. The rest sensors will be introduced in the experiment section. As shown in Fig. 4.2, the lidar is based on a Hokuyo UTM-30LX laser scanner which has $180°$ field of view with $0.25°$ resolution and 40 lines/sec scanning rate. The laser scanner is connected to a motor controlled to rotate at $180°/s$ angular speed between $-90°$ and $90°$ with the horizontal orientation of the laser scanner as zero. With this particular unit, a sweep is a rotation from $-90°$ to $90°$ or in the inverse direction (lasting for 1s). Here, note that for a continuously-spinning lidar, a sweep is simply a semi-spherical or a full-spherical rotation. An onboard encoder measures the motor rotation angle with $0.25°$ resolution, with which, the laser points are back-projected into $\{L\}$.

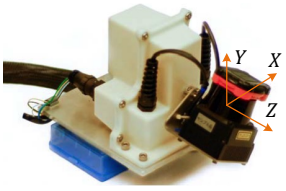


Figure 4.2: An example 3D lidar using in experiment evaluation. I will use data from this sensor to illustrate the method. The sensor consists of a Hokuyo laser scanner driven by a motor for rotational motion, and an encoder that measures the rotation angle. The laser scanner has 180° field of view and 0.25° resolution. The scanning rate is 40 lines/sec. The motor is controlled to rotate from -90° to 90° with the horizontal orientation of the laser scanner as zero.

4.3.2 Software System Overview

Fig. 4.3 shows a diagram of the software system. Let $\hat{\mathcal{P}}$ be the points received in a laser scan. During each sweep, $\hat{\mathcal{P}}$ is registered in $\{L\}$. The combined point cloud during sweep k forms $\mathcal{P}_k$. Then, $\mathcal{P}_k$ is processed in two algorithms. Lidar odometry takes the point cloud and computes the motion of the lidar between two consecutive sweeps. The estimated motion is used to correct distortion in $\mathcal{P}_k$. The algorithm runs at a frequency around 10Hz. The outputs are further processed by lidar mapping, which matches and registers the undistorted cloud onto a map at a frequency of 1Hz. Finally, the pose transforms published by the two algorithms are integrated to generate a transform output around 10Hz, regarding the lidar pose with respect to the map. Section 4.4 and 4.5 present the blocks in the software diagram in detail.

4.4 Lidar Odometry

4.4.1 Feature Point Extraction

I will start with extraction of feature points from the lidar cloud, $\mathcal{P}_k$. Notice that many 3D lidars naturally generate unevenly distributed points in $\mathcal{P}_k$. With the lidar in Fig. 4.2 as an example, the returns from the laser scanner has a resolution of 0.25° within a scan. These points are located on a scan plane. However, as the laser scanner rotates at an angular speed of $180^\circ/s$ and generates scans at 40Hz, the resolution in the perpendicular direction to the scan planes is $180^\circ/40 = 4.5^\circ$. Considering this fact, the feature points are extracted from $\mathcal{P}_k$ using only information from individual scans, with co-planar geometric relationship.

I select feature points that are on sharp edges and planar surface patches. Let i be a point in $\mathcal{P}_k$, $i \in \mathcal{P}_k$, and let $\mathcal{S}$ be the set of consecutive points of i returned by the laser scanner in

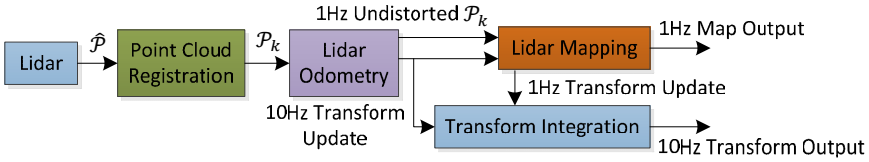


Figure 4.3: Block diagram of the lidar odometry and mapping software system.

the same scan. Since the laser scanner generates point returns in CW or CCW order, $\mathcal{S}$ contains half of its points on each side of i and $0.25°$ intervals between two points (still with the lidar in Fig. 4.2 as an example). Define a term to evaluate the smoothness of the local surface,

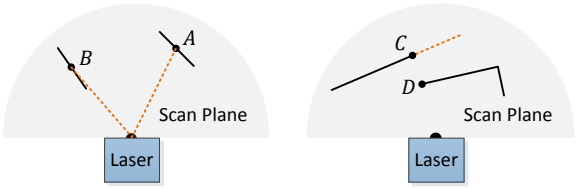
$$c = \frac{1}{|\mathcal{S}| \cdot ||\boldsymbol{X}^L_{(k,i)}||} || \sum_{j \in \mathcal{S}, j \neq i} (\boldsymbol{X}^L_{(k,i)} - \boldsymbol{X}^L_{(k,j)}) ||. \tag{4.1}$$

The term is normalized w.r.t. the distance to the lidar center. This is particularly made to remove scale effect and the term can be used for both near and far points.

The points in a scan are sorted based on the c values, then feature points are selected with the maximum c's, namely, edge points, and the minimum c's, namely planar points. To evenly distribute the feature points within the environment, I separate a scan into four identical subregions. Each subregion can provide maximally 2 edge points and 4 planar points. A point i can be selected as an edge or a planar point only if its c value is larger or smaller than a threshold (5×10^{-3}), and the number of points does not exceed the maximum point number of a subregion.

While selecting feature points, I would like to avoid points whose surrounded points are selected, or points on local planar surfaces that are roughly parallel to the laser beams (point B in Fig. 4.4(a)). These points are usually considered as unreliable. Also, I would like to avoid points that are on boundary of occluded regions [59]. An example is shown in Fig. 4.4(b). Point C is an edge point in the lidar cloud because its connected surface (the dotted line segment) is blocked by another object. However, if the lidar moves to another point of view, the occluded region can change and become observable. To avoid the aforementioned points to be selected, I find again the set of points $\mathcal{S}$. A point i can be selected only if $\mathcal{S}$ does not form a surface patch whose normal is within $10°$ to the laser beam, and there is no point in $\mathcal{S}$ that is disconnected from i by a gap in the direction of the laser beam and is at the same time closer to the lidar then point i (e.g. point B in Fig. 4.4(b)).

In summary, the feature points are selected as edge points starting from the maximum c value, and planar points starting from the minimum c value, and if a point is selected,



(a) (b)

Figure 4.4: (a) The solid line segments represent local surface patches. Point A is on a surface patch that has an angle to the laser beam (the dotted orange line segments). Point B is on a surface patch that is roughly parallel to the laser beam. I treat B as a unreliable laser return and do not select it as a feature point. (b) The solid line segments are observable objects to the laser. Point C is on the boundary of an occluded region (the dotted orange line segment), and can be detected as an edge point. However, if viewed from a different angle, the occluded region can change and become observable. I do not select D as a feature point.

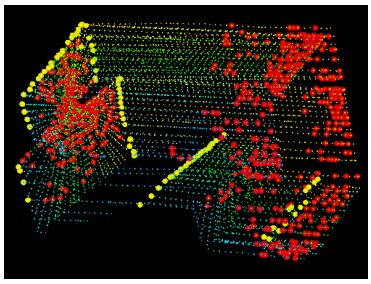


Figure 4.5: An example of extracted edge points (yellow) and planar points (red) from lidar cloud taken in a corridor. Meanwhile, the lidar moves toward the wall on the left side of the figure at a speed of 0.5m/s, this results in motion distortion on the wall.

- The number of selected edge points or planar points cannot exceed the maximum of the subregion, and
- None of its surrounding point is already selected, and
- It cannot be on a surface patch whose normal is within 10° to the laser beam, or on boundary of an occluded region.

An example of extracted feature points from a corridor scene is shown in Fig. 4.5. The edge points and planar points are labeled in yellow and red colors, respectively.

4.4.2 Finding Feature Point Correspondence

The odometry algorithm estimates motion of the lidar within a sweep. Let t_k be the starting time of a sweep k. At the end of sweep $k-1$, the point cloud perceived during the sweep, $\mathcal{P}_{k-1}$, is projected to time stamp t_k, illustrated in Fig. 4.6 (will discuss transforms projecting the points in Section 4.4.3). Let us denote the projected point cloud as $\bar{\mathcal{P}}_{k-1}$. During the next sweep k, $\bar{\mathcal{P}}_{k-1}$ is used together with the newly received point cloud, $\mathcal{P}_k$, to estimate the motion of the lidar.

Let us assume that both $\bar{\mathcal{P}}_{k-1}$ and $\mathcal{P}_k$ are available for now, and start with finding correspondences between the two lidar clouds. With $\mathcal{P}_k$, I find edge points and planar points from the lidar cloud using the methodology discussed in the last section. Let $\mathcal{E}_k$ and $\mathcal{H}_k$ be the sets of edge points and planar points, respectively. I will find edge lines from $\bar{\mathcal{P}}_{k-1}$ as correspondences of the points in $\mathcal{E}_k$, and planar patches as correspondences of those in $\mathcal{H}_k$.

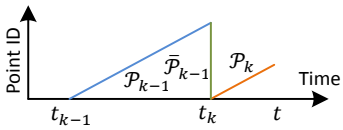


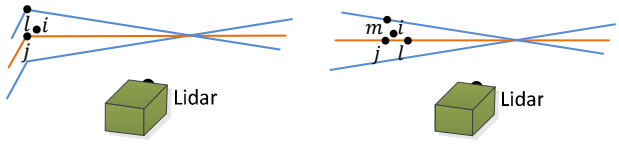
Figure 4.6: Project point cloud to the end of a sweep. The blue colored line segment represents the point cloud perceived during sweep k, $\mathcal{P}_{k-1}$. At the end of sweep $k-1$, $\mathcal{P}_{k-1}$ is projected to time stamp t_k to obtain $\bar{\mathcal{P}}_{k-1}$ (the green colored line segment). Then, during sweep k, $\bar{\mathcal{P}}_{k-1}$ and the newly perceived point cloud $\mathcal{P}_k$ (the orange colored line segment) are used together to estimate the lidar motion.

Note that at the beginning of sweep k, $\mathcal{P}_k$ is an empty set, which grows during the course of the sweep as more points are received. Lidar odometry recursively estimates the 6-DOF motion during the sweep, and gradually includes more points as $\mathcal{P}_k$ increases. $\mathcal{E}_k$ and $\mathcal{H}_k$ are projected to the beginning of the sweep (again, will discuss transforms projecting the points later). Let $\tilde{\mathcal{E}}_k$ and $\tilde{\mathcal{H}}_k$ be the projected point sets. For each point in $\tilde{\mathcal{E}}_k$ and $\tilde{\mathcal{H}}_k$, I am going to find the closest neighbor point in $\bar{\mathcal{P}}_{k-1}$. Here, $\bar{\mathcal{P}}_{k-1}$ is stored in a 3D KD-tree [16] in $\{L_k\}$ for fast index.

Fig. 4.7(a) represents the procedure of finding an edge line as the correspondence of an edge point. Let i be a point in $\tilde{\mathcal{E}}_k$, $i \in \tilde{\mathcal{E}}_k$. The edge line is represented by two points. Let j be the closest neighbor of i in $\bar{\mathcal{P}}_{k-1}$, $j \in \bar{\mathcal{P}}_{k-1}$, and let l be the closest neighbor of i in the preceding and following two scans to the scan of j. $(j,\ l)$ forms the correspondence of i. Then, to verify both j and l are edge points, I check the smoothness of the local surface based on (4.1) and require that both points have $c > 5 \times 10^{-3}$. Here, I particularly require that j and l are from different scans considering that a single scan cannot contain more than one points from the same edge line. There is only one exception where the edge line is on the scan plane. If so, however, the edge line is degenerated and appears as a straight line on the scan plane, and feature points on the edge line should not be extracted in the first place.

Fig. 4.7(b) shows the procedure of finding a planar patch as the correspondence of a planar point. Let i be a point in $\tilde{\mathcal{H}}_k$, $i \in \tilde{\mathcal{H}}_k$. The planar patch is represented by three points. Similar to the last paragraph, I find the closest neighbor of i in $\bar{\mathcal{P}}_{k-1}$, denoted as j. Then, I find another two points, l and m, as the closest neighbors of i, one in the same scan of j but not j, and the other in the preceding and following scans to the scan of j. This guarantees that the three points are non-collinear. To verify that j, l, and m are all planar points, again, I check the smoothness of the local surface and require $c < 5 \times 10^{-3}$.

With the correspondences of the feature points found, now I will derive expressions to compute the distance from a feature point to its correspondence. I will recover the lidar motion by minimizing the overall distances of the feature points in the next section. I will start with edge points. For a point $i \in \tilde{\mathcal{E}}_k$, if $(j,\ l)$ is the corresponding edge line, $j, l \in \bar{\mathcal{P}}_{k-1}$, the point to line



(a) (b)

Figure 4.7: Finding an edge line as the correspondence for an edge point in $\tilde{\mathcal{E}}_k$ (a), and a planar patch as the correspondence for a planar point in $\tilde{\mathcal{H}}_k$ (b). In both (a) and (b), j is the closest point to the feature point i, found in $\bar{\mathcal{P}}_{k-1}$. The orange lines represent the same scan of j, and the blue lines are the preceding and following scans. To find the edge line correspondence in (a), I find another point, l, on the blue lines, and the correspondence is represented as $(j,\ l)$. To find the planar patch correspondence in (b), I find another two points, l and m, on the orange line and the blue line, respectively. The correspondence is $(j,\ l,\ m)$.

distance can be computed as

$$d_{\mathcal{E}} = \frac{\left|(\tilde{\boldsymbol{X}}^L_{(k,i)} - \bar{\boldsymbol{X}}^L_{(k-1,j)}) \times (\tilde{\boldsymbol{X}}^L_{(k,i)} - \bar{\boldsymbol{X}}^L_{(k-1,l)})\right|}{\left|\bar{\boldsymbol{X}}^L_{(k-1,j)} - \bar{\boldsymbol{X}}^L_{(k-1,l)}\right|}, \tag{4.2}$$

where $\tilde{\boldsymbol{X}}^L_{(k,i)}$, $\bar{\boldsymbol{X}}^L_{(k-1,j)}$, and $\bar{\boldsymbol{X}}^L_{(k-1,l)}$ are the coordinates of points i, j, and l in $\{L_k\}$, respectively. Then, for a point $i \in \tilde{\mathcal{H}}_k$, if $(j,\ l,\ m)$ is the corresponding planar patch, $j, l, m \in \bar{\mathcal{P}}_{k-1}$, the point to plane distance is

$$d_{\mathcal{H}} = \frac{\left|\begin{array}{c}(\tilde{\boldsymbol{X}}^L_{(k,i)} - \bar{\boldsymbol{X}}^L_{(k-1,j)}) \\ ((\bar{\boldsymbol{X}}^L_{(k-1,j)} - \bar{\boldsymbol{X}}^L_{(k-1,l)}) \times (\bar{\boldsymbol{X}}^L_{(k-1,j)} - \bar{\boldsymbol{X}}^L_{(k-1,m)}))\end{array}\right|}{\left|(\bar{\boldsymbol{X}}^L_{(k-1,j)} - \bar{\boldsymbol{X}}^L_{(k-1,l)}) \times (\bar{\boldsymbol{X}}^L_{(k-1,j)} - \bar{\boldsymbol{X}}^L_{(k-1,m)})\right|}. \tag{4.3}$$

4.4.3 Motion Estimation

The lidar motion is modeled with constant angular and linear velocities during a sweep. This allows us to linear interpolate the pose transform within a sweep for the points that are received at different times. Let t be the current time stamp, and recall that t_k is the starting time of the current sweep k. Let $\boldsymbol{T}^L_k(t)$ be the lidar pose transform between $[t_k,\ t]$. $\boldsymbol{T}^L_k(t)$ contains 6-DOF motion of the lidar, $\boldsymbol{T}^L_k(t) = [\tau^L_k(t),\ \theta^L_k(t)]^T$, where $\tau^L_k(t) = [t_x,\ t_y,\ t_z]^T$ is the translation and $\theta^L_k(t) = [\theta_x,\ \theta_y,\ \theta_z]^T$ is the rotation in $\{L_k\}$. Given $\theta^L_k(t)$, the corresponding rotation matrix can be defined by the Rodrigues formula [66],

$$\begin{aligned}\mathbf{R}^L_k(t) = e^{\hat{\theta}^L_k(t)} = \mathbf{I} + \frac{\hat{\theta}^L_k(t)}{||\theta^L_k(t)||}\sin||\theta^L_k(t)|| \\ + (\frac{\hat{\theta}^L_k(t)}{||\theta^L_k(t)||})^2(1 - ||\cos\theta^L_k(t)||).\end{aligned} \tag{4.4}$$

where $\hat{\theta}^L_k(t)$ is the skew symmetric matrix of $\theta^L_k(t)$.

Given a point i, $i \in \mathcal{P}_k$, let $t_{(k,i)}$ be its time stamp, and let $\boldsymbol{T}^L_{(k,i)}$ be the pose transform between $[t_k,\ t_{(k,i)}]$. $\boldsymbol{T}^L_{(k,i)}$ can be computed by linear interpolation of $\boldsymbol{T}^L_k(t)$,

$$\boldsymbol{T}^L_{(k,i)} = \frac{t_{(k,i)} - t_k}{t - t_k}\boldsymbol{T}^L_k(t). \tag{4.5}$$

Here, note that $\boldsymbol{T}^L_k(t)$ is a changing variable over time and the interpolation uses the transform of current time t. Recall that $\mathcal{E}_k$ and $\mathcal{H}_k$ are the sets of edge points and planar points extracted from $\mathcal{P}_k$. The next equation projects $\mathcal{E}_k$ and $\mathcal{H}_k$ to the beginning of the sweep, namely $\tilde{\mathcal{E}}_k$ and $\tilde{\mathcal{H}}_k$,

$$\tilde{\boldsymbol{X}}^L_{(k,i)} = \mathbf{R}^L_{(k,i)}\boldsymbol{X}^L_{(k,i)} + \tau^L_{(k,i)}, \tag{4.6}$$

where $\boldsymbol{X}^L_{(k,i)}$ is a point in $\mathcal{E}_k$ or $\mathcal{H}_k$ and $\tilde{\boldsymbol{X}}^L_{(k,i)}$ is the corresponding point in $\tilde{\mathcal{E}}_k$ or $\tilde{\mathcal{H}}_k$. $\mathbf{R}^L_{(k,i)}$ and $\tau^L_{(k,i)}$ are the rotation matrix and translation vector corresponding to $\boldsymbol{T}^L_{(k,i)}$.

Recall that (4.2) and (4.3) compute the distances between points in $\tilde{\mathcal{E}}_k$ and $\tilde{\mathcal{H}}_k$ and their correspondences. Combining (4.2) and (4.6), one can derive a geometric relationship between an edge point in $\mathcal{E}_k$ and the corresponding edge line,

$$f_{\mathcal{E}}(\boldsymbol{X}^L_{(k,i)}, \boldsymbol{T}^L_k(t)) = d_{\mathcal{E}},\ i \in \mathcal{E}_k. \tag{4.7}$$

Similarly, combining (4.3) and (4.6), one can establish another geometric relationship between a planar point in $\mathcal{H}_k$ and the corresponding planar patch,

$$f_{\mathcal{H}}(\boldsymbol{X}^L_{(k,i)}, \boldsymbol{T}^L_k(t)) = d_{\mathcal{H}},\ i \in \mathcal{H}_k. \tag{4.8}$$

Finally, I solve the lidar motion with the Levenberg-Marquardt method [38]. Stacking (4.7) and (4.8) for each feature point in $\mathcal{E}_k$ and $\mathcal{H}_k$, I obtain a nonlinear function,

$$\boldsymbol{f}(\boldsymbol{T}^L_k(t)) = \boldsymbol{d}, \tag{4.9}$$

where each row of $\boldsymbol{f}$ corresponds to a feature point, and $\boldsymbol{d}$ contains the corresponding distances. Compute the Jacobian matrix of $\boldsymbol{f}$ with respect to $\boldsymbol{T}^L_k(t)$, denoted as $\mathbf{J}$, where $\mathbf{J} = \partial f / \partial \boldsymbol{T}^L_k(t)$. Then, (4.9) can be solved through nonlinear iterations by minimizing $\boldsymbol{d}$ toward zero,

$$\boldsymbol{T}^L_k(t) \leftarrow \boldsymbol{T}^L_k(t) - (\mathbf{J}^T\mathbf{J} + \lambda \mathrm{diag}(\mathbf{J}^{\mathrm{T}}\mathbf{J}))^{-1}\mathbf{J}^{\mathrm{T}}\boldsymbol{d}. \tag{4.10}$$

λ is a factor determined by the Levenberg-Marquardt method.

4.4.4 Lidar Odometry Algorithm

Lidar odometry algorithm is shown in Algorithm 2. The algorithm takes as inputs the point cloud from the last sweep, $\bar{\mathcal{P}}_{k-1}$, the growing point cloud of the current sweep, $\mathcal{P}_k$, and the pose transform from the last recursion as initial guess, $\boldsymbol{T}^L_k(t)$. If a new sweep is started, $\boldsymbol{T}^L_k(t)$ is set to zero to re-initialize (line 4-6). Then, the algorithm extracts feature points from $\mathcal{P}_k$ to construct $\mathcal{E}_k$ and $\mathcal{H}_k$ on line 7. For each feature point, I find its correspondence in $\bar{\mathcal{P}}_{k-1}$ (line 9-19). The motion estimation is adapted to a robust fitting framework [1]. On line 15, the algorithm assigns a bisquare weight for each feature point as the following equation. The feature points that have larger distances to their correspondences are assigned with smaller weights, and the feature points with distances larger than a threshold are considered as outliers.

$$w = \begin{cases} (1-\alpha^2)^2 & -1 < \alpha < 1, \\ 0 & \text{otherwise}, \end{cases} \tag{4.11}$$

where

$$\alpha = \frac{r}{6.9459\sigma\sqrt{1-h}}.$$

In the above equation, r is the corresponding residual in the least square problem, σ is the absolute deviation of the residuals from the median, and h is the leverage value or the corresponding element on the diagonal of matrix $\mathbf{J}(\mathbf{J}^T\mathbf{J}))^{-1}\mathbf{J}^T$ where $\mathbf{J}$ is the same Jacobian matrix used in (4.10). Then, on line 16, the pose transform is updated for one iteration. The nonlinear optimization terminates if convergence is found, or the maximum iteration number is met. If the algorithm reaches the end of a sweep, $\mathcal{P}_k$ is projected to time stamp t_{k+1} using the estimated motion during the sweep, forming $\bar{\mathcal{P}}_k$. This makes ready for the next sweep to be matched to $\bar{\mathcal{P}}_k$. Otherwise, only the transform $\boldsymbol{T}^L_k(t)$ is returned by the algorithm for the next round of recursion.

Algorithm 2: Lidar Odometry

input : $\bar{\mathcal{P}}_{k-1}$, $\mathcal{P}_k$, $\boldsymbol{T}_k^L(t)$ from the last recursion at initial guess
output : $\bar{\mathcal{P}}_k$, newly computed $\boldsymbol{T}_k^L(t)$
begin
  if at the beginning of a sweep **then**
    $\boldsymbol{T}_k^L(t) \leftarrow \mathbf{0}$;
  end
  Detect edge points and planar points in $\mathcal{P}_k$, put the points in $\mathcal{E}_k$ and $\mathcal{H}_k$, respectively;
  for a number of iterations **do**
    for each edge point in $\mathcal{E}_k$ **do**
      Find an edge line as the correspondence, then compute point to line distance based on (4.7) and stack the equation to (4.9);
    end
    for each planar point in $\mathcal{H}_k$ **do**
      Find a planar patch as the correspondence, then compute point to plane distance based on (4.8) and stack the equation to (4.9);
    end
    Compute a bisquare weight for each row of (4.9);
    Update $\boldsymbol{T}_k^L(t)$ for a nonlinear iteration based on (4.10);
    if the nonlinear optimization converges **then**
      Break;
    end
  end
  if at the end of a sweep **then**
    Project each point in $\mathcal{P}_k$ to t_{k+1} and form $\bar{\mathcal{P}}_k$;
    Return $\boldsymbol{T}_k^L(t)$ and $\bar{\mathcal{P}}_k$;
  end
  else
    Return $\boldsymbol{T}_k^L(t)$;
  end
end

4.5 Lidar Mapping

The mapping algorithm runs at a lower frequency then the odometry algorithm, and is called only once per sweep. At the end of sweep k, lidar odometry generates a undistorted point cloud, $\bar{\mathcal{P}}_k$, and simultaneously a pose transform, $\boldsymbol{T}_k^L(t_{k+1})$, containing the lidar motion during the sweep, between $[t_k,\ t_{k+1}]$. The mapping algorithm matches and registers $\bar{\mathcal{P}}_k$ in the world coordinates, $\{W\}$, illustrated in Fig. 4.8. To explain the procedure, let us define $\mathcal{Q}_{k-1}$ as the point cloud on the map, accumulated until sweep $k-1$, and let $\boldsymbol{T}_{k-1}^W(t_k)$ be the pose of the lidar on the map at the end of sweep $k-1$, t_k. With the output from lidar odometry, the mapping algorithm extents $\boldsymbol{T}_{k-1}^W(t_k)$ for one sweep from t_k to t_{k+1}, to obtain $\boldsymbol{T}_k^W(t_{k+1})$, and transforms $\bar{\mathcal{P}}_k$ into the world coordinates, $\{W\}$, denoted as $\bar{\mathcal{Q}}_k$. Next, the algorithm matches $\bar{\mathcal{Q}}_k$ with $\mathcal{Q}_{k-1}$ by optimizing the lidar pose $\boldsymbol{T}_k^W(t_{k+1})$.

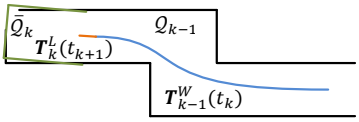


Figure 4.8: Illustration of mapping process. The blue curve represents the lidar pose on the map, $\boldsymbol{T}_{k-1}^W(t_k)$, generated by the mapping algorithm at sweep $k-1$. The orange curve indicates the lidar motion during the entire sweep k, $\boldsymbol{T}_k^L(t_{k+1})$, computed by the odometry algorithm. With $\boldsymbol{T}_{k-1}^W(t_k)$ and $\boldsymbol{T}_k^L(t_{k+1})$, the undistorted point cloud published by the odometry algorithm is projected onto the map, denoted as $\bar{\mathcal{Q}}_k$ (the green line segments), and matched with the existing cloud on the map, $\mathcal{Q}_{k-1}$ (the black colored line segments).

The feature points are extracted in the same way as in Section 4.4.1, but 10 times of feature points are used. To find correspondences for the feature points, I store the point cloud on the map, $\mathcal{Q}_{k-1}$, in 10m cubic areas. The points in the cubes that intersect with $\bar{\mathcal{Q}}_k$ are extracted and stored in a 3D KD-tree [16] in $\{W\}$. I find the points in $\mathcal{Q}_{k-1}$ within a certain region around the feature points. Let $\mathcal{S}'$ be a set of surrounding points. For an edge point, I only keep points on edge lines in $\mathcal{S}'$, and for a planar point, I only keep points on planar patches. The points are distinguished between edge points and planar points based on their c values. Here, I use the same threshold (5×10^{-3}). Then, I compute the covariance matrix of $\mathcal{S}'$, denoted as $\mathbf{M}$, and the eigenvalues and eigenvectors of $\mathbf{M}$, denoted as $\boldsymbol{V}$ and $\boldsymbol{E}$, respectively. These values determine poses of the point clusters and hence the point-to-line and point-to-plane distances. Specifically, if $\mathcal{S}'$ is distributed on an edge line, $\boldsymbol{V}$ contains one eigenvalue significantly larger than the other two, and the eigenvector in $\boldsymbol{E}$ associated with the largest eigenvalue represents the orientation of the edge line. On the other hand, if $\mathcal{S}'$ is distributed on a planar patch, $\boldsymbol{V}$ contains two large eigenvalues with the third one significantly smaller, and the eigenvector in $\boldsymbol{E}$ associated with the smallest eigenvalue denotes the orientation of the planar patch. The position of the edge line or the planar patch is calculated such that the line or the plane passes through the centroid of $\mathcal{S}'$.

To compute the distance from a feature point to its correspondence, I select two points on an edge line, and three points on a planar patch. This allows the distances to be computed using the same formulations as (4.2) and (4.3). Then, an equation is derived for each feature point as (4.7) or (4.8), but different in that all points in $\bar{\mathcal{Q}}_k$ share the same time stamp, t_{k+1}. The nonlinear optimization is solved again by the Levenberg-Marquardt method [38] adapted to robust fitting [1], and then $\bar{\mathcal{Q}}_k$ is registered on the map.

Integration of the pose transforms is illustrated in Fig. 4.9. The blue colored region represents the pose output from lidar mapping, $\boldsymbol{T}_{k-1}^W(t_k)$, generated once per sweep. The orange colored region represents the transform output from lidar odometry, $\boldsymbol{T}_k^L(t)$, at a frequency round 10Hz.

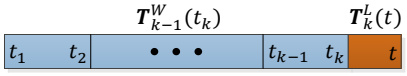


Figure 4.9: Integration of pose transforms. The blue colored region illustrates the lidar pose from the mapping algorithm, $\boldsymbol{T}_{k-1}^W(t_k)$, generated once per sweep. The orange colored region is the lidar motion within the current sweep, $\boldsymbol{T}_k^L(t)$, computed by the odometry algorithm. The motion estimation of the lidar is the combination of the two transforms, at the same frequency as $\boldsymbol{T}_k^L(t)$.

The lidar pose with respect to the map is the combination of the two transforms, at the same frequency as lidar odometry.

4.6 Experiments

During experiments, the algorithms processing the lidar data run on a laptop computer with 2.5GHz quad cores and 6Gib memory. The method consumes a total of two threads, the odometry and mapping programs run on two separate threads.

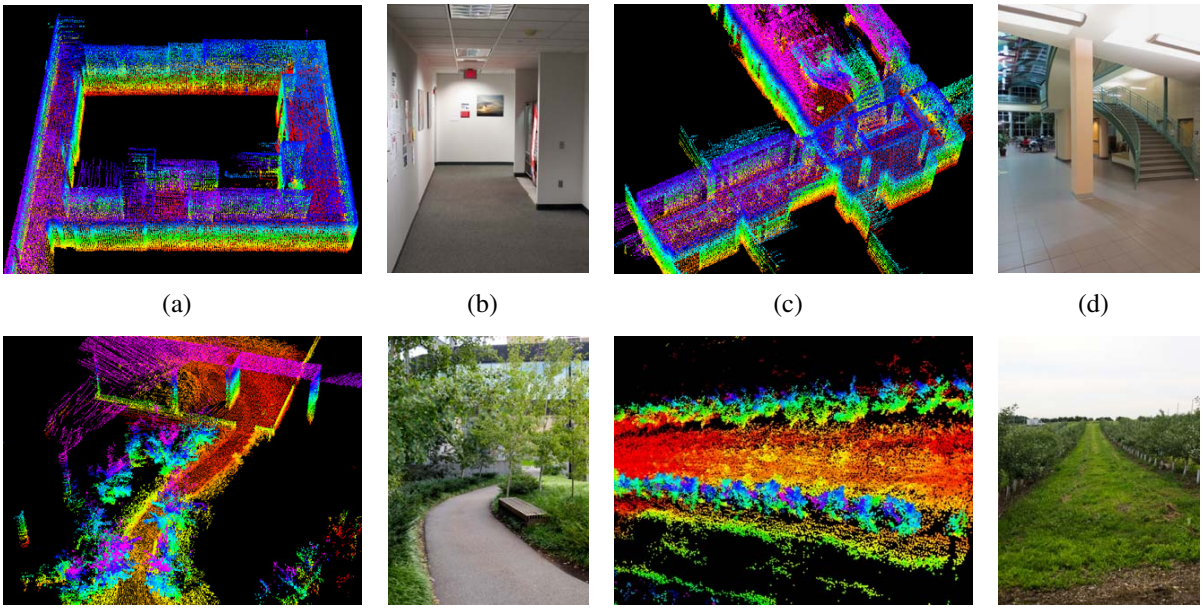
4.6.1 Accuracy Tests

The method has been tested in indoor and outdoor environments using the lidar in Fig. 4.2. During indoor tests, the lidar is placed on a cart together with a battery and a laptop computer. One person pushes the cart and walks. Fig. 4.10(a) and Fig. 4.10(c) show maps built in two representative indoor environments, a narrow and long corridor and a large lobby. Fig. 4.10(b) and Fig. 4.10(d) show two photos taken from the same scenes. In outdoor tests, the lidar is mounted to the front of a ground vehicle. Fig. 4.10(e) and Fig. 4.10(g) show maps generated from a vegetated road and an orchard between two rows of trees, and photos are presented in Fig. 4.10(f) and Fig. 4.10(h), respectively. During all tests, the lidar moves at a speed of 0.5m/s.

To evaluate local accuracy of the maps, I collect a second set of lidar clouds from the same environments. The lidar is kept stationary and placed at a few different places in each environment during data selection. The two point clouds are matched and compared using the point to plane ICP method [81]. After matching is complete, the distances between one point cloud and the corresponding planar patches in the second point cloud are considered as matching errors. Fig. 4.11 shows the density of error distributions. It indicates smaller matching errors in indoor environments than in outdoor. The result is reasonable because feature matching in natural environments is less exact than in manufactured environments.

Further, I would like to understand how lidar odometry and lidar mapping function and contribute to the final accuracy. To this end, I take the dataset in Fig. 4.1 and show output of each algorithm. The trajectory is 32m in length. Fig. 4.12(a) uses lidar odometry output to register laser points directly, while Fig. 4.12(b) is the final output optimized by lidar mapping. As mentioned at the beginning, the role of lidar odometry is to estimate velocity and remove motion distortion in point clouds. The low-fidelity of lidar odometry cannot warrant accurate mapping. On the other hand, lidar mapping further performs careful scan matching to warrant accuracy on the map. Table 4.1 shows computation time brake-down of the two programs in the accuracy tests. One sees lidar mapping takes totally 6.4 times of computation of lidar odometry to remove drift. Here, note that lidar odometry is called 10 times while lidar mapping is called once, resulting in same level of computation load on the two threads.

Additionally, I conduct tests to measure accumulated drift of the motion estimate. I choose corridor for indoor experiments that contains a closed loop. This allows us to start and finish at the same place. The motion estimation generates a gap between the starting and finishing positions, which indicates the amount of drift. For outdoor experiments, I choose orchard environment. The ground vehicle is equipped with a high accuracy GPS/INS for ground truth. The



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 4.10: Maps generated in (a)-(b) a narrow and long corridor, (c)-(d) a large lobby, (e)-(f) a vegetated road, and (g)-(h) an orchard between two rows of trees. The lidar is placed on a cart in indoor tests, and mounted on a ground vehicle in outdoor tests. All tests use a speed of 0.5m/s.

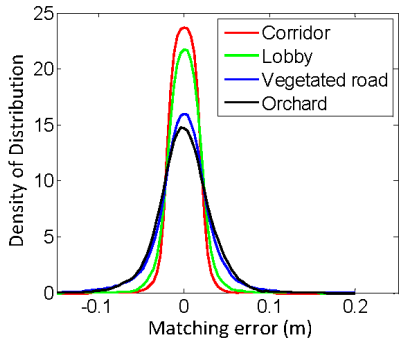
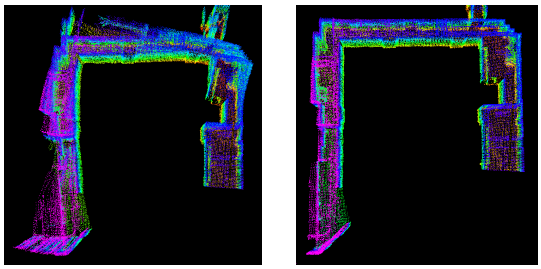


Figure 4.11: Matching errors for corridor (red), lobby (green), vegetated road (blue) and orchard (black), corresponding to the four scenes in Fig. 4.10.

Table 4.1: Computation brake-down for accuracy tests.

Program	Build KD-tree	Match features	Others	Total
Odometry	11ms	23ms	14ms	48ms
Mapping	58ms	134ms	117ms	309ms

measured drifts are compared to the distance traveled as the relative accuracy, and listed in Table 4.2. Specifically, Test 1 uses the same datasets with Fig. 4.10(a) and Fig. 4.10(g). In general, the indoor tests have a relative accuracy around 1% and the outdoor tests are around 2.5%.



(a) (b)

Figure 4.12: Comparison between (a) lidar odometry output and (b) final lidar mapping output with the dataset in Fig. 4.1. The role of lidar odometry is to estimate velocity and remove motion distortion in point clouds. This algorithm has a low fidelity. Lidar mapping further performs careful scan matching to warrant accuracy on the map.

Table 4.2: Relative errors for motion estimation drift.

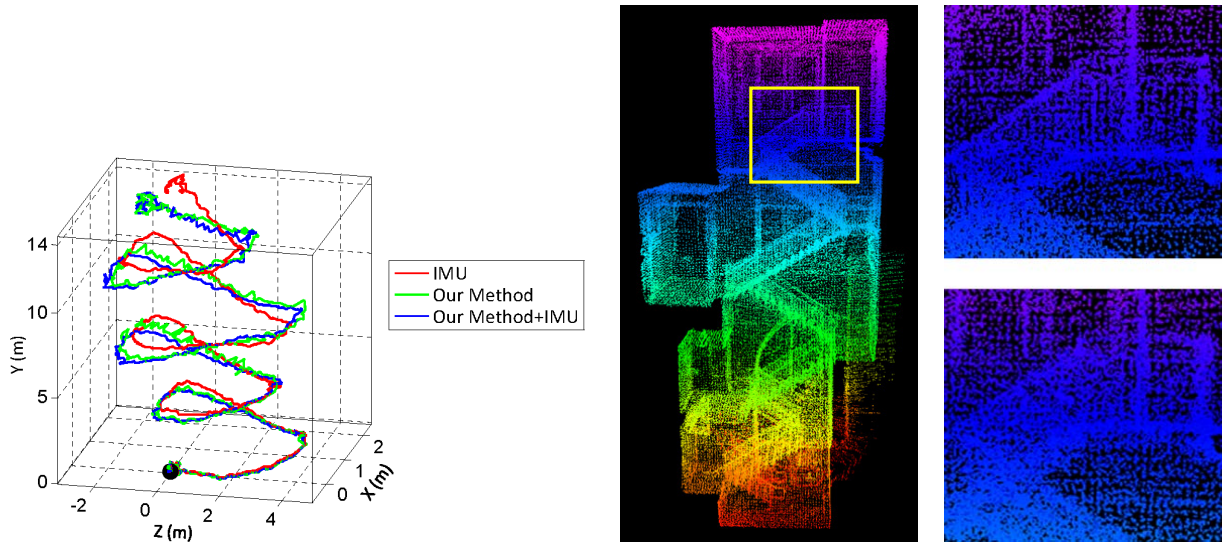
	Test 1		Test 2	
Environment	Distance	Error	Distance	Error
Corridor	58m	0.9%	46m	1.1%
Orchard	52m	2.3%	67m	2.8%

4.6.2 Tests with IMU Assistance

An Xsens MTi-10 IMU is attached to the lidar to deal with fast velocity changes. The point cloud is pre-processed in two ways before sending to the proposed method, 1) with orientation from the IMU, the point cloud received in one sweep is rotated to align with the initial orientation of the lidar in that sweep, 2) with acceleration measurement, the motion distortion is partially removed as if the lidar moves at a constant velocity during the sweep. After IMU pre-processing, the motion left to solve is the orientation drift from the IMU, assumed to be linear within a sweep, and the linear velocity. Hence, it satisfies the assumption that the lidar has linear motion within a sweep. The point cloud is then processed by the lidar odometry and mapping programs.

Fig. 4.13(a) shows a sample result. A person holds the lidar and walks on a staircase. When computing the red curve, I use orientation provided by the IMU, and the method only estimates translation. The orientation drifts over 25° during 5 mins of data collection. The green curve relies only on the optimization in the method, assuming no IMU is available. The blue curve uses the IMU data for preprocessing followed by the proposed method. One observes small difference between the green and blue curves. Fig. 4.13(b) presents the map corresponding to the blue curve. In Fig. 4.13(c), I compare two closed views of the maps in the yellow rectangular in Fig. 4.13(b). The upper and lower figures correspond to the blue and green curves, respectively. Careful comparison finds that the edges in the upper figure are sharper than the lower figure.

Table 4.3 compares relative errors in motion estimation with and without using the IMU. The lidar is held by a person walking at a speed of 0.5m/s and moving the lidar up and down at a magnitude around 0.5m. The ground truth is manually measured by a tape ruler. In all four tests, using the proposed method with assistance from the IMU gives the highest accuracy, while using orientation from the IMU only leads to the lowest accuracy. The results indicate that the IMU is



(a) (b) (c)

Figure 4.13: Comparison of results with/without aiding from an IMU. A person holds the lidar and walks on a staircase. The black dot is the starting point. In (a), the red curve is computed using orientation from the IMU and translation estimated by the method, the green curve relies on the optimization in the method only, and the blue curve uses the IMU data for preprocessing followed by the method. (b) is the map corresponding to the blue curve. In (c), the upper and lower figures correspond to the blue and green curves, respectively, using the region labeled by the yellow rectangle in (b). The edges in the upper figure are sharper, indicating more accuracy.

Table 4.3: Motion estimation errors with/without using IMU.

Environment	Distance	Error		
		IMU	Ours	Ours+IMU
Corridor	32m	16.7%	2.1%	0.9%
Lobby	27m	11.7%	1.7%	1.3%
Vegetated road	43m	13.7%	4.4%	2.6%
Orchard	51m	11.4%	3.7%	2.1%

effective in canceling the nonlinear motion, and the proposed method handles the linear motion.

4.6.3 Tests with Micro-helicopter datasets

Let us further evaluate the method with data collected from an octo-rotor micro aerial vehicle. As shown in Fig. 4.14, the helicopter is mounted with a 2-axis lidar, which shares the same design with the one in Fig. 4.2 except that the laser scanner spins continuously. For such a lidar unit, a sweep is defined as a semi-spherical rotation on the slow axis, lasting for one second. A Microstrain 3DM-GX3-45 IMU is also mounted on the helicopter.

Results from two datasets are shown in Fig. 4.15 and Fig. 4.16. For both tests, the helicopter is manually flown at a speed of 1m/s. In Fig. 4.15, the helicopter starts from one side of the bridge,

(a) (b)

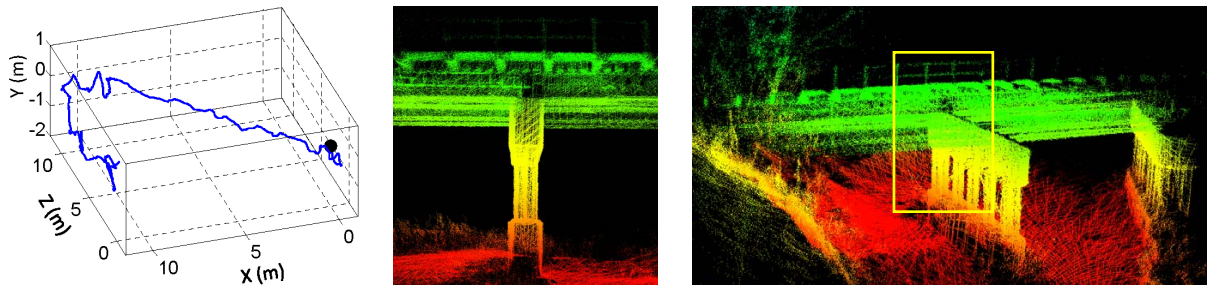
Figure 4.14: (a) Octo-rotor helicopter used in the study. A 2-axis lidar is mounted to the font of the helicopter with a zoomed in view in (b). The lidar is based on a Hokuyo laser scanner, sharing the same design with the one in Fig. 4.2 except the laser scanner spins continuously.

flies underneath the bridge and turns back to fly underneath the bridge for the second time. In Fig. 4.16, the helicopter starts with taking-off from the ground and ends with landing back on the ground. In Fig. 4.15-4.16(a), I show trajectories of the flights, and in Fig. 4.15-4.16(b), I show zoomed in views of the maps, corresponding to the areas inside the yellow rectangles in Fig. 4.15-4.16(c). I am not able to acquire ground truth for the helicopter poses or the maps. For relative small environments as in both tests, the method continuously re-localizes on the maps built at the beginning of the tests. Hence, calculating loop closure errors becomes meaningless. Instead, one can only visually exam accuracy of the maps in the zoomed in views.

4.6.4 Tests with a Velodyne lidar

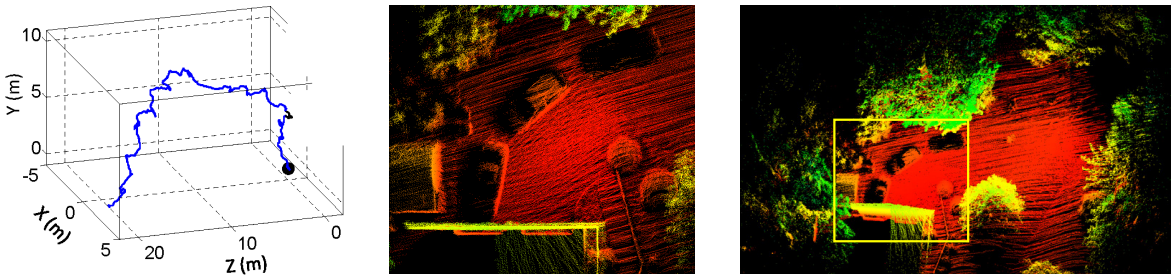
These experiments use a Velodyne HDL-32E lidar mounted on two vehicles shown in Fig. 4.17. Fig. 4.17(a) is a utility vehicle driven on sidewalks and off-road terrains. Fig. 4.17(b) is a passenger vehicle driven on streets. For both vehicles, the lidar is mounted high on the top to avoid possible occlusions by the vehicle body.

The Velodyne HDL-32E is a single-axis laser scanner. It projects 32 laser beams simultaneously into the 3D environment. I treat each plane formed by a laser beam as a scan plane. A



(a) (b) (c)

Figure 4.15: Results from a small bridge. (a) shows trajectory of the helicopter. The black dot is the starting position. (b) and (c) show the map built by the proposed method, where (b) is a zoomed in view of the area inside the yellow rectangle in (c). The helicopter is manually flown during data collection. Starting from one side, it flies underneath the bridge, turns back, and flies underneath the bridge again.



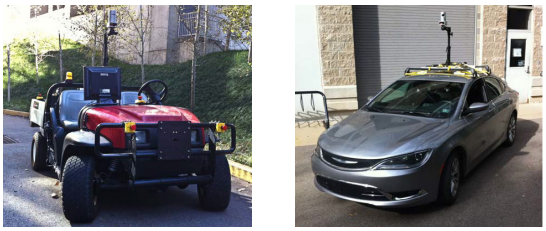
(a) (b) (c)

Figure 4.16: Results from the front of a house. The figures are in the same arrangement with Fig. 4.15. Again, (b) is a zoomed in view of the yellow rectangle in (c). The helicopter is manually flown, starting with taking-off from the ground and ending with landing on the ground.

sweep is defined as a full-circle rotation of the laser scanner. The lidar acquires scans at 10Hz by default. Lidar odometry is configured to run at 10Hz processing individual scans. Lidar mapping stacks scans for a second to do the batch optimization. The computation brake-down for the two programs is shown in Table 4.4.

Fig. 4.18 shows results of mapping the university campus. The data is logged with the vehicle in Fig. 4.17(a) for 1.0km of travel. The driving speed during the test is maintained at 2-3m/s. Fig. 4.18(a) shows the final map built. Fig. 4.18(b) shows the estimated trajectory (red curve) and the registered laser points overlaid on a satellite image. By matching the trajectory to the sidewalk (the vehicle is not driven on the street) and the laser points to the building walls, the horizontal position drift is determined to be $\leq$ 1m. By comparison of mapped buildings from both sides, I am able to determine the vertical drift to be $\leq$ 1.5m. This results in the overall position error to be $\leq 0.2\%$ of the distance traveled.

Fig. 4.19 shows results from another test. The vehicle was driven in Fig. 4.17(b) on streets

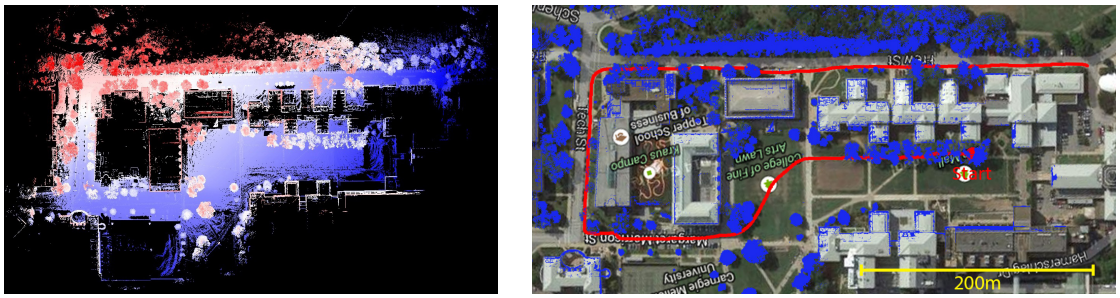


(a) (b)

Figure 4.17: Vehicles carrying a Velodyne HDL-32E lidar for data logging. (a) is a utility vehicle driven on sidewalks and off-road terrains. (b) is a passenger vehicle driven on streets.

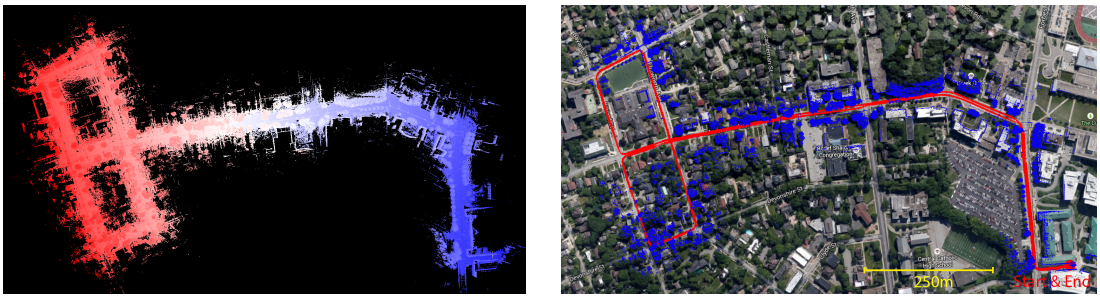
Table 4.4: Computation brake-down for Velodyne HDL-32E tests.

Program	Build KD-tree	Match features	Others	Total
Odometry	18ms	35ms	16ms	69ms
Mapping	131ms	283ms	149ms	563ms



(a) (b)

Figure 4.18: Results of mapping university campus. The overall run is 1.0km and the vehicle speed is at 2-3m/s. (a) shows the final map built and (b) shows the trajectory and registered laser points overlaied on a satellite image. The horizontal position error from matching with the satellite image is $\leq$ 1m.



(a) (b)

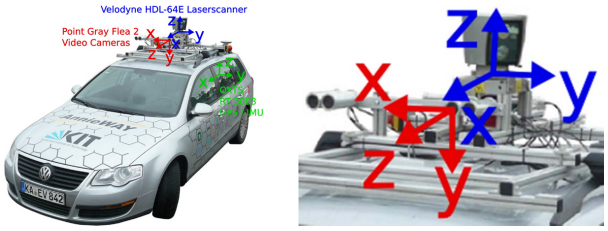
Figure 4.19: Results of mapping streets. The path is 3.6km in length and the vehicle speed is at 11-18m/s. The figures are in the same arrangement with Fig. 4.18. The horizontal position error from matching with the satellite image is $\leq$ 2m.

for 3.6km. Except waiting for traffic lights, the vehicle speed is mostly between 11-18m/s. The figures in Fig. 4.19 are organized in the same way as Fig. 4.18. By comparison with the satellite image, I determine the horizontal position error is $\leq$ 2m. For vertical accuracy, however, I am not able to evaluate.

4.6.5 Tests with KITTI Datasets

Finally, I test the method using the KITTI odometry benchmark [33, 36]. The datasets are logged with sensors mounted on top of a passenger vehicle in road driving scenarios. As shown in Fig. 4.20, the vehicle is equipped with color stereo cameras, monochrome stereo cameras, a Velodyne HDL-64E laser scanner, and a high accuracy GPS/INS for ground truth.

The datasets contain 11 tests with the GPS/INS ground truth provided. The maximum driving speed in the datasets reaches 85km/h (23.6m/s). The data covers mainly three types of environments: "urban" with buildings around, "country" on small roads with vegetations in the scene, and "highway" where roads are wide and the vehicle speed is fast. Fig. 4.21 presents sample results from the three environments. On the top row, the estimated trajectories of the vehicle



(a) (b)

Figure 4.20: (a) Vehicle used by the KITTI benchmark for data logging. The vehicle is mounted with a Velodyne lidar, stereo cameras, and a high accuracy GPS/INS for ground truth acquisition. The method uses data from the Velodyne lidar. (b) Zoomed in view of the sensors.

Table 4.5: Configurations and results of the KITTI benchmark datasets. The errors are measured using segments of trajectories at 100m, 200m, ..., 800m lengthes based on 3D coordinates, as averaged percentages of the segment lengthes.

Data no.	Configuration		Mean relative position error
	Distance	Environment	
0	3714m	Urban	0.78%
1	4268m	Highway	1.43%
2	5075m	Urban + Country	0.92%
3	563m	Country	0.86%
4	397m	Country	0.71%
5	2223m	Urban	0.57%
6	1239m	Urban	0.65%
7	695m	Urban	0.63%
8	3225m	Urban + Country	1.12%
9	1717m	Urban + Country	0.77%
10	919m	Urban + Country	0.79%

compared to the GPS/INS ground truth are shown. On the middle and bottom rows, the map and a corresponding image is shown from each dataset. The maps are color coded by elevation. The complete test results with the 11 datasets are listed in Table 4.5. The three tests from left to right in Fig. 4.21 are datasets 0, 3, and 1 in the table. Here, the accuracy is calculated by averaging relative position errors using segmented trajectories at 100m, 200m, ..., 800m lengthes.

4.7 Conclusion

Motion estimation and mapping using point clouds from a rotating laser scanner can be difficult because the problem involves recovery of motion and correction of motion distortion in lidar clouds. The proposed method divides and solves the problem by two algorithms running in parallel. Cooperation of the two algorithms allows accurate motion estimation and mapping to be realized in real-time. The method has been tested in a large number of experiments using author collected data as well as the KITTI odometry benchmark.

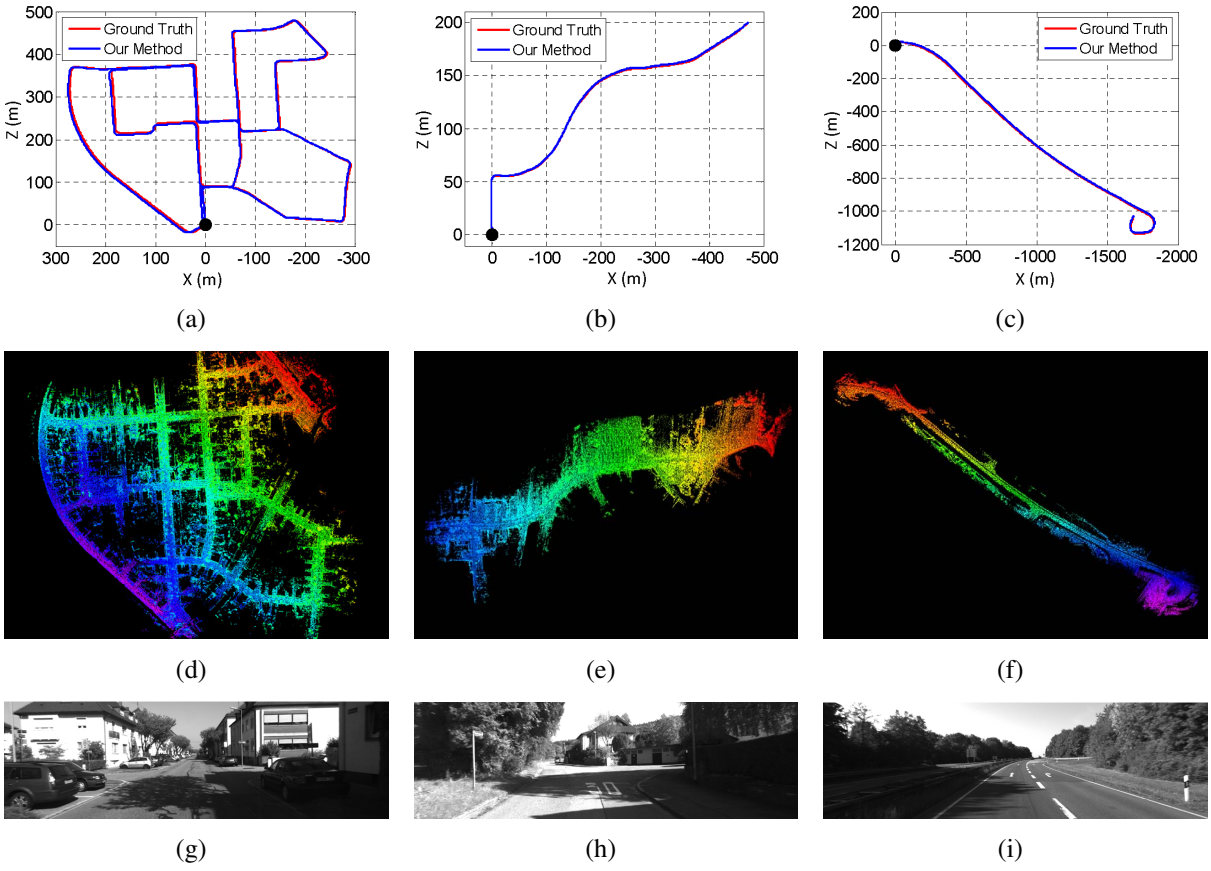


Figure 4.21: Sample results using the KITTI benchmark datasets. The datasets are chosen from three types of environments: urban, country, and highway from left to right, corresponding to tests number 0, 3, and 1 in Table 4.5. In (a)-(c), I compare estimated trajectories of the vehicle to the GPS/INS ground truth. The black dots are starting positions. (d)-(f) show maps corresponding to (a)-(c), color coded by elevation. An image is shown from each dataset to illustrate the environment, in (g)-(h).

Chapter 5

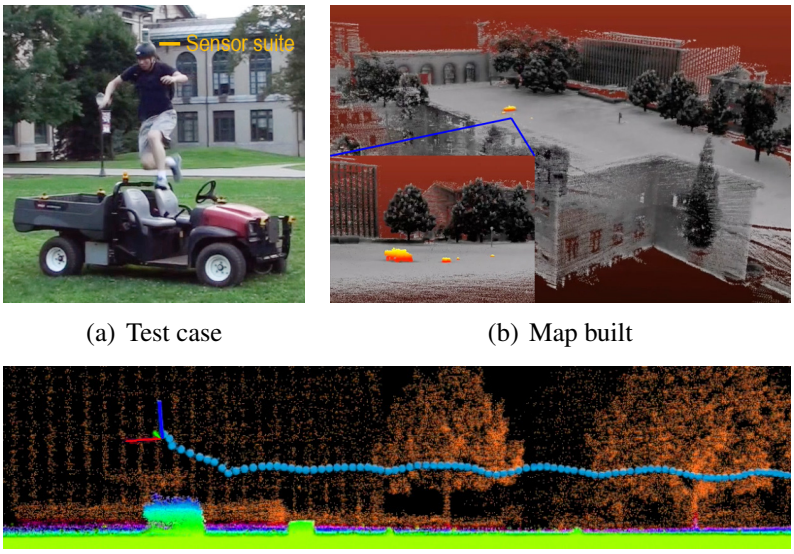
Vision-lidar Odometry and Mapping

This chapter presents a data processing pipeline to online estimate ego-motion and build a map of the traversed environment, using data from a 3D laser, a camera, and an IMU. Different from traditional methods that use a Kalman filter or factor-graph optimization, the proposed system employs a sequential, multi-layer processing pipeline, solving for motion from coarse to fine. Starting with IMU mechanization for motion prediction, a visual-inertial coupled method estimates motion, then a laser scan matching method further refines the motion estimates and registers maps. The resulting system enables high-frequency, low-latency ego-motion estimation, along with dense, accurate 3D map registration. Further, the system is capable of handling sensor degradation by automatic reconfiguration bypassing failure modules. Therefore, it can operate in the presence of highly dynamic motion as well as in dark, texture-less, and structure-less environments. During experiments, the system demonstrates 0.22% of relative position drift over 9.3km of navigation and robustness w.r.t running, jumping and even highway speed driving.

5.1 Introduction

This chapter aims at developing a software system for online ego-motion estimation with data from a 3D laser, a camera, and an IMU. The estimated motion further registers laser points to build a map of the traversed environment. In many real-world applications, ego-motion estimation and mapping must be conducted in real-time. In an autonomous navigation system, the map is crucial for motion planning and obstacle avoidance, while the motion estimation is important for vehicle control and maneuver. Very often, high-accuracy GPS/INS solutions are impractical when the application is GPS-denied, light-weight, or cost-sensitive. The proposed software system utilizes only perception sensors without reliance on GPS.

Specially, I am interested in solving for extremely aggressive motion. Yet, it remains nonobvious how to sove the problem reliably in 6-DOF, in real-time, and in a small form factor. The problem is closely relevant to sensor degradation due to sparsity of the data during dynamic maneuver. To the best of our knowledge, the proposed method is by far the first to enable such high-rate ego-motion estimation capable of handling running and jumping (see Fig. 5.1 as an example), while at the same time develop a dense, accurate 3D map, in the field under various lighting and structural conditions, and using only sensing and computing devices that can be



(a) Test case (b) Map built

(c) Trajectory estimated

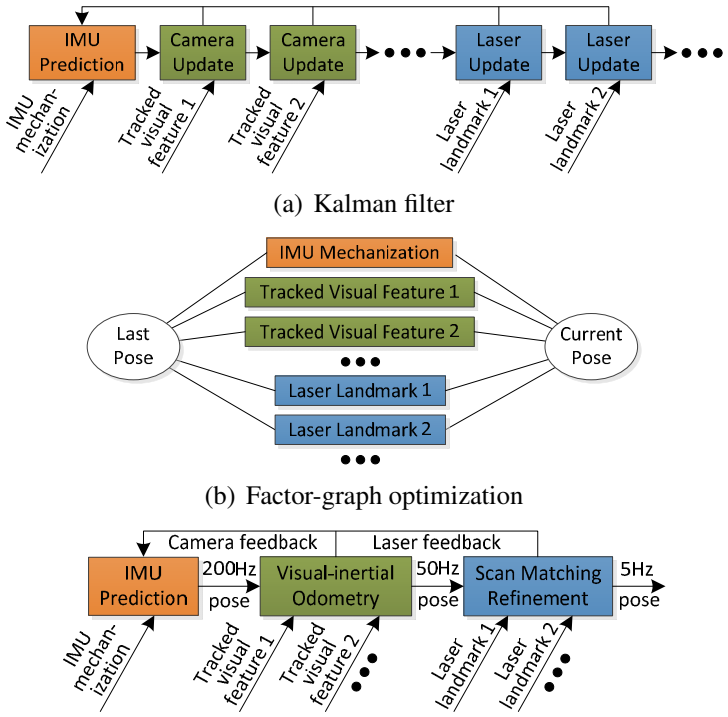
Figure 5.1: Representative results of the odometry and mapping software system. (a) An operator carries the sensor suite on a helmet and a processing computer in a backpack, running and jumping over a vehicle. The software system estimates ego-motion of the sensor suite in real-time, as well as develops a precise 3D map using only onboard processing. (b) presents the map built with the vehicle labeled in orange. (c) shows the estimated trajectory. The coordinate frame in (c) illustrates the sensor pose at the point in (a).

easily carried by a person.

The key reason that enables this level of performance is the novel way of data processing. As shown in Fig. 5.2(a), a standard Kalman filter based method typically uses IMU mechanization in a prediction step followed by update steps seeded with individual visual features or laser landmarks. On the other hand, a factor-graph optimization based method combines constraints from all sensors in one optimization problem (see Fig. 5.2(b)). In comparison, the modularized system sequentially recovers motion in a coarse-to-fine manner (see Fig. 5.2(c)). Starting with motion prediction from an IMU, a visual-inertial coupled method estimates motion and registers laser points locally. Then, a scan matching method further refines the estimated motion. The latest module also registers point clouds to build a map.

The system design follows an observation: drift in ego-motion estimation has a lower frequency than a module's own frequency. The three modules are therefore arranged in decreasing order of frequency. High-frequency modules are specialized to handle aggressive motion, while low-frequency modules cancel drift for the previous modules. The sequential processing also favors computation: modules in the front take less computation and execute at high frequencies, giving sufficient time to modules in the back for thorough processing. The system is therefore able to achieve a high level of accuracy while running online in real-time.

Further, the system is carefully configured to handle sensor degradation. If the camera is non-functional, e.g. due to darkness, dramatic lighting changes, or texture-less environments, or if the laser is non-functional, e.g. due to structure-less environments, the corresponding module



(a) Kalman filter

(b) Factor-graph optimization

(c) Proposed sequential pipeline

Figure 5.2: Diagram of the odometry and mapping software system. (a) shows a standard Kalman filter setup. IMU mechanization is used for prediction, then each visual feature and laser landmark seeds an individual update step. (b) shows a factor-graph optimization setup. All constraints from the IMU, visual features, and laser landmarks are combined in an optimization problem. (c) presents the proposed sequential data processing pipeline. Starting with IMU mechanization for prediction, a visual-inertial coupled method estimates ego-motion, then a scan matching method further refines the estimated motion. From left to right, motion is recovered from coarse to fine and accuracy is improved step by step to a high level. Further, both modules on the right provide feedback to correct velocity drift and biases of the IMU.

is bypassed and the rest of the system is staggered to function reliably. The system is tested through a large number of experiments and can produce high accuracy over several kilometers of navigation and robustness w.r.t. environmental degradation and aggressive motion.

5.2 Assumptions, Coordinates, and Problem

5.2.1 Assumptions and Coordinate Systems

Consider a sensor system including a laser, a camera, and an IMU, assume that the camera is modeled by a pin-hole camera model [37] and the intrinsic parameters are known [102]. The extrinsic parameters among the three sensors are calibrated. The relative pose between the camera and the laser is obtained based on [93], and the relative pose between the laser and the IMU is calibrated in the same way as [33] by solving a hand-eye problem [42]. As extrinsic calibration

is made, I use a single coordinate system for the camera and the laser. This is chosen to be the camera coordinate system – all laser points are projected into the camera coordinate system in pre-processing. For simplicity of notations, I also assume that the IMU coordinate system is parallel to the camera coordinate system – IMU measurements are rotationally corrected upon receiving. Define coordinate systems in the following,

- Camera coordinate system $\{C\}$ is originated at the camera optical center. The x-axis points to the left, the y-axis points upward, and the z-axis points forward coinciding with the camera principal axis.
- IMU coordinate system $\{I\}$ is originated at the IMU measurement center. The x-, y-, and z- axes are parallel to $\{C\}$ pointing to the same directions.
- World coordinate system $\{W\}$ is the coordinate system coinciding with $\{C\}$ at the start.

5.2.2 MAP Estimation Problem

A state estimation problem can be formulated as a *maximum a posterior* (MAP) estimation problem. Let us follow the definition in [17]. Define $\mathcal{X} = \{\boldsymbol{x}_i\}$, $i \in \{1, 2, ..., m\}$, as the set of system states, $\mathcal{U} = \{\boldsymbol{u}_i\}$, $i \in \{1, 2, ..., m\}$, as the set of control inputs, and $\mathcal{Z} = \{\boldsymbol{z}_k\}$, $k \in \{1, 2, ..., n\}$, as the set of landmark measurements. Given the proposed system, $\mathcal{Z}$ is composed of both visual features and laser landmarks. The joint probability of the system is defined as follows,

$$P(\mathcal{X}|\mathcal{U}, \mathcal{Z}) \propto P(\boldsymbol{x}_0) \prod_{i=1}^{m} P(\boldsymbol{x}_i|\boldsymbol{x}_{i-1}, u_i) \prod_{k=1}^{n} P(z_k|\boldsymbol{x}_{i_k}), \tag{5.1}$$

where $P(\boldsymbol{x}_0)$ is a prior of the first system state, $P(\boldsymbol{x}_i|\boldsymbol{x}_{i-1}, u_i)$ represents the motion model, and $P(z_k|\boldsymbol{x}_{i_k})$ represents the landmark measurement model. For each problem formulated as (5.1), there is a corresponding Bayesian belief network representation of the problem [49]. The MAP estimation is to maximize (5.1). Under the assumption of zero-mean Gaussian noise, the problem is equivalent to a least-square problem,

$$\mathcal{X}^* = \arg\min_{\mathcal{X}} \sum_{i=1}^{m} ||r_{\boldsymbol{x}_i}||^2 + \sum_{k=1}^{n} ||r_{\boldsymbol{z}_k}||^2. \tag{5.2}$$

Here, $r_{\boldsymbol{x}_i}$ and $r_{\boldsymbol{z}_k}$ are residual errors associated with the motion model and the landmark measurement model, respectively.

The standard way of solving (5.2) is to combine all sensor data, i.e. visual features, laser landmarks, and IMU measurements, into a large factor-graph optimization problem. The proposed data processing pipeline, instead, formulates multiple small optimization problems and solves the problems in a coarse to fine manner. The problem statement is,

Problem: Given data from a laser, a camera, and an IMU, formulate and solve problems as (5.2) to determine poses of $\{C\}$ w.r.t $\{W\}$, then use the estimated poses to register laser points and build a map of the traversed environment in $\{W\}$.

5.3 IMU Prediction Subsystem

5.3.1 IMU Mechanization

This subsection describes the IMU prediction subsystem. As the system considers $\{C\}$ as the fundamental sensor coordinate system, the IMU is also characterized w.r.t. $\{C\}$. Recall in Section 5.2.1, it is stated that $\{I\}$ and $\{C\}$ are parallel coordinate systems. Let $\boldsymbol{\omega}(t)$ and $\boldsymbol{a}(t)$ be two 3×1 vectors indicating the angular rates and accelerations of $\{C\}$ at time t. Let $\boldsymbol{b_\omega}(t)$ and $\boldsymbol{b_a}(t)$ be the corresponding biases, and $\boldsymbol{n_\omega}(t)$ and $\boldsymbol{n_a}(t)$ be the corresponding noises. All above terms are defined in $\{C\}$. Additionally, let $\boldsymbol{g}$ be the constant gravity vector in $\{W\}$. The IMU measurement terms are,

$$\hat{\boldsymbol{\omega}}(t) = \boldsymbol{\omega}(t) + \boldsymbol{b_\omega}(t) + \boldsymbol{n_\omega}(t), \tag{5.3}$$

$$\hat{\boldsymbol{a}}(t) = \boldsymbol{a}(t) - {}^{C}_{W}\mathbf{R}(t)\boldsymbol{g} - {}^{I}_{C}\boldsymbol{t}||\boldsymbol{\omega}(t)||^2 + \boldsymbol{b_a}(t) + \boldsymbol{n_a}(t), \tag{5.4}$$

where ${}^{C}_{W}\mathbf{R}(t)$ is the rotation matrix from $\{W\}$ to $\{C\}$, and ${}^{I}_{C}\boldsymbol{t}$ is the translation vector between $\{C\}$ and $\{I\}$.

Note that the term ${}^{I}_{C}\boldsymbol{t}||\boldsymbol{\omega}(t)||^2$ represents the centrifugal force due to the fact that the rotation center (origin of $\{C\}$) is different from the origin of $\{I\}$. State of the art visual-inertial navigation methods [48, 58] tend to model the motion in $\{I\}$ to eliminate this term. Since the method uses visual features both with and without depth information (discussed in Section 5.4.3), converting features without depth from $\{C\}$ to $\{I\}$ is not straight forward. I model the motion in $\{C\}$ instead. Practically, the camera and the IMU are mounted close to each other to maximally reduce effect of the term.

The IMU biases are slowly changing variables. I take the most recently updated biases for motion integration. First, (5.3) is integrated over time. Then, the resulting orientation is used with (5.4) for integration over time twice to obtain translation.

5.3.2 Bias Correction

The IMU bias correction can be made by feedback from either the camera or the laser. Each one contains the estimated incremental motion over a short amount of time. When calculating the biases, I model the biases to be constant during the incremental motion. Still starting with (5.3), by comparing the estimated orientation with IMU integration, I can calculate $\boldsymbol{b_\omega}(t)$. The updated $\boldsymbol{b_\omega}(t)$ is used in one more round of integration to recompute the translation, which is compared with the estimated translation to calculate $\boldsymbol{b_a}(t)$.

To reduce the effect of high-frequency noises, a sliding window is employed keeping a certain number of biases. The averaged biases from the sliding window are used. In this implementation, the length of the sliding window functions as a parameter determining update rate of the biases. A rigorous way is to model the biases as random walks and update the biases through optimization [30, 57]. However, this non-standard implementation is preferred to keep IMU processing in a separate module. The implementation favors dynamic reconfiguration of the system, i.e. the IMU can be coupled with either the camera or the laser. If the camera is non-functional, the IMU biases are corrected by the laser instead.

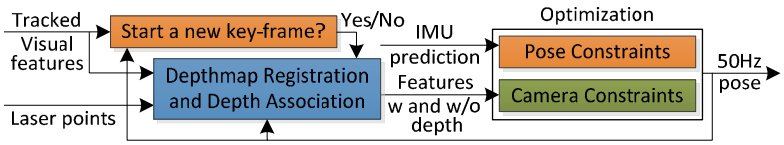


Figure 5.3: Diagram of the visual-inertial odometry subsystem.

5.4 Visual-inertial Odometry Subsystem

The section summarizes the visual-inertial odometry subsystem. This is based on a method proposed in the previous work [100]. A system diagram is shown in Fig. 5.3. The method couples vision with an IMU. Both provide constraints to an optimization problem that estimates incremental motion. At the same time, the method associates depth information to visual features. If a feature is located in an area where laser range measurements are available, depth is obtained from laser points. Otherwise, depth is calculated from triangulation using the previously estimated motion sequence. As the last option, the method can also use features without any depth by formulating constraints in a different way. This is true for those features which do not have laser range coverage or cannot be triangulated due to the fact that they are not tracked long enough or located in the direction of camera motion.

5.4.1 Camera Constraints

The visual-inertial odometry is a key-frame based method. A new key-frame is determined if more than a certain number of features lose tracking or the image overlap is below a certain ratio. Here, let us use right superscript $l \in Z^+$ to indicate the last key-frame, and c, $c \in Z^+$ and $c > k$, to indicate the current frame. As discussed, the method combines features with and without depth. For a feature that is associated with depth at key-frame l, denote it as $\boldsymbol{X}_l = [x_l,\ y_l,\ z_l]^T$ in $\{C_l\}$. Correspondingly, a feature without depth is denoted as $\bar{\boldsymbol{X}}_l = [\bar{x}_l,\ \bar{y}_l,\ 1]^T$ using normalized coordinates instead. Note that $\boldsymbol{X}_l$, $\bar{\boldsymbol{X}}_l$, x_l, and $\bar{x}_l$ are different from $\mathcal{X}$ and $\boldsymbol{x}$ in (5.1) which represent the system state. I only associate depth to features at key-frames, for two reasons: 1) depth association takes some processing and executing at key-frames only helps reduce computation intensity; and 2) laser points need to be registered on a depthmap (more discussion in Section 5.4.3), however, the transform to frame c has not been established at this point and the depthmap is not available at frame c. Denote a normalized feature in $\{C_c\}$ as $\bar{\boldsymbol{X}}_c = [\bar{x}_c,\ \bar{y}_c,\ 1]^T$.

Let $\mathbf{R}_l^c$ and $\boldsymbol{t}_l^c$ be the 3×3 rotation matrix and 3×1 translation vector between frames l and c, where $\mathbf{R}_l^c \in \mathbf{SO}(3)$ and $\boldsymbol{t}_l^c \in \mathbb{R}^3$, $\mathbf{R}_l^c$ and $\boldsymbol{T}_l^c$ form an $\mathbf{SE}(3)$ transformation [66]. The motion function between frames l and c is written as,

$$\boldsymbol{X}_c = \mathbf{R}_l^c \boldsymbol{X}_l + \boldsymbol{t}_l^c. \tag{5.5}$$

Recall that $\boldsymbol{X}_c$ has unknown depth, let d_c be the depth, where $\boldsymbol{X}_c = d_c \bar{\boldsymbol{X}}_c$. Substituting $\boldsymbol{X}_c$ with $d_c \bar{\boldsymbol{X}}_c$ and combining the 1st and 2nd rows with the 3rd row in (5.5) to eliminate d_c, I have,

$$(\boldsymbol{R}(1) - \bar{x}_c \boldsymbol{R}(3)) \boldsymbol{X}_l + t(1) - \bar{x}_c t(3) = 0, \tag{5.6}$$

$$(\boldsymbol{R}(2) - \bar{y}_c \boldsymbol{R}(3)) \boldsymbol{X}_l + t(2) - \bar{y}_c t(3) = 0. \tag{5.7}$$

Here, $\boldsymbol{R}(h)$ and $t(h)$, $h \in \{1,2,3\}$, are the h-th rows of $\mathbf{R}_l^c$ and $\boldsymbol{t}_l^c$. In the case that depth in unavailable to a feature, let d_l be the unknown depth at key-frame l. Substituting $\boldsymbol{X}_l$ and $\boldsymbol{X}_c$ with $d_k\bar{\boldsymbol{X}}_l$ and $d_c\bar{\boldsymbol{X}}_c$, respectively, and combining all three rows in (5.5) to eliminate d_k and d_c, I obtain another constraint,

$$[\bar{y}_c t(3) - t(2),\ -\bar{x}_c t(3) + t(1),\ \bar{x}_c t(2) - \bar{y}_c t(1)]\mathbf{R}_l^c\bar{\boldsymbol{X}}_l = 0. \tag{5.8}$$

5.4.2 Motion Estimation

The motion estimation is to solve an optimization problem combining three sets of constraints: 1) from features with known depth as (5.6)-(5.7); 2) from features with unknown depth as (5.8); and 3) from the IMU prediction. Let us define $\mathbf{T}_a^b$ as the 4×4 transformation matrix between frames a and b,

$$\mathbf{T}_a^b = \begin{bmatrix} \mathbf{R}_a^b & \boldsymbol{t}_a^b \\ \boldsymbol{0}^T & 1 \end{bmatrix}, \tag{5.9}$$

where $\mathbf{R}_a^b$ and $\boldsymbol{t}_a^b$ are the corresponding rotation matrix and translation vector. Further, let $\boldsymbol{\theta}_a^b$ be a 3×1 vector corresponding to $\mathbf{R}_a^b$ through an exponential map [66], where $\boldsymbol{\theta}_a^b \in \mathfrak{so}(3)$. The normalized term $\boldsymbol{\theta}/||\boldsymbol{\theta}||$ represents direction of the rotation and $||\boldsymbol{\theta}||$ is the rotation angle. Each $\mathbf{T}_a^b$ corresponds to a set of $\boldsymbol{\theta}_a^b$ and $\boldsymbol{t}_a^b$ containing 6-DOF motion of the camera.

To formulate the IMU pose constraints, let us take the solved motion transform between frames l and $c-1$, namely $\mathbf{T}_l^{c-1}$. From IMU mechanization, I obtain a predicted transform between the last two frames $c-1$ and c, denoted as $\hat{\mathbf{T}}_{c-1}^c$. The predicted transform at frame c is,

$$\hat{\mathbf{T}}_l^c = \hat{\mathbf{T}}_{c-1}^c \mathbf{T}_l^{c-1}. \tag{5.10}$$

Let $\hat{\boldsymbol{\theta}}_l^c$ and $\hat{\boldsymbol{t}}_l^c$ be the 6-DOF motion corresponding to $\hat{\mathbf{T}}_l^c$. It is worth to mention that the IMU predicted translation, $\hat{\boldsymbol{t}}_l^c$, is dependent on the orientation, i.e. the orientation determines projection of the gravity vector through rotation matrix ${}_W^C\mathbf{R}(t)$ in (5.4), and hence the accelerations being integrated. I formulate $\hat{\boldsymbol{t}}_l^c$ as a function of $\boldsymbol{\theta}_l^c$, and rewrite it as $\hat{\boldsymbol{t}}_l^c(\boldsymbol{\theta}_l^c)$ from now on. At this point, I should clarify that the 200Hz pose and so as the 50Hz pose in Fig. 5.2(c) are in indeed pose functions. When calculating $\hat{\boldsymbol{t}}_l^c(\boldsymbol{\theta}_l^c)$, I start at frame c and integrate accelerations inversely w.r.t. time. Let $\boldsymbol{\theta}_l^c$ be the rotation vector corresponding to $\mathbf{R}_l^c$ in (5.5), $\boldsymbol{\theta}_l^c$ and $\boldsymbol{t}_l^c$ are the motion to be solved. The constraints are expressed as,

$$\Sigma_l^c[(\hat{\boldsymbol{\theta}}_l^c - \boldsymbol{\theta}_l^c)^T,\ (\hat{\boldsymbol{t}}_l^c(\boldsymbol{\theta}_l^c) - \boldsymbol{t}_l^c)^T]^T = \boldsymbol{0}, \tag{5.11}$$

where Σ_l^c is a relative covariance matrix scaling the pose constraints appropriately w.r.t. the camera constraints.

In the visual-inertial odometry subsystem, the pose constraints fulfill the motion model and the camera constraints fullfill the landmark measurement model in (5.2). The optimization problem is solved by the Newton gradient-descent method [71] adapted to a robust fitting framework [1] for outlier feature removal. In this problem, the state space contains $\boldsymbol{\theta}_l^c$ and $\boldsymbol{t}_l^c$. In other words, I do not perform full-scale MAP estimation but only solve a marginalized problem. The landmark positions are not optimized. This means only six unknowns in the state space keeping

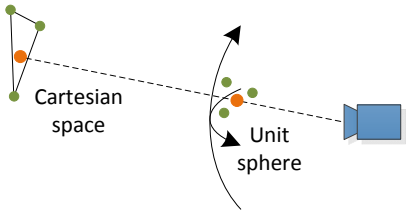


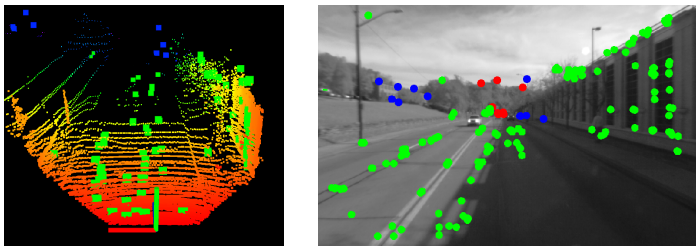
Figure 5.4: Illustration of feature depth association. Features (orange) and laser points on the demthmap (green) are projected from Cartesian space onto a unit sphere. Then, the three closest laser points on the sphere are found for each feature using a 2D KD-tree. The three points form a local planar patch in Cartesian space and the depth is interpolated for the feature.

computation intensity low. The argument is that the method involves laser range measurements to provide precise depth information to features, warranting motion estimation accuracy. Further optimizing the features' depth in bundle adjustment is practically unnecessary.

5.4.3 Depth Association

The method registers laser points on a depthmap using previously estimated motion. Laser points within the camera field of view are kept for a certain amount of time. The depthmap is down-sampled to keep a constant density and stored in a 2D KD-tree [16] for fast index. In the KD-tree, all laser points are projected onto a unit sphere around the camera center (as in Fig. 5.4). A point is represented by its two angular coordinates. When associating depth to features, I project the features onto the sphere. The three closest laser points are found on the sphere for each feature. Then, I further check their validity by calculating distances among the three points in Cartesian space. If a distance is larger than a threshold, the chance that the points are from different objects, e.g. a wall and an object in front of the wall, is high and the validity check fails. Finally, the depth is interpolated from the three points assuming a local planar patch in Cartesian space.

For those features without laser range coverage, if they are tracked over a certain distance and not located in the direction of camera motion, I triangulate them using the image sequences where the features are tracked. This uses a similar procedure as [29, 94], where the depth is updated at each frame based on a Bayesian probabilistic mode. Fig. 5.5 shows an example depthmap and



(a) (b)

Figure 5.5: (a) Example depthmap (colored points) and 3D projected visual features. The green points are features whose depth is from the depthmap. The blue points are by triangulation. (b) Corresponding features in an image. The red points have unknown depth.

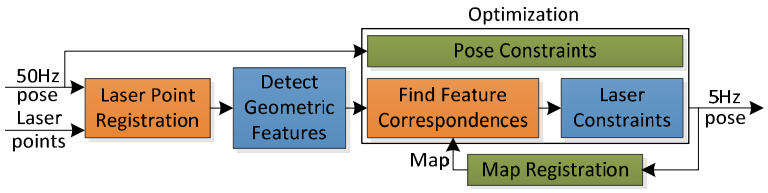


Figure 5.6: Diagram of the scan matching subsystem.

3D projected features. The green points have depth from the depthmap, the blue points are by triangulation, and the red points have unknown depth.

5.5 Scan Matching Subsystem

This subsystem further refines motion estimates from the previous module by laser scan matching. The method is based on the previous work published in [97]. A diagram is present in Fig. 5.6. The subsystem registers laser points in a local point cloud using provided odometry estimation. Then, geometric features are detected from the point cloud and matched to the map. The scan matching minimizes the feature-to-map distances, similar to many existing methods [31]. However, the odometry estimation from the previous module also provides pose constraints in the optimization. The implementation uses voxel representation of the map. Further, it can dynamically configure to run on one to multiple CPU threads in parallel.

5.5.1 Laser Constraints

When receiving laser scans, the method first registers points from a scan into a common coordinate system. Let us use $m \in Z^+$ to indicate the scan number. Recall that the camera coordinate system is used for both the camera and the laser. For scan m, I associate it with the camera coordinate system at the beginning of the scan, denoted as $\{C_m\}$. To locally register the laser points, I take the odometry estimation from the visual-inertial odometry as key-points, and use IMU measurements to interpolate in between the key-points.

Let $\mathcal{P}_m$ be the locally registered point cloud from scan m. I extract two sets of geometric features from $\mathcal{P}_m$, one on sharp edges, namely edge points and denoted as $\mathcal{E}_m$, and the other on local planar surfaces, namely planar points and denoted as $\mathcal{H}_m$. When extracting geometric features, I use the term defined in [97] to evaluate smoothness of local surfaces around the points. Denote a point as $\boldsymbol{X}^i_m = [x^i_m,\ y^i_m,\ z^i_m]^T$, $\boldsymbol{X}^i_m \in \mathcal{P}_m$. The set of surrounding points of $\boldsymbol{X}^i_m$ in $\mathcal{P}_m$ is denoted as $\mathcal{C}^i_m$. Given that orientations of the points around $\boldsymbol{X}^i_m$ from a laser scanner distribute evenly, the smoothness is defined as,

$$c^i_m = \frac{1}{|\mathcal{C}^i_m| \cdot ||\boldsymbol{X}^i_m||} || \sum_{\boldsymbol{X}^j_m \in \mathcal{C}^i_m, j \neq i} (\boldsymbol{X}^i_m - \boldsymbol{X}^j_m) ||. \tag{5.12}$$

Edge points and planar points are extracted with large and small c^i_m values, respectively. I avoid selecting points whose neighbor points are already selected, points on boundaries of occluded regions (point B in Fig. 5.7), or points whose local surfaces are close to be parallel to laser

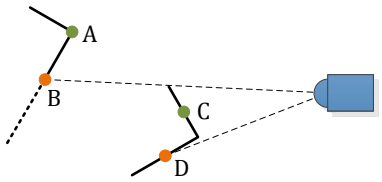


Figure 5.7: Illustration of geometric feature selection. Point A is a good edge point but point B is not. This is because point B is on the boundary of an occluded region, making it appear to be an edge point. Point C is a good planar point but point D is not, because the local surface of point D is close to be parallel to the laser beam. Points B and D are likely to contain large noises or change positions as the sensor moves, hence are not selected.

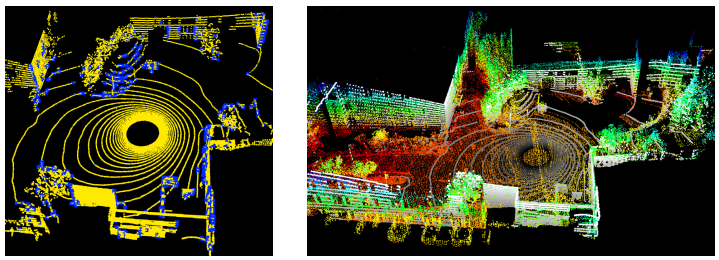
beams (point D in Fig. 5.7). These points are likely to contain large noises or change positions over time as the sensor moves. Fig. 5.8(a) gives an example of detected edge points (blue) and planar points (yellow).

The geometric features are then matched to the current map built. Let $\mathcal{Q}_{m-1}$ be the map point cloud after processing the last scan, $\mathcal{Q}_{m-1}$ is defined in $\{W\}$. The points in $\mathcal{Q}_{m-1}$ are separated into two sets containing edge points and planar points, respectively. Voxels are used to store the map truncated at a certain distance around the sensor. For each voxel, I construct two 3D KD-trees [16], one for edge points and the other for planar points. Using KD-trees for individual voxels accelerates point searching since given a query point, I only need to search in a specific KD-tree associated with a single voxel (more discussion in Section 5.5.3).

When matching scans, I first project $\mathcal{E}_m$ and $\mathcal{H}_m$ into $\{W\}$ using the best guess of motion available, then for each point in $\mathcal{E}_m$ and $\mathcal{H}_m$, a cluster of closest points are found from the corresponding set on the map. To verify geometric distributions of the point clusters, I examine the associated eigenvalues and eigenvectors. Specifically, one large and two small eigenvalues indicate an edge line segment, and two large and one small eigenvalues indicate a local planar patch. If the matching is valid, an equation is formulated regarding the distance from a point to its correspondence,

$$d_m^i = f(\boldsymbol{X}_m^i, \boldsymbol{\theta}_m, \boldsymbol{t}_m), \tag{5.13}$$

where $\boldsymbol{X}_m^i \in \mathcal{E}_m$ or $\boldsymbol{X}_m^i \in \mathcal{H}_m$, $\boldsymbol{\theta}_m \in \mathfrak{so}(3)$ and $\boldsymbol{t}_m \in \mathbb{R}^3$ indicate the 6-DOF pose of $\{C_m\}$



(a) (b)

Figure 5.8: (a) Example edge points (blue) and planar points (yellow) detected from a scan. (b) Matching a scan (grey points) to the map (colored points), then the scan is merged with the map to extend the map further.

in $\{W\}$. Specifically, if $\boldsymbol{X}_m^i$ is an edge point, (5.13) describes the distance between $\boldsymbol{X}_m^i$ and the corresponding edge line segment,

$$d_m^i = \frac{\left|(\hat{\boldsymbol{X}}_m^i(\boldsymbol{\theta}_m, \boldsymbol{t}_m) - \hat{\boldsymbol{X}}_{m-1}^j) \times (\hat{\boldsymbol{X}}_m^i(\boldsymbol{\theta}_m, \boldsymbol{t}_m) - \hat{\boldsymbol{X}}_{m-1}^k)\right|}{\left|\hat{\boldsymbol{X}}_{m-1}^j - \hat{\boldsymbol{X}}_{m-1}^k\right|}, \tag{5.14}$$

where $\hat{\boldsymbol{X}}_m^i(\boldsymbol{\theta}_m, \boldsymbol{t}_m)$ is the projected point of $\hat{\boldsymbol{X}}_m^i$ into $\{W\}$ using $\boldsymbol{\theta}_m$ and $\boldsymbol{t}_m$, $\hat{\boldsymbol{X}}_{m-1}^j$ and $\hat{\boldsymbol{X}}_{m-1}^k$ are two points located on the edge line segment in $\{W\}$. If $\boldsymbol{X}_m^i$ is a planar point, (5.13) describes the distance between $\boldsymbol{X}_m^i$ and the corresponding local planar patch,

$$d_m^i = \frac{\left|\begin{array}{c}(\hat{\boldsymbol{X}}_m^i(\boldsymbol{\theta}_m, \boldsymbol{t}_m) - \hat{\boldsymbol{X}}_{m-1}^j) \\ (\hat{\boldsymbol{X}}_{m-1}^j - \hat{\boldsymbol{X}}_{m-1}^k) \times (\hat{\boldsymbol{X}}_{m-1}^j - \hat{\boldsymbol{X}}_{m-1}^l))\end{array}\right|}{\left|(\hat{\boldsymbol{X}}_{m-1}^j - \hat{\boldsymbol{X}}_{m-1}^k) \times (\hat{\boldsymbol{X}}_{m-1}^j - \hat{\boldsymbol{X}}_{m-1}^l)\right|}, \tag{5.15}$$

where $\hat{\boldsymbol{X}}_{m-1}^j$, $\hat{\boldsymbol{X}}_{m-1}^k$, $\hat{\boldsymbol{X}}_{m-1}^l$ are three points located on the local planar patch in $\{W\}$. Fig. 5.8(b) shows an example where a scan is matched. The gray points are from a scan and the colored points are from the map.

5.5.2 Motion Estimation

The scan matching is formulated into an optimization problem minimizing the overall distances described by (5.13). The optimization also involves pose constraints from prior motion. Let $\mathbf{T}_{m-1}$ be the 4×4 transformation matrix regarding the pose of $\{C_{m-1}\}$ in $\{W\}$, $\mathbf{T}_{m-1}$ is generated by processing the last scan. Let $\hat{\mathbf{T}}_{m-1}^m$ be the pose transform from $\{C_{m-1}\}$ to $\{C_m\}$, as provided by the odometry estimation. Similar to (5.10), the predicted pose transform of $\{C_m\}$ in $\{W\}$ is,

$$\hat{\mathbf{T}}_m = \hat{\mathbf{T}}_{m-1}^m \mathbf{T}_{m-1}. \tag{5.16}$$

Let $\hat{\boldsymbol{\theta}}_m$ and $\hat{\boldsymbol{t}}_m$ be the 6-DOF pose corresponding to $\hat{\mathbf{T}}_m$, and let Σ_m be a relative covariance matrix. The constraints are,

$$\Sigma_m[(\hat{\boldsymbol{\theta}}_m - \boldsymbol{\theta}_m)^T,\ (\hat{\boldsymbol{t}}_m - \boldsymbol{t}_m)^T]^T = \boldsymbol{0}. \tag{5.17}$$

Eq. (5.17) refers to the case that the prior motion is from the visual-inertial odometry, assuming the camera is functional. Otherwise, the constraints are from the IMU prediction. Let us use $\hat{\boldsymbol{\theta}}_m'$ and $\hat{\boldsymbol{t}}_m'(\boldsymbol{\theta}_m)$ to denote the same terms by IMU mechanization. $\hat{\boldsymbol{t}}_m'(\boldsymbol{\theta}_m)$ is a function of $\boldsymbol{\theta}_m$ due to the fact that integration of accelerations is dependent on the orientation (same with $\hat{\boldsymbol{t}}_l^c(\boldsymbol{\theta}_l^c)$ in (5.11)). The IMU pose constraints are,

$$\Sigma_m'[(\hat{\boldsymbol{\theta}}_m' - \boldsymbol{\theta}_m)^T,\ (\hat{\boldsymbol{t}}_m'(\boldsymbol{\theta}_m) - \boldsymbol{t}_m)^T]^T = \boldsymbol{0}, \tag{5.18}$$

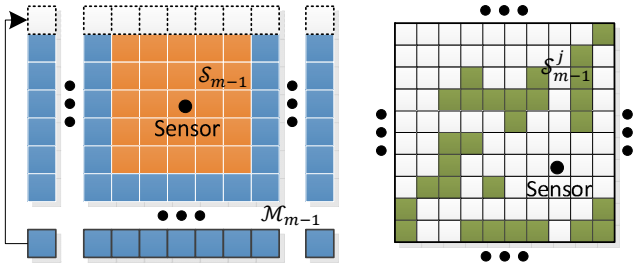
where Σ_m' is the corresponding relative covariance matrix. In the optimization problem, (5.17) and (5.18) are linearly combined into one set of constraints. The linear combination is determined by working mode of the visual-inertial odometry (recapped in Section 5.7). The optimization problem refines $\boldsymbol{\theta}_m$ and $\boldsymbol{t}_m$, which is solved by the Newton gradient-descent method [71] adapted to a robust fitting framework [1].

5.5.3 Map in Voxels

The points on the map are kept in voxels. I use a 2-level voxel implementation as illustrated in Fig. 5.9(a). Let us use $\mathcal{M}_{m-1}$ to indicate the set of voxels on the map after processing the last scan. Voxels surrounding the sensor form a subset of $\mathcal{M}_{m-1}$, denoted as $\mathcal{S}_{m-1}$. Given a 6-DOF sensor pose, $\hat{\boldsymbol{\theta}}_m$ and $\hat{\boldsymbol{t}}_m$, there is a corresponding $\mathcal{S}_{m-1}$ which moves with the sensor on the map. When the sensor approaches the boundary of the map, as an example in Fig. 5.9(a), voxels on the opposite side of the boundary are moved over to extend the map boundary. Points in moved voxels are cleared resulting in truncation of the map.

As illustrated in Fig. 5.9(b), each voxel $j \in \mathcal{S}_{m-1}$ is formed by a set of voxels that are a magnitude smaller, denoted as $\mathcal{S}^j_{m-1}$. Before matching scans, I project points in $\mathcal{E}_m$ and $\mathcal{H}_m$ onto the map using the best guess of motion, and fill them into $\{\mathcal{S}^j_{m-1}\}$, $j \in \mathcal{S}_{m-1}$. Voxels occupied by points from $\mathcal{E}_m$ and $\mathcal{H}_m$ are labeled in green. Then, map points in green voxels are extracted to form $\mathcal{Q}_{m-1}$ and stored in 3D KD-trees for scan matching. An example of $\mathcal{Q}_{m-1}$ is the colored points in Fig. 5.8(b). Upon completion of scan matching, the scan is merged into the green voxels with the map. After that, the map points are downsized for a constant density.

One question regarding the algorithm implementation is why using two levels of voxels instead of one. The idea is that I use $\mathcal{M}_{m-1}$ to keep the map and $\{\mathcal{S}^j_{m-1}\}$, $j \in \mathcal{S}_{m-1}$ to retrieve the map around the sensor for scan matching. The map is truncated only when the sensor approaches the map boundary. In other words, if the sensor navigates inside the map, no truncation is needed. Another consideration is that I use two KD-trees for each individual voxel in $\mathcal{S}_{m-1}$ - one for edge points and the other for planar points. As discussed, this accelerates point searching. At the same time, I would like to avoid building too many KD-trees as opposed to using two KD-trees for each individual voxel in $\{\mathcal{S}^j_{m-1}\}$, $j \in \mathcal{S}_{m-1}$. The later requires more resources for KD-tree building and maintenance.



(a) (b)

Figure 5.9: (a) Voxels on the map $\mathcal{M}_{m-1}$ (all voxels in (a)), and voxels surrounding the sensor $\mathcal{S}_{m-1}$ (orange voxels). $\mathcal{S}_{m-1}$ is a subset of $\mathcal{M}_{m-1}$. If the sensor approaches the boundary of the map, voxels on the opposite side of the boundary (bottom row) are moved over to extend the map boundary. Points in moved voxels are cleared and the map is truncated. (b) Each voxel $j \in \mathcal{S}_{m-1}$ (an orange voxel in (a)) is formed by a set of voxels $\mathcal{S}^j_{m-1}$ that are a magnitude smaller (all voxels in (b) $\in \mathcal{S}^j_{m-1}$). Before scan matching, I project points in $\mathcal{E}_m$ and $\mathcal{H}_m$ onto the map using the best guess of motion. Voxels in $\{\mathcal{S}^j_{m-1}\}$, $j \in \mathcal{S}_{m-1}$ occupied by points from $\mathcal{E}_m$ and $\mathcal{H}_m$ are labeled in green. Then, map points in green voxels are extracted as $\mathcal{Q}_{m-1}$. The figure is drawn in 2D but the algorithm is implemented in 3D.

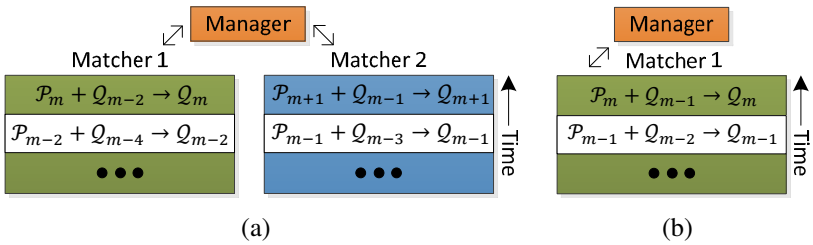
Table 5.1: Comparsion of average CPU processing time on KD-tree operation

Task	1-level voxels		2-level voxels	
	KD-trees for all voxels	KD-trees for each voxel	KD-trees for all voxels	KD-trees for each voxel
Build (time per KD-tree)	54ms	47ms	24ms	**21ms**
Query (time per point)	4.2ns	4.1ns	2.4ns	**2.3ns**

Table 5.1 compares CPU processing time using different voxel and KD-tree configurations. The time is averaged from multiple datasets collected from different types of environments covering confined and open, structured and vegetated areas. One sees that using only one level of voxels, $\mathcal{M}_{m-1}$, results in about twice of processing time for KD-tree building and querying. This is because the second level of voxels, $\{\mathcal{S}_{m-1}^{j}\}$, $j \in \mathcal{S}_{m-1}$, help retrieve the map precisely. Without these voxel, more points are contained in $\mathcal{Q}_{m-1}$ and built into the KD-trees. Also, one sees that using KD-trees for each voxel helps reduce processing time sightly in comparison to using KD-trees for all voxels in $\mathcal{M}_{m-1}$.

5.5.4 Parallel Processing

The scan matching is time-consuming and takes major computation in the system. This involves building KD-trees, repetitively querying geometric feature points, and matrix manipulations for nonlinear optimization. Without taking advantage from a powerful GPU, the chapter employs a CPU based multi-thread implementation warranting the desired frequency. Fig. 5.10(a) illustrates the case where two scans are matched in parallel. Upon receiving of a scan, a manager program arranges it to match with the latest map available. In a clustered environment with plenty of structures, matching is slow and may not complete before arrival of the next scan. Two matcher programs are called alternatively. On each matcher, $\mathcal{P}_m$, $\mathcal{P}_{m-1}$, ..., are matched with $\mathcal{Q}_{m-2}$,



(a) (b)

Figure 5.10: Illustration of multi-thread scan matching. A manager program calls multiple matcher programs running on separate CPU threads and matches scans to the latest map available. (a) shows a two-thread case. Scans $\mathcal{P}_m$, $\mathcal{P}_{m-1}$, ..., are matched with map $\mathcal{Q}_{m-2}$, $\mathcal{Q}_{m-3}$, ..., on each matcher, giving twice amount of time for processing. In comparison, (b) shows a one-thread case, where $\mathcal{P}_m$, $\mathcal{P}_{m-1}$, ..., are matched with $\mathcal{Q}_{m-1}$, $\mathcal{Q}_{m-2}$, ... The implementation is dynamically configurable using up to four threads.

$\mathcal{Q}_{m-3}$, ..., respectively, giving twice amount of time for processing. On the other hand, in a clean environment with few structures, computation is light. Only the first matcher is called as in Fig. 5.10(b), and $\mathcal{P}_m$, $\mathcal{P}_{m-1}$, ..., are matched with $\mathcal{Q}_{m-1}$, $\mathcal{Q}_{m-2}$, ..., respectively. The implementation is configured to use maximally four threads, however it is uncommon that more than two threads are needed.

An alternative way is to process a single scan on multiple CPU threads in parallel. However, this will unavoidably cause efficiency loss. Lu and Hart's research [60] indicates that constructing KD-trees using four threads with low-dimensional data results in 87.5% of the original efficiency. Further, the multi-thread implementation is dynamically configured to arrange all processing on the minimum number of threads. This ensures low-latency ego-motion estimation as it leaves room on other threads for the real-time processing.

5.6 Transform Integration

The final motion estimation is integration of outputs from the three modules in Fig. 5.2(c). As illustrated in Fig. 5.11, the 5Hz scan matching output (blue section) fundamentally warrants accuracy. The 50Hz visual-inertial odometry output (green section) and the 200Hz IMU prediction (orange section) are integrated to the front for high-frequency motion estimates.

5.7 On Robustness

The robustness of the system is determined by its ability to handle sensor degradation. Assume the IMU is always reliable functioning as the backbone in the system. Camera is sensitive to dramatic lighting changes. It also fails in a dark/texture-less environment or when significant motion blur is present causing visual features lose tracking. Laser cannot handle structure-less environments, e.g. a scene that is dominant by a plane. Further, the same degradation can be caused by sparsity of the data due to aggressive motion.

The method that is used to deal with these failures is originally proposed in [101]. Both the visual-inertial odometry and the scan matching modules formulate and solve optimization problems as (5.2). When a failure happens, it corresponds to a degraded optimization problem, i.e. some directions of the state space are loosely constrained and noises are dominate in determining the solution in these directions. Let $\mathbf{J}$ be the Jacobian matrix associated with (5.2), the method starts with computing eigenvalues, denoted as λ_1, λ_2, ..., λ_6, and eigenvectors, denoted as $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, ..., $\boldsymbol{v}_6$, of $\mathbf{J}^T\mathbf{J}$. Here, six eigenvalues/eigenvectors are present because the state space

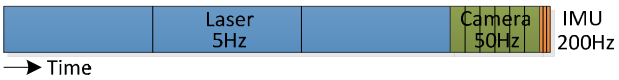


Figure 5.11: Illustration of transform integration. The final motion estimation is integration of the 5Hz scan matching output (blue), the 50Hz visual-inertial odometry output (green), and the 200Hz IMU prediction (orange), at the IMU frequency. The density of the vertical line segments in each section indicate the motion estimation frequency from the corresponding module.

contains 6-DOF motion of the sensor. Without loosing generality, $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, ..., $\boldsymbol{v}_6$ are sorted in decreasing order. Each eigenvalue describes how well the solution is conditioned in the direction of its corresponding eigenvector. By comparing the eigenvalues to a threshold, I can separate well-conditioned directions from degraded directions in the state space. Let h, $h = 0, 1, ..., 6$, be the number of well-conditioned directions. Here, define two matrices,

$$\mathbf{V} = [\boldsymbol{v}_1, ..., \boldsymbol{v}_6]^T,\ \bar{\mathbf{V}} = [\boldsymbol{v}_1, ..., \boldsymbol{v}_h, 0, ..., 0]^T. \tag{5.19}$$

When solving an optimization problem, the nonlinear iteration starts with an initial guess. With the sequential pipeline in Fig. 5.2(c), the IMU prediction provides the initial guess for the visual-inertial odometry, whose output is taken as the initial guess for the scan matching. For the last two modules, let $\boldsymbol{x}$ be a solution and $\Delta\boldsymbol{x}$ be an update of $\boldsymbol{x}$ in a nonlinear iteration. Given the Newton gradient-descent method [71] used by the chapter, $\Delta\boldsymbol{x}$ is calculated as,

$$\Delta\boldsymbol{x} = -(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\boldsymbol{b}. \tag{5.20}$$

Here, $\boldsymbol{b}$ is a matrix containing residuals of the linearized problem. During the optimization process, instead of updating $\boldsymbol{x}$ in all directions, I only update $\boldsymbol{x}$ in well-conditioned directions, keeping the initial guess in degraded directions instead,

$$\boldsymbol{x} \leftarrow \boldsymbol{x} + \mathbf{V}^{-1}\bar{\mathbf{V}}\Delta\boldsymbol{x}. \tag{5.21}$$

Let us further explain the intuition behind (5.21). The system solves for motion in a coarse-to-fine order, starting with the IMU prediction, the following two modules further solve/refine the motion as much as possible, fully (in 6-DOF) if the problem is well-conditioned, and partially (in 0 to 5-DOF) otherwise. If the problem is completely degraded, $\bar{\mathbf{V}}$ becomes a zero matrix and the previous module's output is kept.

Recap: Let us recap on the pose constraints described in (5.17) and (5.18). In fact, the two equations are linearly combined in the scan matching problem. Let us use $\mathbf{V}_V$ and $\bar{\mathbf{V}}_V$ to denote the matrices defined in (5.19) containing eigenvectors from the visual-inertial odometry module, $\bar{\mathbf{V}}_V$ represents well-conditioned directions in the subsystem, and $\mathbf{V}_V - \bar{\mathbf{V}}_V$ represents degraded directions. The combined constraints are,

$$\begin{aligned}&\Sigma_m \mathbf{V}_V^{-1}\bar{\mathbf{V}}_V[(\hat{\boldsymbol{\theta}}_m - \boldsymbol{\theta}_m)^T,\ (\hat{\boldsymbol{t}}_m - \boldsymbol{t}_m)^T]^T + \Sigma'_m \mathbf{V}_V^{-1} \\ &(\mathbf{V}_V - \bar{\mathbf{V}}_V)[(\hat{\boldsymbol{\theta}}'_m - \boldsymbol{\theta}_m)^T,\ (\hat{\boldsymbol{t}}'_m(\boldsymbol{\theta}_m) - \boldsymbol{t}_m)^T]^T = \boldsymbol{0}.\end{aligned} \tag{5.22}$$

In a normal case where the camera is functional, $\bar{\mathbf{V}}_V = \mathbf{V}_V$ and (5.22) is composed of pose constraints from the visual-inertial odometry as (5.17). On the other hand, if the camera is completely degraded, $\bar{\mathbf{V}}_V$ is a zero matrix and (5.22) is composed of pose constraints from the IMU prediction as (5.18).

5.7.1 Case Study of Camera Degradation

As shown in Fig. 5.12(a), if visual features are insufficiently available for the visual-inertial odometry, the IMU prediction bypasses the green block fully or partially, depending on the number of well-conditioned directions in the visual-inertial odometry problem, and locally registers

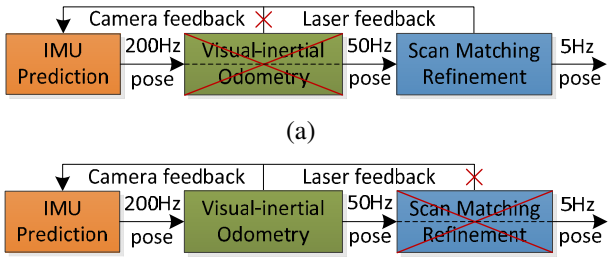


Figure 5.12: Case study of camera and laser degradation. (a) If visual features are insufficient for the visual-inertial odometry, the IMU prediction (partially) bypasses the green block to register laser points locally. Correction of velocity drift and biases of the IMU is made with the laser feedback. (b) If environmental structures are insufficient for the scan matching, the visual-inertial odometry output (partially) bypasses the blue block to register laser points on the map. Here, the dashed line segments indicate "bypass".

laser points for the scan matching. The bypassing IMU prediction is subject to drift. The laser feedback compensates for the camera feedback correcting velocity drift and biases of the IMU, only in directions where the camera feedback is unavailable. In other words, the camera feedback has a higher priority, due to the higher frequency making it more suitable. When sufficient visual features are found, the laser feedback is not used.

5.7.2 Case Study of Laser Degradation

As shown in Fig. 5.12(b), if environmental structures are insufficient for the scan matching to refine motion estimates, the visual-inertial odometry output fully or partially bypasses the blue block to register laser points on the map. If well-conditioned directions exist in the scan matching problem, the laser feedback contains refined motion estimates in those directions. Otherwise, the laser feedback becomes empty.

5.7.3 Case Study of Camera and Laser Degradation

Finally, let us discuss a complex scenario where both the camera and the laser are degraded. I will use the example in Fig. 5.13 to illustrate this scenario. A vertical bar with six rows represents a 6-DOF pose where each row is a DOF, corresponding to an eigenvector in (5.19). In this example, both the visual-inertial odometry and the scan matching update 3-DOF motion, leaving the motion unchanged in the other 3-DOF. Starting with the IMU prediction on the left where all six rows are orange, the visual-inertial odometry updates in 3-DOF where the rows change to green, then the scan matching updates in 3-DOF further where the rows turn blue. The camera and the laser feedback contains updates from each module on the green and the blue rows, respectively (white means empty). The feedback is combined upon receiving by the IMU prediction module as the vertical bar on the left. The camera feedback has a higher priority than the laser feedback (discussed in Section 5.7.1). During the combination, the blue rows are only filled in if the green rows are not present.

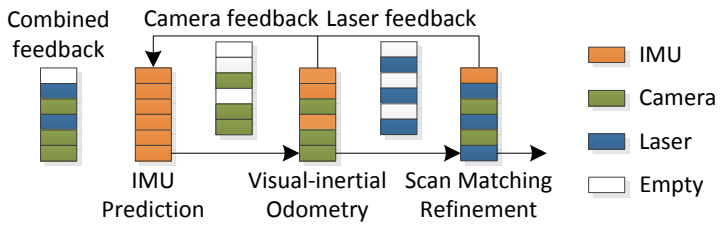


Figure 5.13: An example where both the camera and the laser are degraded. A vertical bar represents a 6-DOF pose and each row is a DOF. Starting with the IMU prediction on the left where all six rows are orange, the visual-inertial odometry updates in 3-DOF where the rows become green, then the scan matching updates in another 3-DOF where the rows turn blue. The camera and the laser feedback is combined as the vertical bar on the left. The camera feedback has a higher priority – blue rows from the laser feedback are only filled in if green rows from the camera feedback are not present.

In reality, however, the visual-inertial odometry and the scan matching execute at different frequencies and have each own degraded directions. I take the poses from the scan matching output and use IMU messages to interpolate in between the poses. This way, I create an incremental motion that is time aligned with the visual-inertial odometry output. Let $\boldsymbol{\theta}^c_{c-1}$ and $\boldsymbol{t}^c_{c-1}$ be the 6-DOF motion estimated by the visual-inertial odometry between frames $c-1$ and c, where $\boldsymbol{\theta}^c_{c-1} \in \mathfrak{so}(3)$ and $\boldsymbol{t}^c_{c-1} \in \mathbb{R}^3$. Let $\boldsymbol{\theta}'^c_{c-1}$ and $\boldsymbol{t}'^c_{c-1}$ be the corresponding terms estimated by the scan matching after time interpolation. Consider $\mathbf{V}_V$ and $\bar{\mathbf{V}}_V$ to be the matrices defined in (5.19) containing eigenvectors from the visual-inertial odometry module, $\bar{\mathbf{V}}_V$ represents well-conditioned directions, and $\mathbf{V}_V - \bar{\mathbf{V}}_V$ represents degraded directions. Let $\mathbf{V}_S$ and $\bar{\mathbf{V}}_S$ be the same matrices from the scan matching module. The following equation calculates the combined feedback, $\boldsymbol{f}_C$,

$$\boldsymbol{f}_C = \boldsymbol{f}_V + \mathbf{V}_V^{-1}(\mathbf{V}_V - \bar{\mathbf{V}}_V)\boldsymbol{f}_S, \tag{5.23}$$

where $\boldsymbol{f}_V$ and $\boldsymbol{f}_S$ represent the camera and the laser feedback,

$$\boldsymbol{f}_V = \mathbf{V}_V^{-1}\bar{\mathbf{V}}_V[(\boldsymbol{\theta}^c_{c-1})^T,\ (\boldsymbol{t}^c_{c-1})^T]^T, \tag{5.24}$$

$$\boldsymbol{f}_S = \mathbf{V}_S^{-1}\bar{\mathbf{V}}_S[(\boldsymbol{\theta}'^c_{c-1})^T,\ (\boldsymbol{t}'^c_{c-1})^T]^T. \tag{5.25}$$

Note that $\boldsymbol{f}_C$ only contains solved motion in a subspace of the state space. I take the motion from the IMU prediction, namely $\hat{\boldsymbol{\theta}}^c_{c-1}$ and $\hat{\boldsymbol{t}}^c_{c-1}$, and project it to the nullspace of $\boldsymbol{f}_C$,

$$\boldsymbol{f}_I = \mathbf{V}_V^{-1}(\mathbf{V}_V - \bar{\mathbf{V}}_V)\mathbf{V}_S^{-1}(\mathbf{V}_S - \bar{\mathbf{V}}_S)[(\hat{\boldsymbol{\theta}}^c_{c-1})^T,\ (\hat{\boldsymbol{t}}^c_{c-1})^T]^T. \tag{5.26}$$

Next, I use $\tilde{\boldsymbol{\theta}}^c_{c-1}(\boldsymbol{b}_{\boldsymbol{\omega}}(t))$ and $\tilde{\boldsymbol{t}}^c_{c-1}(\boldsymbol{b}_{\boldsymbol{\omega}}(t), \boldsymbol{b}_{\boldsymbol{a}}(t))$ to denote the IMU predicted motion formulated as functions of $\boldsymbol{b}_{\boldsymbol{\omega}}(t)$ and $\boldsymbol{b}_{\boldsymbol{a}}(t))$, through integration of (5.3) and (5.4). The orientation $\tilde{\boldsymbol{\theta}}^c_{c-1}(\boldsymbol{b}_{\boldsymbol{\omega}}(t))$ is only relevant to $\boldsymbol{b}_{\boldsymbol{\omega}}(t)$, but the translation $\tilde{\boldsymbol{t}}^c_{c-1}(\boldsymbol{b}_{\boldsymbol{\omega}}(t), \boldsymbol{b}_{\boldsymbol{a}}(t))$ is dependent on both $\boldsymbol{b}_{\boldsymbol{\omega}}(t)$ and $\boldsymbol{b}_{\boldsymbol{a}}(t))$. The biases can be calculated by solving the following equation,

$$[(\tilde{\boldsymbol{\theta}}^c_{c-1}(\boldsymbol{b}_{\boldsymbol{\omega}}(t)))^T,\ (\tilde{\boldsymbol{t}}^c_{c-1}(\boldsymbol{b}_{\boldsymbol{\omega}}(t), \boldsymbol{b}_{\boldsymbol{a}}(t)))^T]^T = \boldsymbol{f}_C + \boldsymbol{f}_I. \tag{5.27}$$

When the system functions normally, $\boldsymbol{f}_C$ spans the state space, and $\mathbf{V}_V - \bar{\mathbf{V}}_V$ and $\mathbf{V}_S - \bar{\mathbf{V}}_S$ in (5.26) are zero matrices. Correspondingly, $\boldsymbol{b}_{\boldsymbol{\omega}}(t)$ and $\boldsymbol{b}_{\boldsymbol{a}}(t))$ are calculated from $\boldsymbol{f}_C$. In a

degraded case, the IMU predicted motion, $\hat{\boldsymbol{\theta}}^c_{c-1}$ and $\hat{\boldsymbol{t}}^c_{c-1}$, is used in directions where the motion is unsolvable (e.g. white row of the combined feedback in Fig. 5.13). The result is that the previously calculated biases are kept in these directions.

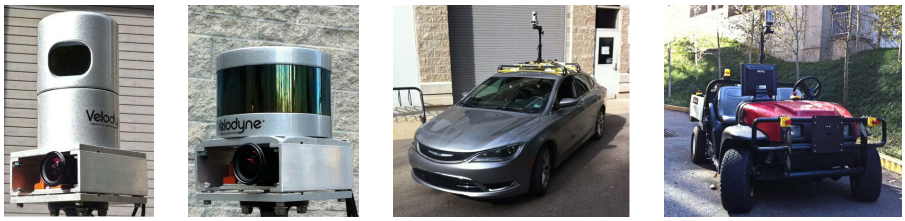
5.8 Experiments

5.8.1 Tests with Velodyne Scanners

The odometry and mapping software system is first validated on two sensor suites. In Fig. 5.14(a), a Velodyne HDL-32E laser scanner is attached to a UI-1220SE monochrome camera and an Xsens MTi-30 IMU. The laser scanner has 360° horizontal FOV, 40° vertical FOV, and receives 0.7 million points/second at 5Hz spinning rate. The camera is configured at the resolution of 752×480 pixels, 76° horizontal FOV, and 50Hz frame rate. The IMU frequency is set at 200Hz. In Fig. 5.14(b), a Velodyne VLP-16 laser scanner is attached to the same camera and IMU. This laser scanner has 360° horizontal FOV, 30° vertical FOV, and receives 0.3 million points/second at 5Hz spinning rate. Both sensor suites are attached to the vehicles in Fig. 5.14(c) and Fig. 5.14(d) for data collection, which are driven on streets and in off-road terrains.

For both sensor suites, maximally 300 Harris corners are tracked using the Kanade Lucas Tomasi (KLT) method [61]. To evenly distribute the visual features, an image is separated into 5×6 identical subregions, each subregion provides up to 10 features. When a feature loses tracking, a new feature is generated to maintain the feature number in each subregion.

The software runs on a laptop computer with a 2.6GHz i7 quad-core processor (2 threads on each core and 8 threads overall) and an integrated GPU, in a Linux system running Robot Operating System (ROS) [79]. I implement two versions of the software with visual feature tracking running on GPU and CPU, respectively. The processing time is shown in Table 5.2. The time used by the visual-inertial odometry (middle module in Fig. 5.2(c)) does not vary much w.r.t. the environment or sensor configuration. For the GPU version, it consumes about 25% of a CPU thread executing at 50Hz. For the CPU version, it takes about 75% of a thread. The sensor suite in Fig. 5.14(a) results in slightly more processing time than the one in Fig. 5.14(b). This



(a) (b) (c) (d)

Figure 5.14: Sensor suites and vehicles used in experiments. (a) is a Velodyne HDL-32E laser scanner attached with a uEye UI-1220SE monochrome camera and an Xsens MTi-30 IMU. (b) is a Velodyne VLP-16 laser scanner attached with the same camera and IMU. (c) is a passenger vehicle for street driving. (d) is a utility vehicle for off-road driving. Each sensor suite in (a) and (b) is attached to both vehicles in (c) and (d) for experiment validation.

Table 5.2: Average CPU processing time using the sensor suites in Fig. 5.14(a) and Fig. 5.14(b).

Environment	Senor suite	Visual-inertial odometry (time per image frame)		Scan Matching (time per laser scan)
		GPU Tracking	CPU Tracking	
Structured	Fig. 5.14(a)	4.8ms	14.3ms	148ms
	Fig. 5.14(b)	4.2ms	12.9ms	103ms
Vegetated	Fig. 5.14(a)	5.5ms	15.2ms	267ms
	Fig. 5.14(b)	5.1ms	14.7ms	191ms

is because the scanner receives more points and the program needs more time to maintain the depthmap and associate depth to the visual features.

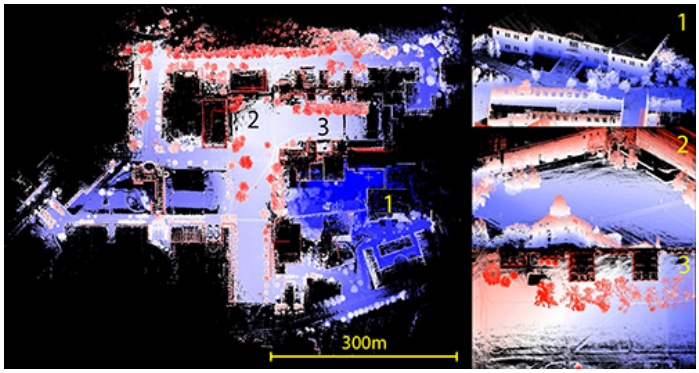
The scan matching (rightmost module in Fig. 5.2(c)) consumes more processing time which also varies w.r.t. the environment and sensor configuration. With the sensor suite in Fig. 5.14(a), the scan matching takes about 75% of a thread executing at 5Hz if operated in structured environments. In vegetated environments, however, more points are registered on the map and the program typically consumes about 135% of a thread. With the sensor suite in Fig. 5.14(b), the scanner receives less number of points. The scan matching uses about 50-95% of a thread depending on the environment. The time used by the IMU prediction (leftmost module in Fig. 5.2(c)) is neglectable compared to the other two modules.

Accuracy Tests

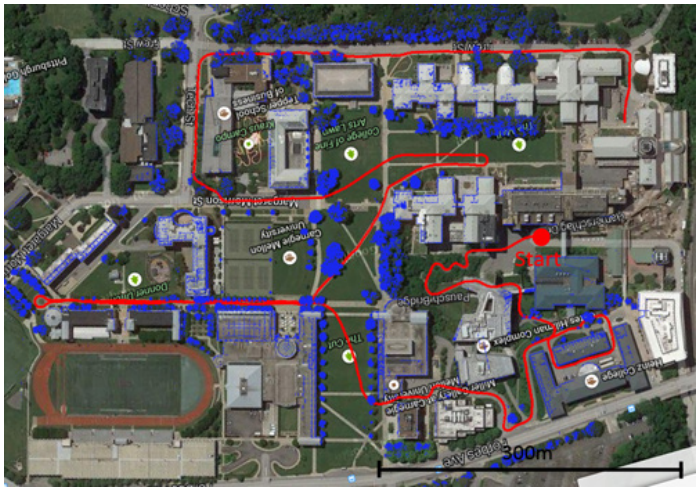
I will first conduct tests to evaluate accuracy of the proposed system. In these tests, the sensor suite in Fig. 5.14(a) is used. I first mount the sensors on the vehicle in Fig. 5.14(d) driving around the university campus. After 2.7km of driving within 16 minutes, a campus map is built (shown in Fig. 5.15(a)). The average speed over the test is 2.8m/s. In addition to the overall map, three close views are present on the right for readers to inspect the local registration accuracy. The corresponding locations are labeled with numbers 1-3 on the map.

To evaluate motion estimation drift over the test, I align the estimated trajectory and registered laser points on a satellite image in Fig. 5.15(b). Here, laser points on the ground are manually removed. By matching the trajectory with streets on the satellite image, I am able to determine an upper bound of the horizontal error to be < 1.0m. By comparing buildings on the same floor, I further determine the vertical error to be < 2.0m. This gives an overall relative position drift at the end to be $< 0.09\%$ of the distance traveled. I am aware that precision cannot be guaranteed for the measurements, hence only an upper bound of the drift is calculated.

Further, I conduct a more comprehensive test with the same sensors mounted on the vehicle in Fig. 5.14(c). The vehicle is driven on structured roads for 9.3km of travel. As shown in Fig. 5.16, the path goes through vegetated environments, bridges, hilly terrains, and streets with heavy traffic, and finally returns to the starting position. The elevation changes over 70m along the path. Except waiting for traffic lights, the vehicle speed is between 9-18m/s during the test. On the left side of Fig. 5.16, the complete map color coded by elevation is shown. On the right, a few close views are present with corresponding locations labeled with numbers 1-5 on the map.



(a)



(b)

Figure 5.15: Accuracy test 1. The sensor suite in Fig. 5.14(a) is mounted on the vehicle in Fig. 5.14(d) to map the university campus. The overall path is 2.7km in length, finished in 16 minutes with an average driving speed of 2.8m/s. (a) shows the map built and three close views labeled with numbers 1-3. The corresponding locations on the map are marked with the same numbers. (b) shows the vehicle trajectory (red) and registered laser points (blue) overlayed on a satellite image. Laser points on the ground are manually removed. The relative position drift at the end is $< 0.09\%$ of the distance traveled.

In particular, close view 1 shows the starting and the ending positions. Carefully examining the figure, one sees that a building is registered into two. This is because of motion estimation drift over the path, while one is registered when the vehicle leaves from the start and the other when the vehicle returns at the end. I measure the gap to be < 20m, which results in a relative position error at the end to be $< 0.22\%$ of the distance traveled. More details are shown in close views 2-5 with corresponding images logged by the camera.

Additionally, I examine how each module in the system contributes to the overall accuracy. As shown in Fig. 5.17, I first plot output of the visual-inertial odometry as the green dash-dot curve. This uses the left two modules in Fig. 5.2(c). Next, I directly forward the IMU prediction to the scan matching module, bypassing the visual-inertial odometry. This configuration uses the

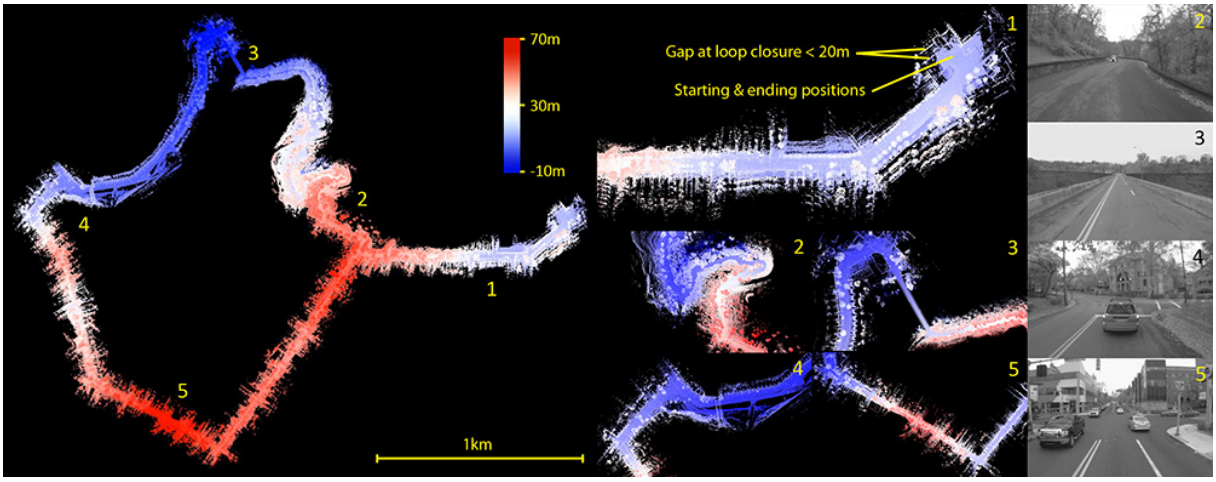


Figure 5.16: Accuracy test 2. The sensor suite in Fig. 5.14(a) is mounted on the vehicle in Fig. 5.14(c) for 9.3km of street driving. The path goes through vegetated environments, bridges, hilly terrains, and roads with heavy traffic. The elevation changes over 70m. Except waiting for traffic lights, the vehicle is driven at 9-18m/s. On the left, the complete map color coded by elevation is shown. On the right, a few close views are shown with locations labeled with numbers 1-5 on the map. In close view 1, the starting and the ending positions are present. Because of drift, a building is registered into two, one during the vehicle leaves from the start and the other during the vehicle returns at the end. I manually measure the gap to be < 20m, resulting in a relative position error at the end to be $< 0.22\%$ of the distance traveled. Close views 2-5 show more details with corresponding images logged by the camera.

leftmost and the rightmost modules in Fig. 5.2(c). The result is drawn as the blue dash curve. Finally, I plot output of the complete pipeline as the red solid curve, with the least amount of drift. The position errors of the first two configurations are about four and two times larger.

One can consider the green dash-dot curve and the blue dash curve as the expected system performance when encountering individual sensor degradation: if scan matching is degraded, the system reduces to a mode indicated by the green dash-dot curve; if vision is degraded, the system reduces to that indicated by the blue dash curve. Further, I reconfigure the system to incorporate all constraints in one large optimization problem as in Fig. 5.2(b). The system takes the IMU prediction as the initial guess and runs at the laser frequency (5Hz). The system produces a trajectory as the black dot curve. The resulting accuracy is only little better in comparison to the blue dash curve which uses the IMU directly coupled with the laser, passing the visual-inertial odometry. The result indicates that the high-frequency advantage of the camera is unexplored if solving the problem with all constraints stacked together.

I would like to further understand how the modules corporate in the system. To this end, I compare accuracy of the system running at the original 1x speed and an accelerated 2x speed. When running at 2x speed, I skip every other data frame for all three sensors, resulting in much more aggressive motion through the test. The results are listed in Table 5.3. At each speed, I evaluate three configurations. As one can see, when running at 2x speed, the accuracy of the visual-inertial odometry and the IMU + scan matching configurations reduce significantly, by 0.54% and 0.38% of the distance traveled in comparison to the accuracy at 1x speed. However,

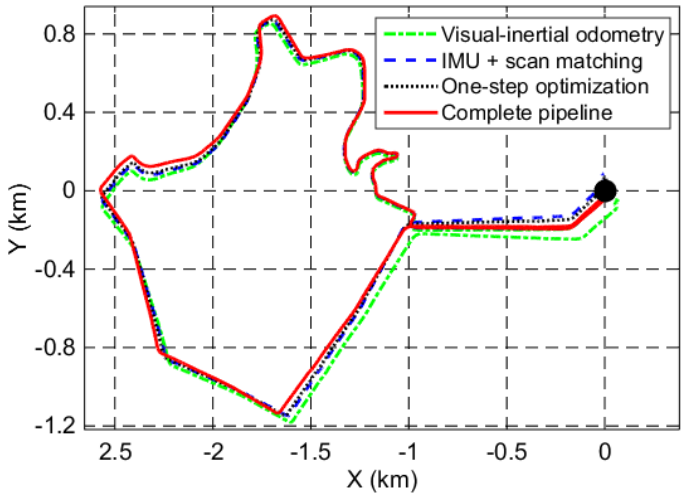


Figure 5.17: Estimated trajectories in accuracy test 2. The trajectories start with the black dot. Four system configurations are compared in the test. The green dash-dot curve is from the visual-inertial odometry module (using the left two modules in Fig. 5.2(c)). The blue dash curve is from the scan matching module with the IMU prediction directly taken as input (leftmost and rightmost modules in Fig. 5.2(c)). The black dot curve has the system reconfigured to solve one large optimization problem incorporating all constraints, as in Fig. 5.2(b). The red solid curve is from the proposed data processing pipeline.

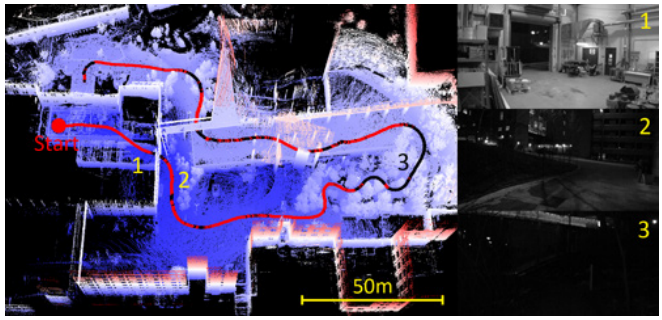
the complete pipeline reduces accuracy very little, only by 0.04%. The results indicate that the camera and the laser compensate for each other keeping the overall accuracy. This is especially true when the motion is aggressive.

Robustness Tests

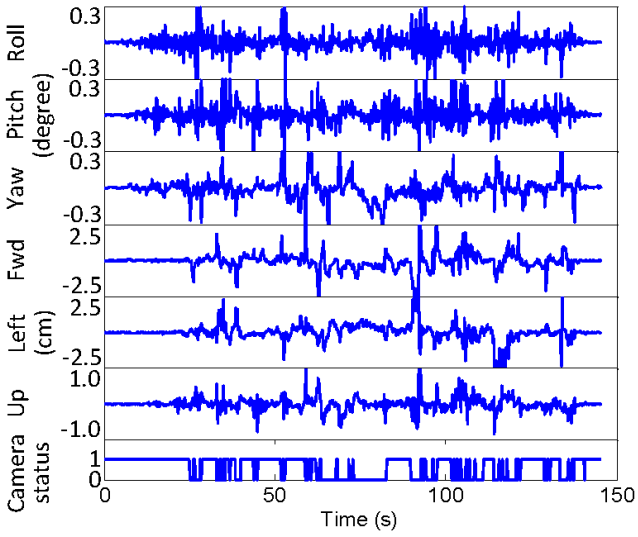
I will further inspect the system robustness w.r.t. sensor failures. Specifically, I carry out test cases to produce vision degradation due to low-light and scan matching degradation due to lack of structures. Here, it is worth to mention that the same mechanism in the system that ensures the robustness w.r.t. environmental degradation also warrants the robustness w.r.t. and aggressive motion. This is because both affect the system in a similar way. The problem caused by environmental degradation is due to the fact that the environment does not contain sufficient information, resulting in an ill-conditioned problem as discussed in Section 5.7. The problem caused by aggressive motion is because of sparsity of the data in which insufficient information

Table 5.3: Relative position errors as percentages of the distance traveled in accuracy test 2. The errors at 1x speed correspond to the trajectories in Fig. 5.17.

Configuration	1x speed	2x speed
Visual-inertial odometry	0.93%	1.47%
IMU + scan matching	0.51%	0.89%
Complete pipeline	**0.22%**	**0.26%**



(a)

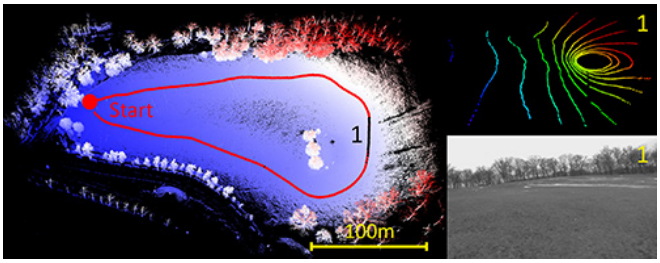


(b)

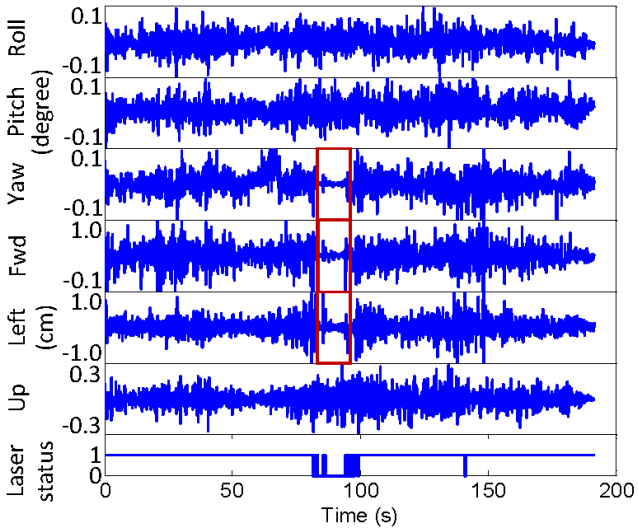
Figure 5.18: Robustness test 1. The sensor suite in Fig. 5.14(b) is attached to the vehicle in Fig. 5.14(d) driven from indoor to outdoor. The test is conducted at night. Frequently, the camera cannot capture enough visual features and the visual-inertial odometry module is bypassed. In (a), the estimated trajectory overlayed on the map built is shown. The red segments indicate vision is functional and the black segments indicate degradation. Also, three images logged by the camera from locations 1-3 labeled on the map are shown. Location 1 is indoor and locations 2-3 are outdoor. In (b), pose corrections applied by the scan matching to refine motion estimates are show. On the bottom row, the camera status being one indicates functioning. When the camera status is zero, corrections on the top six rows become larger because the IMU prediction produces more drift than the visual-inertial odometry.

is captured from the environment, resulting in the same ill-conditioned problem. Both problems are handled using the method introduced in Section 5.7.

The experiments use the sensor suite in Fig. 5.14(b) attached to the vehicle in Fig. 5.14(d). First, the vehicle is driven at night where vision degrades. When insufficient number of visual features are tracked, the visual-inertial odometry module is bypassed, and the IMU prediction is directly sent to the scan matching module. As shown in Fig. 5.18(a), the red and the black segments on the trajectory respectively indicate vision is functional and degraded. In Fig. 5.18(b),



(a)

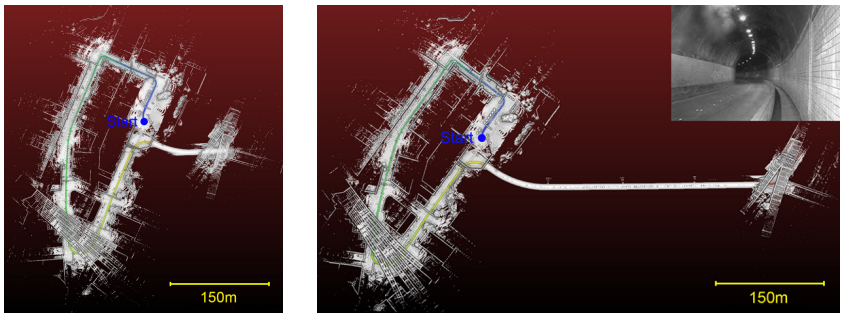


(b)

Figure 5.19: Robustness test 2. The sensor suite in Fig. 5.14(b) is attached to the vehicle in Fig. 5.14(d) driven in an off-road terrain. In (a), when the vehicle reaches the rightmost side of the path, only the flat ground is seen, causing the scan matching to partially degrade. The corresponding trajectory is drawn in black. Here, it determines the scan matching is able to refine 3-DOF out of the 6-DOF motion, which are roll, pitch, and elevation. The other 3-DOF are unsolvable due to the planar scene, where the pose is directly taken from the visual-inertial odometry. In addition, a laser scan and an image logged from location 1 labeled on the map are shown. In (b), pose corrections applied by the scan matching are shown. On the last row, the laser status being one indicates functioning. When the laser status is zero, pose corrections in degraded directions (in the red boxes) become much smaller.

pose corrections applied by the scan matching for motion estimation refinement are shown. On the bottom row. the camera status being zero indicates degradation. Correspondingly, pose corrections on the top six rows become larger because the IMU prediction is less precise in comparison to the visual-inertial odometry.

Next, I bring the vehicle to an open area where scan matching degrades due to the planar environment. As shown in Fig. 5.19(a), when the vehicle researches the rightmost side of the path (black segment), only the flat ground is seen by the laser. The system determines the scan matching is able to refine 3-DOF out of the 6-DOF motion using the method introduced in Section 5.7. Specifically, roll, pitch, and elevation are well-conditioned, but yaw, forward, left



(a) (b)

Figure 5.20: Robustness test 3. Succeeding test of laser degradation in a smooth tunnel. The tunnel is 380m in length, with a 45° curve close to its left end. In this case, if the scan matching module updates in all 6-DOF, the resulting map is problematic as shown in (a). The tunnel is squeezed because of estimation failure in the down-track direction (translation parallel to the tunnel). In the proposed processing pipeline, the scan matching module is partially bypassed in the degraded direction to successfully generate the map in (b).

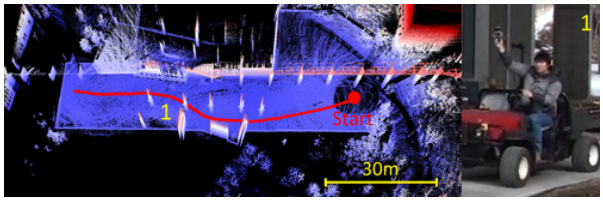
are unsolvable due to the planar scene. The visual-inertial odometry output is used directly in the degraded directions. In Fig. 5.19(b), pose corrections applied by the scan matching are shown. On the bottom row, the laser status being zero indicates partial degradation. Correspondingly, corrections in the degraded directions (labeled in the red boxes) are much smaller because the corrections are only applied in well-conditioned directions of the problem (rightmost module in Fig. 5.2(c)) as determined by the method in Section 5.7.

Further, I show in Fig. 5.20 succeeding result of laser degradation. The test is passing through a smooth tunnel that is 380m long. The system encounters laser degradation and the scan matching module is partially bypassed in order to produce the result in Fig. 5.20(a). However, if I force the scan matching module to solve in 6-DOF, the result is shown in Fig. 5.20(b) where the tunnel is squeezed due to state estimation failure in the direction parallel to the tunnel.

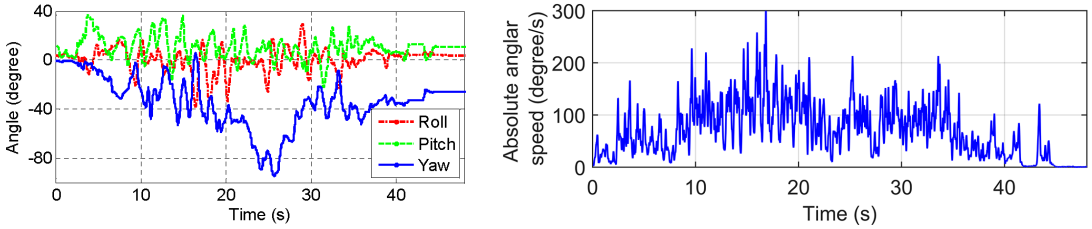
Aggressive Motion Tests

In this section, I will evaluate the system performance w.r.t. high-speed rotation and translation. As discussed, the system uses the method in Section 5.7 to hold the robustness w.r.t. aggressive motion. The tests use the sensor suite in Fig. 5.14(b). First, the sensors are held by a person who drives the vehicle in Fig. 5.14(d). The vehicle carries power supply and a data processing computer. The person oscillates the sensor suite to introduce fast rotation. Next, the sensors are mounted to the vehicle in Fig. 5.14(c) driven along an ”S” shaped path. Results of the two tests are in Fig. 5.21 and Fig. 5.22. In Fig. 5.21-5.22(a), the estimated trajectories are overlayed on the maps built, with photos showing experiment setups. In Fig. 5.21-5.22(b), the estimated orientations are present. In Fig. 5.21-5.22(c), the estimated absolute angular speeds are shown. For the first test, the maximum angular speed exceeds $250^\circ/s$. For the second test, the accumulative rotation is over 330° in eight seconds.

Finally, I mount the sensors on the vehicle in Fig. 5.14(c) and drive along a straight path at a high speed. As shown in Fig. 5.23, the overall path is 701m in length. The blue points are laser



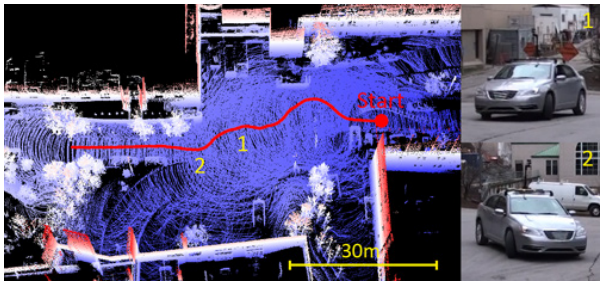
(a)



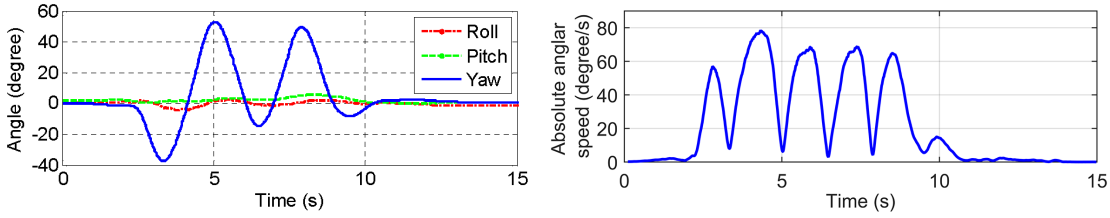
(b) (c)

Figure 5.21: Aggressive motion test 1. The sensor suite in Fig. 5.14(b) is held by a person in one hand who drives the vehicle in Fig. 5.14(d) with the other hand. The person oscillates the sensor suite to introduce fast rotation. (a) shows the estimated trajectory overlayed on the map built and a photo taken from location 1 during the test. (b) shows the estimated orientation. (c) is the estimated absolute angular speed. The maximum angular speed exceeds $250^\circ/s$.

points registered and overlayed on a satellite image. One sees the mapped trees and houses are well aligned with the satellite image. Through this comparison, I believe the horizontal position error is < 1.0m, resulting in a horizontal position drift to be $< 0.15\%$ of the distance traveled.



(a)



(b) (c)

Figure 5.22: Aggressive motion test 2. The sensor suite in Fig. 5.14(b) is mounted to the vehicle in Fig. 5.14(c). The vehicle is driven along an "S" shaped path. The sensor rotates over 330° within eight seconds during the test.

Figure 5.23: Aggressive motion test 3. The sensor suite in Fig. 5.14(b) is mounted to the vehicle in Fig. 5.14(c), driven at a high speed along the red path. The overall path is 701m and the maximum linear speed is 33m/s. The blue points are laser points overlayed on a satellite image. Three mapped houses and a corresponding image taken from location 1 labeled on the satellite image are shown. Meanwhile, the three houses are on the left side of the image.

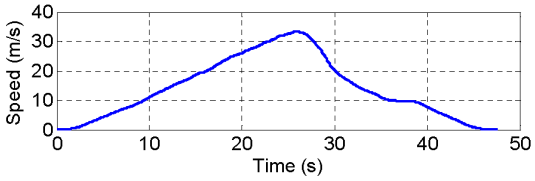


Figure 5.24: Estimated linear speed in aggressive motion test 3. The highest speed reaches 33m/s (119km/h or 74 miles/hour) during the test.

For the vertical drift, however, I do not have a means to evaluate. Three mapped houses in close views on the right side of Fig. 5.23 are shown, and a corresponding image taken from location 1 on the satellite image. The houses are on the left side of the image. In Fig. 5.24, I plot the linear speed. The maximum speed reaches as high as 33m/s (119km/h or 74 miles/hour).

5.8.2 Tests with Custom-built Contour

The odometry and mapping software system is further validated on a custom-built Contour. As shown in Fig. 5.25, the device includes a 2D Hokuyo UTM-30LX-EW laser scanner acquiring 43.2 thousand points/second. The laser scanner is attached to a motor-encoder shaft spinning at 1Hz, functioning as a 3D scanner. The device also includes a wide-angle camera configured at the resolution of 640×512 pixels for motion estimation, an HD color camera at 1600×1200

(a) (b)

Figure 5.25: (a) Front and (b) back views of a custom-built Contour. The device includes a spinning 2D Hokuyo UTM-30LX-EW laser scanner functioning as a 3D scanner, a wide-angle camera at 640×512 pixels for motion estimation, an HD color camera at 1600×1200 pixels for point cloud colorization, and an Xsens MTi-20 IMU. The device is also equipped with an embedded i7 computer for online data processing and a touch-screen monitor.

Table 5.4: Average CPU processing time on Contour in Fig. 5.25.

Environment	Visual-inertial odometry (time per image frame)		Scan Matching (time per laser scan)
	GPU Tracking	CPU Tracking	
Structured	6.4ms	16.7ms	162ms
Vegetated	6.9ms	18.7ms	343ms

pixels for point cloud colorization, and an Xsens MTi-20 IMU. An onboard embedded computer with an 1.8GHz i7 dual-core processor (4 threads overall) runs the data processing software and displays mapping results on a touch-screen monitor in real-time.

I use the same data processing pipeline as in Fig. 5.2(c) except that the scan matching module runs at 1Hz instead of 5Hz. This is due to the fact that the 3D scanner on Contour has a lower spinning rate. Maximally 300 visual features are tracked. The CPU processing time is shown in Table 5.4. Note that the embedded computer in Contour is less powerful than the laptop tested with the Velodyne scanners. When running feature tracking on GPU, the visual-inertial odometry consumes about 35% of a CPU thread executing at 50Hz. If running feature tracking on CPU, however, the processing time is about 85% of a thread. The scan matching takes between 15-35% of a thread executing at 1Hz, depending on the type of environment.

Fig. 5.26 shows results from a 4-floor residential house. Fig. 5.26(a) is a photo of the house. Fig. 5.26(b)-(c) are maps of surrounding of the house, in perspective view and top-down view. Fig. 5.26(d)-(g) are respectively the basement, first floor, second floor, and third floor of the interior. Further, three maps in Fig. 5.27 with the sensor paths inserted are shown. Here, Fig. 5.27(a) is the same area as Fig. 5.26(e). Fig. 5.27(b)-(c) are from a different building. In all tests, the device is moved at a speed around 0.5m/s, resulting in each of the three maps in Fig. 5.27 built in about two minutes. The exterior map in Fig. 5.26(b)-(c) takes about six minutes due to its larger scale and more details to cover. Due to difficulty to acquire ground truth, meanwhile, one can only let readers visually inspect quality of the maps.

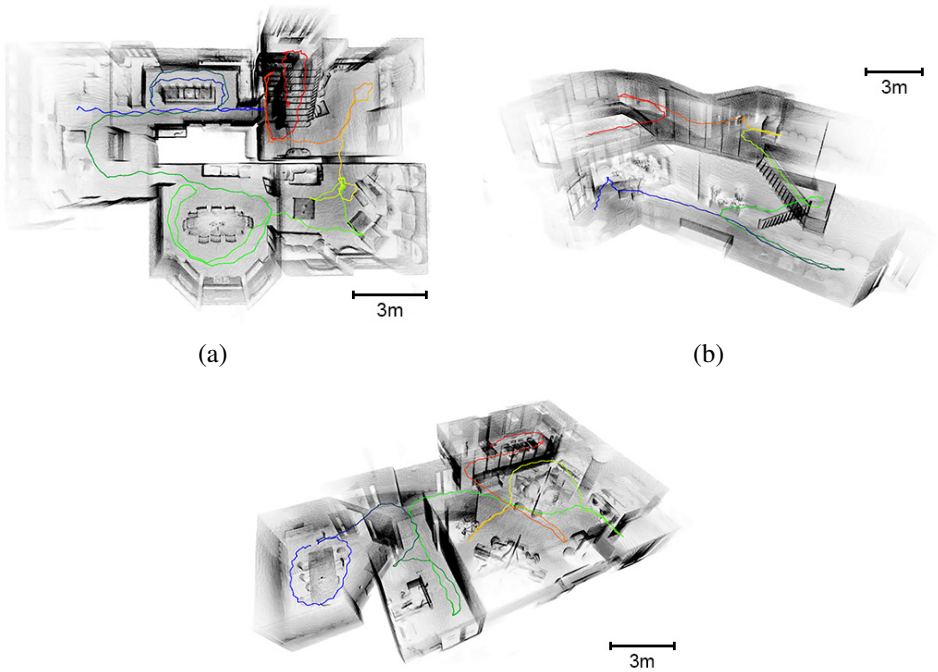
Finally, I evaluate the performance in aggressive motion. Contour is hand-carried by a person who rotates the device fast inside a room while walking and traversing the room. The results are shown in Fig. 5.28. In Fig. 5.28(a), the estimated trajectory overlayed on the map built is present. In Fig. 5.28(b), the estimated orientation is shown. In Fig. 5.28(c), the estimated absolute angular speed is shown. From the results, one can see that the maximum angular speed is up to $370^\circ/s$. During 82 seconds of the test, the accumulated rotation is about $6.8^\circ \times 10^3$. This results in an average angular speed of $83^\circ/s$ over the test.

5.9 Conclusion

The chapter presents a data processing pipeline for ego-motion estimation and mapping. The pipeline couples a 3D laser, a camera, and an IMU, running three modules sequentially to produce real-time ego-motion estimation. The coarse-to-fine data processing generates high-rate estimation and registers low-drift maps over a long distance of travel. Further, the system is

Figure 5.26: (a) A photo and (b)-(g) maps built from a 4-floor residential house using Contour in Fig. 5.25. (b)-(c) are horizontal and top-down views of the surrounding. (d)-(g) are respectively the basement, first floor, second floor, and third floor of the interior. The device is moved at around 0.5m/s over the test.

robust to individual sensor failures. Due to degraded environments or aggressive motion, if the camera or the laser is not fully functional, the corresponding module is bypassed and the rest system is staggered to warrant the overall functionality. The system is validated through a large

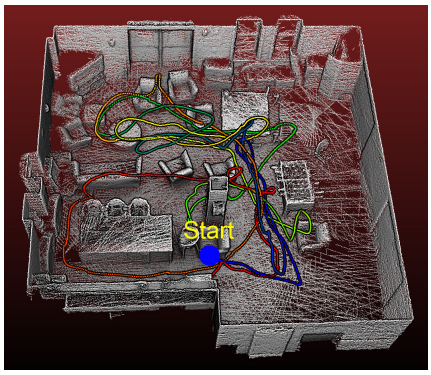


(a) (b)

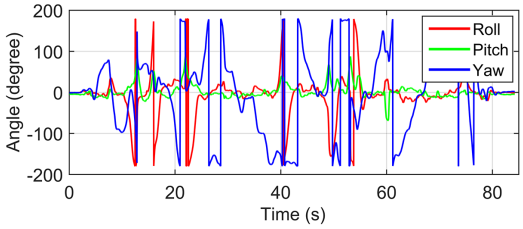
(c)

Figure 5.27: Three maps built by Contour in Fig. 5.25 with sensor paths inserted. (a) is the same area as Fig. 5.26(e). (b)-(c) are from a different building. The device is moved at around 0.5m/s. Each of the three tests lasts about 2 minutes.

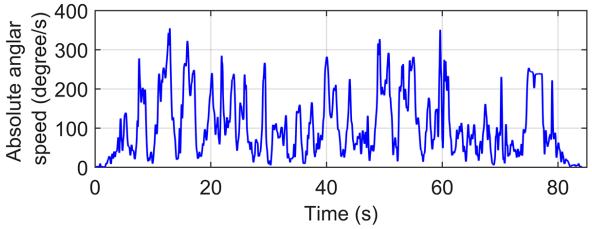
number of experiments. In particular, tests are conducted to evaluate the accuracy and robustness over several kilometers of travel, in complex road conditions, with dramatic lighting changes and structural degradation, and with high-rate motion in rotation and translation. Results indicate that the system can conquer all challenging scenarios, producing position drift around 0.2% of the distance traveled with robustness w.r.t running, jumping motion and highway speed driving.



(a)



(b)



(c)

Figure 5.28: Aggressive motion test with Contour in Fig. 5.25. The device is hand-carried and rotated fast inside a room. (a) shows the estimated trajectory overlayed on the map built. (b) shows the estimated orientation. (c) is the estimated absolute angular speed. The maximum angular speed is as high as $370^\circ/s$. During 82 seconds of the test, the accumulated rotation reaches $6.8^\circ \times 10^3$, leading to an average angular speed of $83^\circ/s$.

Chapter 6

Further Comparison of Three Methods

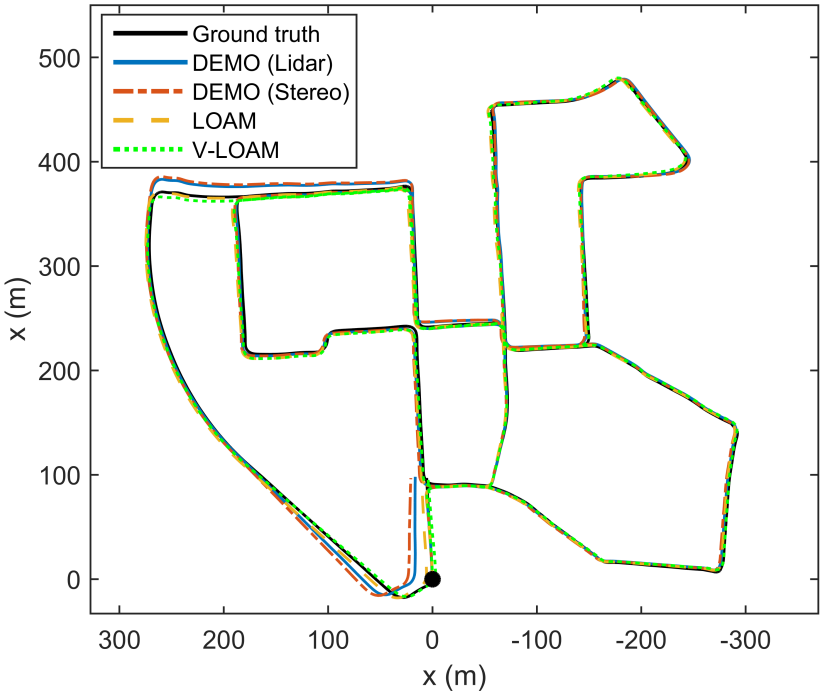
In this chapter, I will further compare the three methods using datasets from the KITTI odometry benchmark [33, 36]. The datasets are logged with sensors mounted on top of a passenger vehicle in road driving scenarios. The vehicle is equipped with color stereo cameras, monochrome stereo cameras, a Velodyne HDL-64E laser scanner, and a high accuracy GPS/INS for ground truth. The image resolution is around 1230×370 pixels, with 81° horizontal field of view. Both image and lidar data are logged at 10Hz. The datasets contain 22 tests in total. The first 11 datasets are training data with GPS/INS ground truth provided, while the last 11 datasets are for evaluation. The overall distance in the datasets is 39km with the maximum driving speed of 85km/h (23.6m/s). The data covers mainly three types of environments: "urban" with buildings around, "country" on small roads with vegetations in the scene, and "highway" where roads are wide and the vehicle speed is fast.

Results of the three methods are shown in Table 6.1. This is using the 11 evaluation datasets where GPS/INS ground truth is not publicly available. By uploading results to the benchmark server, the accuracy and ranking are automatically calculated. Here, the accuracy is measured by averaging relative translation and rotation errors using data segments at 100m, 200m, ..., 800m lengthes, based on 3D coordinates. Specifically, the Depth Enhanced Monocular Odometry (DEMO) method produces the lowest accuracy among the three, while the Vision-lidar Odometry and Mapping (V-LOAM) method is the most accurate. All three methods use lidar data. In addition, DEMO and V-LOAM also use images from the left monochrome camera. Note that the KITTI odometry benchmark does not provide inertial data, the IMU prediction module is removed from the processing pipeline present in Chapter 5.

Table 6.1: Comparison of three methods using the KITTI odometry benchmark. Both translation and rotation errors are averaged errors as fractions of the distance traveled, using data segments in 100m, 200m,..., 800m lengthes based on 3D coordinates.

Chapter	Acronym	Method	Error	
			Translation	Rotation
3	DEMO	Depth Enhanced Monocular Odometry	1.14%	0.0049°/m
4	LOAM	Lidar Odometry and Mapping	0.70%	0.0017°/m
5	V-LOAM	Vision-lidar Odometry and Mapping	0.68%	0.0016°/m

Further, I compare the estimated trajectories from three representative datasets in Fig. 6.1-Fig. 6.3. The datasets are collected from urban, country, and highway scenes, respectively. In these figures, DEMO (Lidar) refers to the standard version of the method present in Chapter 3, which associates depth information to visual features from lidar data. DEMO (Stereo) is a variant version which obtains depth information from stereo imagery, using the method in [34]. Range information from a lidar is replaced by a stereo camera during ego-motion estimation. Since method [34] runs at 5Hz using a single CPU thread, DEMO (Stereo) is executed at $0.5\times$ of real-time speed (image data is collected at 10Hz on the KITTI odometry benchmark). All the other methods are executed at real-time speed. One can see that DEMO (Stereo) produces accuracy slightly less than DEMO (Lidar). V-LOAM generates accuracy slightly higher than LOAM. Worth to mention, by feeding the results of DEMO (Lidar) and DEMO (Stereo) to the scan matching module in Chapter 5, it generates very similar results to V-LOAM (see Table 6.2



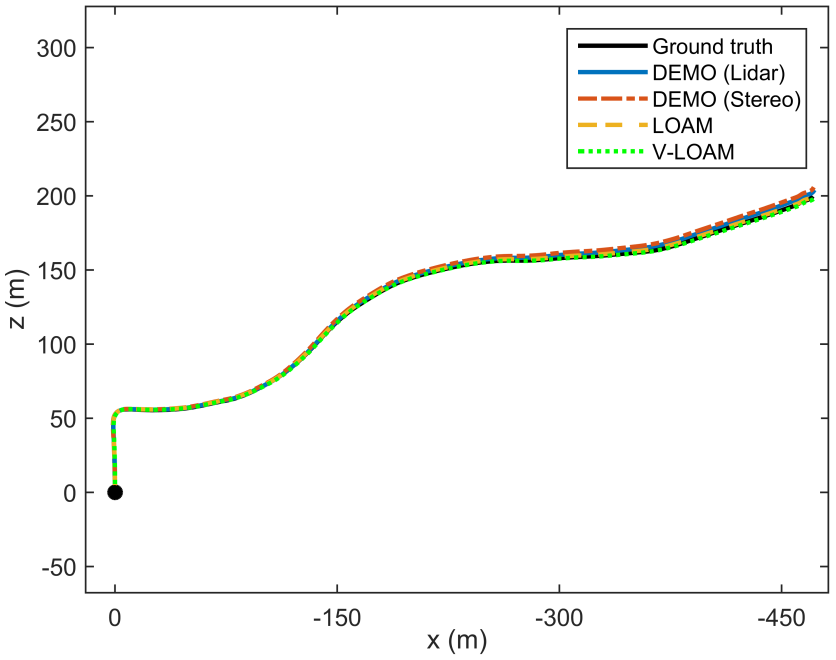
(a)

(b)

Figure 6.1: (a) Estimated trajectories using the KITTI odometry benchmark from an urban scene. Relative position errors are, DEMO (Lidar): %1.05, DEMO (Stereo): %1.09, LOAM: %0.78, and V-LOAM: %0.71. (b) An image from the dataset.

for accuracy comparison), proving vision based estimation can be improved by scan matching.

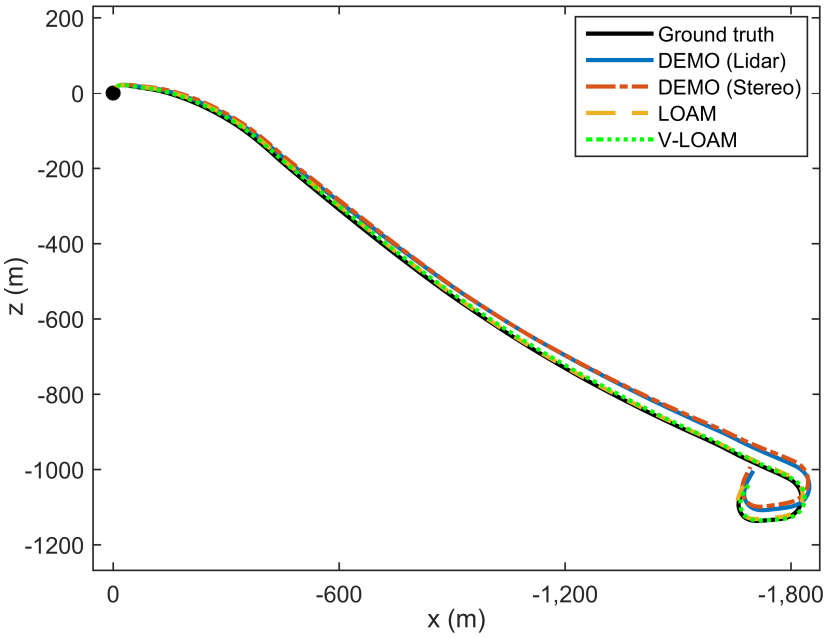
Finally, I investigate processing time w.r.t. accuracy using V-LOAM as an example. Two parameters in the processing pipeline are chosen – visual feature number and scan matching resolution. These two parameters are the most sensitive to determine vision and lidar data processing time, respectively. Results are shown in Table 6.3 andTable 6.4. Comparing the two tables, one can see that when visual features are fewer, the ego-motion estimation accuracy does not drop significantly. However, when the feature number is less than a threshold, the estimation fails. In comparison, when the scan matching resolution decreases, the accuracy drops considerably. Even when the resolution reduces to 10%, the estimation still succeeds (with low accuracy).



(a)

(b)

Figure 6.2: (a) Estimated trajectories using the KITTI odometry benchmark from a country scene. Relative position errors are, DEMO (Lidar): %0.99, DEMO (Stereo): %1.02, LOAM: %0.86, and V-LOAM: %0.72. (b) An image from the dataset.



(a)

(b)

Figure 6.3: (a) Estimated trajectories using the KITTI odometry benchmark from a highway scene. Relative position errors are, DEMO (Lidar): %1.87, DEMO (Stereo): %1.93, LOAM: %1.43, and V-LOAM: %1.41. (b) An image from the dataset.

Table 6.2: Comparsion of relative position errors using the KITTI odometry benchmark from urban, country, and highway scenes corresponding to Fig. 6.1-Fig. 6.3, respectively.

Figure	Scene	DEMO (Lidar)	DEMO (Stereo)	LOAM	V-LOAM	DEMO (Lidar) + Scan Matching	DEMO (Stereo) + Scan Matching
Fig. 6.1	Urban	%1.05	%1.09	%0.78	%0.71	%0.68	%0.69
Fig. 6.2	Country	%0.99	%1.02	%0.86	%0.72	%0.72	%0.73
Fig. 6.3	Highway	%1.87	%1.93	%1.43	%1.41	%1.40	%1.41

Table 6.3: Comparsion of processing time and relative position errors w.r.t. visual feature number using the KITTI odometry benchmark combining datasets in Fig. 6.1-Fig. 6.3.

Visual Feature Number		Processing Time		Relative Position Error	
Feature points	Relative	Time	Relative	Error	Relative
2400	%100	163ms	100%	0.94%	100%
1200	%50	72ms	44%	0.99%	105%
600	%25	31ms	19%	1.07%	114%
240	%10	12ms	7%	Failed	N/A

Table 6.4: Comparsion of processing time and relative position errors w.r.t. scan matching resolution using the KITTI odometry benchmark combining datasets in Fig. 6.1-Fig. 6.3.

Scan Matching Resolution		Processing Time		Relative Position Error	
Edge/Planar Points	Relative	Time	Relative	Error	Relative
0.1/0.2m	%100	217ms	100%	0.94%	100%
0.2/0.4m	%50	67ms	31%	1.17%	124%
0.4/0.8m	%25	29ms	13%	1.52%	162%
1.0/2.0m	%10	13ms	6%	3.68%	391%

Chapter 7

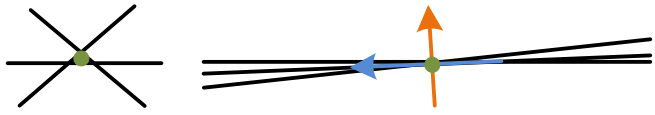
Dealing with Degeneracy

Positioning and mapping can be conducted accurately by state-of-the-art state estimation methods. However, reliability of these methods is largely based on avoiding degeneracy that can arise from cases such as scarcity of texture features for vision sensors and lack of geometrical structures for range sensors. Since the problems are inevitably solved in uncontrived environments where sensors cannot function with their highest quality, it is important for the estimation methods to be robust to degeneracy. This chapter proposes an online method to mitigate for degeneracy in optimization-based problems, through analysis of geometric structure of the problem constraints. The method determines and separates degenerate directions in the state space, and only partially solves the problem in well-conditioned directions. I will demonstrate utility of this method with data from a camera and lidar sensor pack to estimate 6-DOF ego-motion. Experimental results show that the system is able to improve estimation in environmentally degenerate cases, resulting in enhanced robustness for online positioning and mapping.

7.1 Introduction

Recent developments on state estimation methods have shown promising results in positioning and mapping. It is now common to use vision and/or range sensing to recover sensor motion with low drift over long distances. However, the problem of degeneracy remains less studied and prevents navigation systems from reliable functioning, e.g. a camera in a feature-poor scene or a lidar in a planar environment, causing estimation failure. Common ways to deal with degeneracy are 1) switching to a different method, and 2) adding in artificial constraints such as from a constant velocity model during the course of state estimation. Neither approach is satisfactory as the first approach requires a spare method being available, and the second approach brings in unnecessary errors even when the problem itself is well-conditioned and solvable.

The approach is motivated by the observation that even if a set of data used by an estimation method is locally degenerate, very often, some of the constraints provided can be used to solve in a subspace of the original problem. In other words, additional constraints or assumptions do not need to apply in well-conditioned directions, but degenerate directions only, to maximally reduce their negative effect. The above discussion leads to two extreme cases, 1) the problem is well-conditioned entirely and can be solved as is, and 2) the problem is degenerate com-



(a) (b)

Figure 7.1: Intuition of the proposed method in determining state estimation degeneracy. The black lines represent constraints in a state estimation problem. (a) illustrates a well-conditioned problem. The solution (green dot) is constrained in different directions. (b) gives an example of degeneracy. The constraints are mostly parallel such that the problem is degenerate along the blue arrow. The method evaluates the constraints online to determine degeneracy. A simple technique called solution remapping automatically separates degenerate directions (blue arrow) from well-conditioned directions (orange arrow), and only solves the problem in the well-conditioned directions. This prevents faulty solutions caused by degeneracy.

pletely and needs help in all DOF. The method automatically separates degenerate directions from well-conditioned directions. When degeneracy is determined, a simple technique called solution remapping linearly combines the solution with a best guess. The problem is only solved in well-conditioned directions, and the best guess is used in degenerate directions.

The proposed method online evaluates degeneracy for optimization-based methods, through analysis of geometric structure of the problem constraints. In a linearized system, intuitively, a constraint is a (hyper-)plane in the state space. A set of well-conditioned constraints should distribute toward different directions, constraining the solution from different angles (an example is shown in Fig. 7.1(a)). On the other hand, the case that all planes are mostly parallel corresponds to degeneracy – the solution is poorly constrained in directions parallel to the planes (as shown in Fig. 7.1(b)). I mathematically define a *degeneracy factor* as stiffness of the solution w.r.t. disturbances to the constraints. By formulating and solving an optimization problem, a closed form expression of the *degeneracy factor* containing nothing but eigenvalues and eigenvectors of the system is derived – without reinventing the wheel, the findings enrich existing eigenvalues and eigenvectors with new geometrical meanings.

The method functions as a plug-in step and can be adapted to common linear and nonlinear solvers with negligible add-in computational complexity. The method is evaluated with a custom-built state estimation system combing both vision and lidar sensors. Experimental results show that the system is able to improve estimation in environmentally degenerate cases and robustly conduct online positioning and mapping.

7.2 Problem Statement

The goal of this study is to evaluate degeneracy of an optimization-based state estimation problem, by studying the structure of the constraints in the state space. Define $\boldsymbol{x}$ as a $n \times 1$ state vector, where n is the dimension of the state space. The state estimation problem is to solve,

$$\arg\min_{\boldsymbol{x}} \boldsymbol{f}^2(\boldsymbol{x}). \tag{7.1}$$

In the case that (7.1) is a linear function of $\boldsymbol{x}$, I can directly solve for $\boldsymbol{x}$ with the singular value

decomposition or QR decomposition method [91]. On the other hand, if (7.1) is a nonlinear function, methods such as Gauss-Newton, gradient descent, and Levenberg-Marquardt [71] can be used. Most nonlinear optimization methods locally linearize the problem by computation of the Jacobian matrix of $\boldsymbol{f}$ w.r.t. $\boldsymbol{x}$,

$$\mathbf{J} = \partial \boldsymbol{f}(\boldsymbol{x}) / \partial \boldsymbol{x}. \tag{7.2}$$

Given an initial guess, the methods iteratively adjust $\boldsymbol{x}$ by usage of $\mathbf{J}$ until convergence.

Regardless of linearity, a linear problem is always involved, either as the problem itself or as a step when solving a nonlinear problem. Hence I investigate the linear problem

$$\arg\min_{\boldsymbol{x}} \|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|^2 . \tag{7.3}$$

The method considers each row in (7.3) as a (hyper-)plane in the space of $\boldsymbol{x}$, and studies geometric distribution of the planes to determine degeneracy. I make two assumptions,

- I assume that matrix $\mathbf{A}$ is appropriately weighted taking into account sensor noise. In other words, (7.3) is a linearized form of (7.1) that retains the weights of the original problem.
- I assume that the problem is full rank, i.e. not under-constrained and the state estimate is dominated by the true sensor measurements in the case that the problem itself is not degenerate.

With assumptions made, the problem can be stated as,

Problem 1 *Given a linearized system as (7.3), determine degeneracy and corresponding degenerate directions in the state space. In the case of degeneracy, prevent faulty solutions from occurring in the degenerate directions.*

7.3 Degeneracy Evaluation

7.3.1 Mathematical Derivation

This section describes the methodology to evaluate degeneracy of a linearized system. I will start with mathematical definition of the *degeneracy factor*. As shown in Fig. 7.2, the black lines represent the constraints in a system described by (7.3), and the blue dot indicates the true solution, denoted as $\boldsymbol{x}_0$. To measure degeneracy of the system, I insert an additional constraint passing through $\boldsymbol{x}_0$ as the orange line,

$$\boldsymbol{c}^T(\boldsymbol{x} - \boldsymbol{x}_0) = 0, \ ||\boldsymbol{c}|| = 1, \tag{7.4}$$

where $\boldsymbol{c}$ is a $n \times 1$ vector indicating the normal of the constraint (the black arrow in Fig. 7.2). Since the constraint intersects with $\boldsymbol{x}_0$, insertion of the constraint does not change $\boldsymbol{x}_0$. Then, I move the constraint toward its normal direction $\boldsymbol{c}$ for a certain distance δd. Correspondingly, I measure the shift of $\boldsymbol{x}_0$ in the same direction. Let δx_c be the amount of shift. For a given δd, the shift δx_c varies as a function of the direction of $\boldsymbol{c}$. Let δx_c^* be the maximum amount of shift,

$$\delta x_c^* = \max_{\boldsymbol{c}} \delta x_c. \tag{7.5}$$

I now define the *degeneracy factor* as follows,

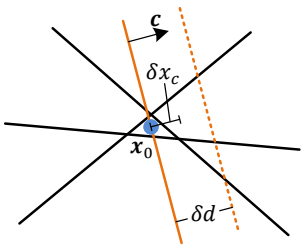


Figure 7.2: Definition of *degeneracy factor*. I insert an additional constraint (orange line) as a disturbance and inspect movement of the solution $\boldsymbol{x}_0$.

Definition 1 *The* degeneracy factor $\mathcal{D}$ *of a system is mathematically defined as* $\mathcal{D} = \delta d / \delta x_c^*$.

By such a definition, I evaluate stiffness of the solution w.r.t. disturbances to the constraints. By maximizing δx_c, I find a direction in which the solution is the least stable. The corresponding stiffness in that direction is considered a measurement of degeneracy. In other words, the degeneracy is determined by the lower-bound of stiffness w.r.t. disturbances in all possible directions. Here, note that Definition 1 is not limited to linearized systems. However, a closed form expression of $\mathcal{D}$ is available if the system is linearized. Next, Lemma 1 and Lemma 2 help us derive $\mathcal{D}$.

Lemma 1 *For the linearized system (7.3), the* degeneracy factor $\mathcal{D}$ *is a function of* $\boldsymbol{A}$*, but not* $\mathbf{b}$.

Proof: Let us start with insertion of the additional constraint passing through $\boldsymbol{x}_0$. Eq. (7.3) can be viewed as solving $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ with the l^2 norm. Stacking (7.3) with (7.4),

$$\begin{bmatrix} \mathbf{A} \\ \boldsymbol{c}^T \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{c}^T \boldsymbol{x}_0 \end{bmatrix}. \tag{7.6}$$

It is trivial to see that the solution of (7.6) is still $\boldsymbol{x}_0$. Now, I introduce a disturbance by shifting the constraint toward this normal direction by δd. This gives

$$\begin{bmatrix} \mathbf{A} \\ \boldsymbol{c}^T \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{c}^T \boldsymbol{x}_0 + \delta d \end{bmatrix}. \tag{7.7}$$

Let $\delta \boldsymbol{x}$ be the corresponding shift of $\boldsymbol{x}_0$. With the disturbance introduced, the solution of (7.7) becomes $\boldsymbol{x}_0 + \delta \boldsymbol{x}$. Applying left pseudo-inverse to the left sides of (7.6) and (7.7), where

$$\begin{bmatrix} \mathbf{A} \\ \boldsymbol{c}^T \end{bmatrix}^{-1}_{\text{left}} = (\begin{bmatrix} \mathbf{A}^T & \boldsymbol{c} \end{bmatrix} \begin{bmatrix} \mathbf{A} \\ \boldsymbol{c}^T \end{bmatrix})^{-1} \begin{bmatrix} \mathbf{A}^T & \boldsymbol{c} \end{bmatrix}, \tag{7.8}$$

and subtracting (7.6) from (7.7), I can compute $\delta \boldsymbol{x}$,

$$\delta \boldsymbol{x} = (\mathbf{A}^T \mathbf{A} + \boldsymbol{c}\boldsymbol{c}^T)^{-1} \boldsymbol{c} \delta d. \tag{7.9}$$

Recall that $||\boldsymbol{c}|| = 1$, the shift, δx_c, can be calculated as the dot product of $\boldsymbol{c}$ and $\delta \boldsymbol{x}$,

$$\delta x_c = \boldsymbol{c}^T \delta \boldsymbol{x} = \boldsymbol{c}^T (\mathbf{A}^T \mathbf{A} + \boldsymbol{c}\boldsymbol{c}^T)^{-1} \boldsymbol{c} \delta d. \tag{7.10}$$

Eq. (7.10) tells us that δx_c is a function of $\mathbf{A}$, $\boldsymbol{c}$, and δd, but not a function of $\boldsymbol{b}$. Proof complete.

Lemma 1 indicates that the degeneracy is only determined by directions of the constraints, represented by $\mathbf{A}$, and irrelevant to positions of the constraints, $\boldsymbol{b}$. This confirms the intuition in Fig. 7.1 that parallelism of the constraints introduces degeneracy while different directions of the constraints maintain a well-conditioned system. Further, the following Lemma 2 will help us derive the expression of $\mathcal{D}$.

Lemma 2 *For the linearized system (7.3), $\mathcal{D} = \lambda_{\min} + 1$, where $\lambda_{\min}$ is the smallest eigenvalue of $\boldsymbol{A}^T\boldsymbol{A}$.*

Proof: Since $||\boldsymbol{c}|| = 1$, the left and right pseudo-inverse of $\boldsymbol{c}$ are $\boldsymbol{c}_{\text{left}}^{-1} = (\boldsymbol{c}^T\boldsymbol{c})^{-1}\boldsymbol{c}^T = \boldsymbol{c}^T$ and $\boldsymbol{c}_{\text{right}}^{-T} = \boldsymbol{c}(\boldsymbol{c}^T\boldsymbol{c})^{-1} = \boldsymbol{c}$. Substituting these two equations into (7.10), I obtain

$$\begin{aligned}\delta x_c &= \boldsymbol{c}_{\text{left}}^{-1}(\mathbf{A}^T\mathbf{A} + \boldsymbol{c}\boldsymbol{c}^T)^{-1}\boldsymbol{c}_{\text{right}}^{-T}\delta d \\ &= (\boldsymbol{c}^T(\mathbf{A}^T\mathbf{A} + \boldsymbol{c}\boldsymbol{c}^T)\boldsymbol{c})^{-1}\delta d \\ &= (\boldsymbol{c}^T\mathbf{A}^T\mathbf{A}\boldsymbol{c} + 1)^{-1}\delta d. \end{aligned} \tag{7.11}$$

In (7.11), δd is given. To maximize δx_c, I equally minimize $\boldsymbol{c}^T\mathbf{A}^T\mathbf{A}\boldsymbol{c}$ in the following problem,

Problem 2 *Compute* $\mathbf{c}^*$ *to minimize function*

$$\mathbf{c}^* = \arg\min_{\mathbf{c}} \ \mathbf{c}^T\boldsymbol{A}^T\boldsymbol{A}\mathbf{c}, \ s.t. \ ||\mathbf{c}|| = 1. \tag{7.12}$$

In (7.12), since $||\boldsymbol{c}|| = 1$, Problem 2 is equal to

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}} \ \frac{\boldsymbol{c}^T\mathbf{A}^T\mathbf{A}\boldsymbol{c}}{\boldsymbol{c}^T\boldsymbol{c}}. \tag{7.13}$$

The term to be minimized in (7.13) is a Rayleigh quotient [44]. Since $\mathbf{A}^T\mathbf{A}$ is a symmetric matrix, the minimum of the quotient is equal to the minimum eigenvalue of $\mathbf{A}^T\mathbf{A}$, namely $\lambda_{\min}$. This happens when $\boldsymbol{c}^*$ is the corresponding eigenvector of $\lambda_{\min}$. Substituting $\boldsymbol{c}^*$ into (7.11), I derive

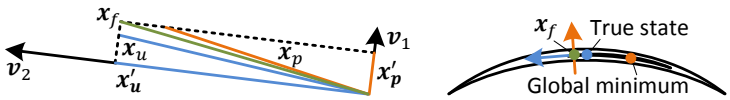
$$\delta x_c^* = \frac{\delta d}{\lambda_{\min} + 1}. \tag{7.14}$$

Therefore, the *degeneracy factor* $\mathcal{D} = \delta d/\delta x_c^* = \lambda_{\min} + 1$. Proof complete.

Lemma 2 indicates that the degeneracy is determined by $\lambda_{\min}$ of $\mathbf{A}^T\mathbf{A}$. The associated eigenvector, denoted as $\boldsymbol{v}_{\min}$, represents the first degenerate direction. Let λ_i and $\boldsymbol{v}_i$ be the i-th smallest eigenvalue and eigenvector of $\mathbf{A}^T\mathbf{A}$, $i = 1, ..., n$, where $\lambda_1 = \lambda_{\min}$ and $\boldsymbol{v}_1 = \boldsymbol{v}_{\min}$. Further expanding the above result for one more step using the theory of Rayleigh quotient, I conclude that the degeneracy in the perpendicular direction to $\boldsymbol{v}_1$,..., $\boldsymbol{v}_{i-1}$ is $\lambda_i + 1$, and the corresponding $\boldsymbol{v}_i$ indicates the i-th degenerate direction.

7.3.2 Solution Remapping

In this section, I will introduce a simple technique which is called solution remapping to handle degeneracy. I will start with a linear problem and then discuss usage of the technique in a nonlinear problem. I first find a number m, $0 \leq m \leq n$, of eigenvalues λ_1,..., λ_m that are smaller



(a) (b)

Figure 7.3: (a) Illustration of solution remapping in a linear problem. (b) An example of solution remapping applied to a nonlinear problem.

than a threshold. Here, $m = 0$ indicates a well-conditioned system and $m = n$ indicates a completely degenerate system. Let us construct three matrices as follows,

$$\mathbf{V}_p = [\boldsymbol{v}_1, ..., \boldsymbol{v}_m, 0, ..., 0]^T, \tag{7.15}$$

$$\mathbf{V}_u = [0, ..., 0, \boldsymbol{v}_{m+1}, ..., \boldsymbol{v}_n]^T, \tag{7.16}$$

$$\mathbf{V}_f = [\boldsymbol{v}_1, ..., \boldsymbol{v}_m, \boldsymbol{v}_{m+1}, ..., \boldsymbol{v}_n]^T. \tag{7.17}$$

Following the convention of Kalman filters, let us define $\boldsymbol{x}_p$ as a prediction which is the best guess of the true state. Let $\boldsymbol{x}_u$ be an update obtained from solving the system equation described in (7.3). Here, note that even though the problem is degenerate, (7.3) is still solvable and yields a solution due to noise contained in the system. The key idea of solution remapping is to use $\boldsymbol{x}_p$ in the degenerate directions, $\boldsymbol{v}_1$,..., $\boldsymbol{v}_m$, and $\boldsymbol{x}_u$ in the well-conditioned directions, $\boldsymbol{v}_{m+1}$,..., $\boldsymbol{v}_n$. The final solution, $\boldsymbol{x}_f$, is a linear combination of $\boldsymbol{x}_p$ and $\boldsymbol{x}_u$,

$$\boldsymbol{x}_f = \boldsymbol{x}'_p + \boldsymbol{x}'_u, \tag{7.18}$$

where $\boldsymbol{x}'_p = \mathbf{V}_f^{-1}\mathbf{V}_p\boldsymbol{x}_p$ and $\boldsymbol{x}'_u = \mathbf{V}_f^{-1}\mathbf{V}_u\boldsymbol{x}_u$.

Fig. 7.3(a) explains the intuition behind solution remapping, in a two dimensional example. The black axes represent the eigenvectors of a system and the lengths of the axes indicate the eigenvalues. In this example, $\boldsymbol{v}_1$ is a degenerate direction and $\boldsymbol{v}_2$ is a well-conditioned direction. With (7.18), I project $\boldsymbol{x}_p$ onto $\boldsymbol{v}_1$ to obtain $\boldsymbol{x}'_p$, and $\boldsymbol{x}_u$ onto $\boldsymbol{v}_2$ to obtain $\boldsymbol{x}'_u$. Finally, $\boldsymbol{x}_f$ is the vector sum of $\boldsymbol{x}'_p$ and $\boldsymbol{x}'_u$. Here, the threshold determining degeneracy is calculated from one test containing both well-conditioned scenes and degenerate scenes. Fig. 7.4 provides an example. The threshold $\lambda_{\min}$ is generated by a scan matching method which encounters a planar environment. The threshold is set at the mid-point of the margin between both groups.

Now, let us adapt solution remapping to a nonlinear solver in Algorithm 3. The algorithm takes a nonlinear function $\boldsymbol{f}$ and a prediction $\boldsymbol{x}_p$ as input, and computes the final solution $\boldsymbol{x}_f$. Solution remapping is done by lines 6, 7, and 10. On line 6, I compute the eigenvalues λ_i

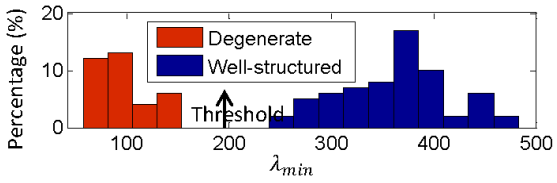


Figure 7.4: Determining threshold. From one sample dataset, I calculate the distribution of $\lambda_{\min}$ in both well-conditioned scenes and degenerate scenes. The threshold is set at the mid-point of the margin between both groups.

Algorithm 3: Nonlinear Solver with Solution Remapping

1 **input** : $\boldsymbol{f}$ (nonlinear function), $\boldsymbol{x}_p$ (predicted solution)
2 **output** : $\boldsymbol{x}_f$ (final solution)
3 **begin**
4 $\quad \boldsymbol{x}_f \leftarrow \boldsymbol{x}_p$; $O(*)$
5 $\quad$ Linearize $\boldsymbol{f}$ at $\boldsymbol{x}_p$ to get $\mathbf{A}$, $\boldsymbol{b}$, and $\mathbf{A}^T\mathbf{A}$; $O(*)$
6 $\quad$ Compute λ_i and $\boldsymbol{v}_i$ of $\mathbf{A}^T\mathbf{A}$, $i = 1, ..., n$; $O(n^3)$
7 $\quad$ Determine a number m of λ_i smaller than a threshold, construct $\mathbf{V}_p$, $\mathbf{V}_u$, and $\mathbf{V}_f$ based on (7.15)-(7.17); $O(n^2)$
8 $\quad$ **while** nonlinear iterations **do** $O(kn^2 + *)$
9 $\quad\quad$ Compute update $\Delta\boldsymbol{x}_u$; $O(*)$
10 $\quad\quad$ $\boldsymbol{x}_f \leftarrow \boldsymbol{x}_f + \mathbf{V}_f^{-1}\mathbf{V}_p\Delta\boldsymbol{x}_u$; $O(n^2)$
11 $\quad$ **end**
12 $\quad$ Return $\boldsymbol{x}_f$;
13 **end**

and eigenvectors $\boldsymbol{v}_i$, $i = 1, ..., n$. On line 7, I construct the three matrices $\mathbf{V}_p$, $\mathbf{V}_u$, and $\mathbf{V}_f$ by comparing λ_i to a threshold. The solution is only updated in the well-conditioned directions on line 10, leaving $\boldsymbol{x}_p$ in the degenerate directions unchanged. From our experience it is not necessary to update the directions for each nonlinear iteration but the first iteration only, saving computation time. Here, I use $O(*)$ to denote the original complexity of the nonlinear solver. By introduction of solution remapping, the additional add-in complexity is stated.

Theorem 1 *Algorithm 3 adds $O(kn^2 + n^3)$ time to the original nonlinear solver, where k is the number of iterations, and n is the dimension of the state space.*

Here, note that when the dimension of the state space n is constant and relatively small, the add-in complexity becomes $O(k)$. This is often the case, for example when tracking the state under 6-DOF motion. Finally, let us give an example to explain the intuition behind Algorithm 1. As shown in Fig. 7.3(b), the black curves represent elevation curves. In this example, the global minimum (orange dot) is far away from the true state (blue dot) because of the degenerate constraint structure. In fact, the global minimum is determined by noise along the degenerate direction (indicated by the blue arrow). To prevent the solution from moving toward the faulty global minimum, I use $\boldsymbol{x}_p$ in the degenerate direction. The solution is only updated along the orange arrow during optimization. The result is that I find a solution as the green dot, much closer to the true state than the global minimum.

7.4 Vision-lidar Odometry and Mapping System

7.4.1 Sensor Hardware

The study of this chapter is validated on, but not limited to a custom built camera and lidar system. The camera is a uEye monochrome camera configured at 60Hz frame rate and $752 \times$

Figure 7.5: Sensors involved in the study. The sensors are composed of an uEye camera and a custom built 3D lidar based on a Hokuyo laser scanner.

480 pixel resolution with 76° horizontal field of view. The lidar is a Hokuyo UTM-30LX laser scanner which has 180° field of view and 0.25° resolution with 40 lines/sec scanning rate. A motor rotates the laser scanner at $180°/s$ average angular speed back and forth between $-90°$ and 90° to realize 3D scanning. An encoder measures the motor rotation angle with 0.25° resolution.

7.4.2 Software System

The vision and lidar integrated motion estimation system built in the previous work [98] takes visual images and lidar clouds from the sensors in Fig. 7.5 and estimates its ego-motion to build a map of the traversed environment. The system uses a visual odometry method running at a high frequency (60Hz) followed by scan matching at a low frequency (1Hz) to refine motion estimates, hence is able to map in real-time on the move. I have chosen this system as it combines multiple components for evaluation of the proposed method. The system combines three modules as shown in Fig. 7.6, each formulates and solves a nonlinear optimization problem with the Levenberg-Marquardt method [71], and is adapted with solution remapping.

Frame to Frame Visual Odometry

The visual odometry estimates motion between two consecutive frames. I track Harris corners [38] by the Kanade Lucas Tomasi (KLT) method [61]. The scale of translation is determined from lidar range measurements. Lidar clouds are registered on a depthmap in the camera field of view and associate depth to visual features from the depthmap. For a feature point, I find three points on the depthmap that form a planar patch with a KD-tree [16]. The depth is calculated by projecting a ray from the camera center to the planar patch.

When solving for motion, the method first tries to associate depth from the depthmap. However, if depth is unavailable from the depthmap, it tries to reconstruct the depth by triangulation using the estimated motion if the feature is tracked long enough. As the last choice, the method uses the feature without depth by using a different type of constraint. The motion estimation solves an optimization problem including constraints from features both with and without depth.

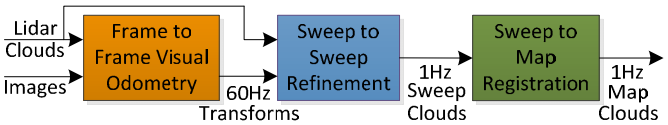


Figure 7.6: Block diagram of the motion estimation software system.

Sweep to Sweep Refinement

The refinement module matches lidar clouds between consecutive sweeps to refine the motion estimates. Here, a sweep is the process that the lidar completes for one full scan coverage, or a 180° rotation of the Hokuyo laser scanner (lasting for 1s). The drift of the visual odometry is modeled with constant velocity within a sweep. This module combines a linear motion model to remove distortion in the lidar clouds caused by drift of the visual odometry.

The scan matching uses geometric features located on local edges and planar surfaces, namely edge points and planar points. It matches an edge point and a planar point from the current sweep to an edge line segment and a planar surface patch from the previous sweep. The motion refinement minimizes overall distances from the edge points and planar points to correspondences.

Sweep to Map Registration

The registration module matches lidar clouds from sweeps to the current map and registers the lidar clouds to incrementally expand the map. As distortion caused by drift of the visual odometry is removed, this module simply assumes rigid body transformation, similar to the standard iterative closest point method [77]. Again, both edge points and planar points are used.

7.5 Experiments

Experiments are conducted with the vision-lidar motion estimation system. For each test, the sensor hardware in Fig. 7.5 is carried by a person who walks at a speed of 0.5m/s. The proposed *degeneracy factor* is compared with two other terms:

- *Inverse Maximum Covariance Eigenvalue* (*IMCE*): From the least-square regression theory [91], one is able to determine the covariance of a linear solution,

$$\Sigma = \sigma^2(\mathbf{A}^T\mathbf{A})^{-1}, \tag{7.19}$$

  where σ^2 is a variance computed from the residuals, $\sigma^2 = ||\mathbf{A}\boldsymbol{x}_0 - \boldsymbol{b}||^2/(n-m)$. Here, recall that n is the number of constraints and m is the dimension of the linear problem. Since I propose to use the eigenvalues of $\mathbf{A}^T\mathbf{A}$ to evaluate degeneracy, let us also use the eigenvalues of Σ. Here, *IMCE* is defined as inverse of the maximum eigenvalue of Σ. Additionally, to inspect how each element contributes to the covariance, let us define another term $\mathcal{R}$ as squared sum of the residuals, $\mathcal{R} = ||\mathbf{A}\boldsymbol{x}_0 - \boldsymbol{b}||^2$.

- *Inverse Condition Number* (*ICN*): The condition number [12] of a linear system is determined by the ratio between the minimum and maximum eigenvalues of $\mathbf{A}^T\mathbf{A}$, defined as $\sqrt{\lambda_m/\lambda_1}$. The term evaluates numerical condition of a linear system. A large condition number indicates ill-conditioning. In this case, the resulting solution suffers from inaccuracy due to numerical calculation errors. For comparison purposes, I take its inverse and denote it as *ICN*.

The overall experiments consist of four tests. Test 1 validates the proposed method for visual odometry (first module in Fig. 7.6), Test 2 concentrates on the two scan matching sections (second and third modules in Fig. 7.6), and Test 3 covers all three modules. I consider failure cases

of the visual odometry and classify them into motion blur, dynamic environment, and feature-poor scene. Motion blur can be handled by a fast camera and image frame rate, and dynamic environments can be addressed by outlier rejection or distraction suppression technique [63]. A feature-poor scene is the most relevant to degeneracy. Typically, this occurs where the camera faces a texture-less environment, points to the sun, or is in a dark environment. Likely, few features are available or the features are extracted from a concentrated area within the images.

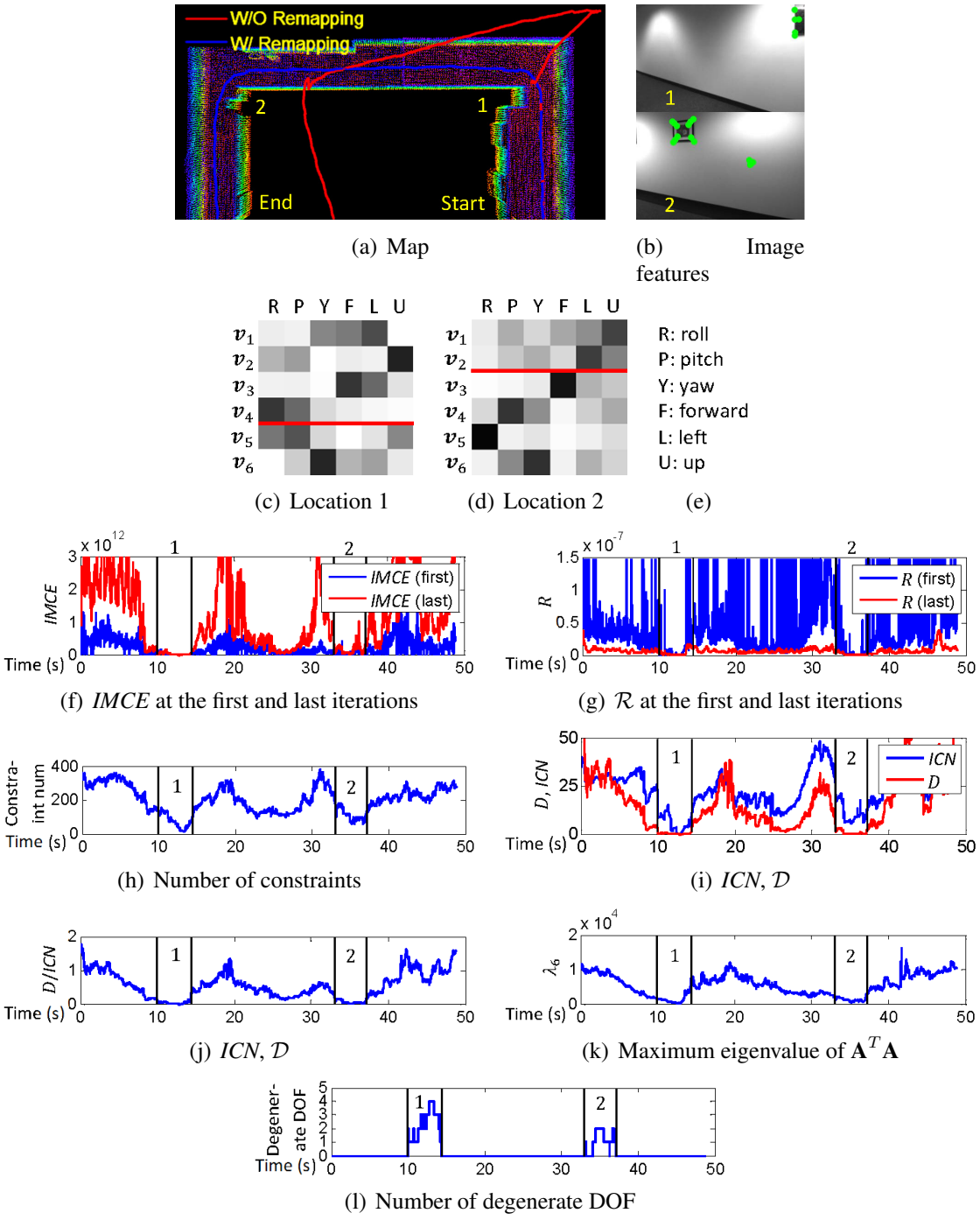
With such consideration, Test 1 is conducted in a corridor. As shown in Fig. 7.7, the path goes through two feature-poor corners labeled with numbers 1 and 2. Fig. 7.7(a) presents the estimated trajectory and the map built when degeneracy is eliminated by solution remapping (Algorithm 1). Here, prediction is provided by a constant velocity model. Fig. 7.7(b) shows one sample image from each of the labeled corners. In Fig. 7.7(c)-(d), the eigenvectors of matrix $\mathbf{A}^T\mathbf{A}$ corresponding to the two images in Fig. 7.7(c) are shown. Darker blocks indicate larger values, and rows in top-down order correspond to small to large eigenvalues. The directions above the red lines are degenerate. Careful comparison finds that in location 1, the most degenerate direction is lateral translation (the darkest block on the first row is labeled with "L: left"). This makes sense as features in location 1 are vertically distributed, resulting in lateral translation to be poorly constrained. Correspondingly, the red trajectory jumps leftward. In location 2, Fig. 7.7(d) indicates degeneracy mostly in vertical translation (the darkest block on the first row is labeled with "U: up"). This is because the features span horizontally. The red trajectory jumps upward.

Next I will examine *IMCE*. In Fig. 7.7(f), the values are shown for both the first and last iterations in the nonlinear optimization. One sees that the values of *IMCE* are noisy mostly due to the noisy nature of the squared sum of the residuals $\mathcal{R}$ (Fig. 7.7(g)). Note that the value at the first iteration is more meaningful as ideally one wants to detect degeneracy from the beginning of the optimization so that solution remapping can be introduced. However, one also sees *IMCE* at the first iteration is much noisier than at the last iteration as $\mathcal{R}$ is not yet minimized. In Fig. 7.7(h), the number of constraints is shown, which is also involved in the computation of the covariance Σ and possibly brings in uncertainty.

Finally I compare *ICN* and *degeneracy factor* $\mathcal{D}$ in Fig. 7.7(i). Here, *ICN* is manually scaled to match with $\mathcal{D}$. Further, I compare the ratio of the two terms $\mathcal{D}$/*ICN* in Fig. 7.7(j). It is apparent that the value of the ratio reduces in locations 1 and 2, indicating $\mathcal{D}$ is more effective. The reason is that *ICN* calculates the ratio between the minimum and maximum eigenvalues of $\mathbf{A}^T\mathbf{A}$ such that the maximum eigenvalue also has effect on the term. In Fig. 7.7(k), the maximum eigenvalue λ_6 is shown which decreases in locations 1 and 2. The reduction of λ_6 contributes to the increase of *ICN*. Here, the consideration is that degeneracy should not be determined by the well-conditioned directions but by the degenerate directions themselves. Finally, the number of degenerate DOF during Test 1 in Fig. 7.7(l) is shown.

Then, in Test 2, I choose an environment which contains a piece of flat ground as in Fig. 7.8. Traveling on the flat ground results in sliding of the scan matching on the red curve in Fig. 7.8(a). Correspondingly, the top three rows of Fig. 7.8(c)-(d) indicate that degeneracy occurs in the directions of forward translation, lateral translation, and yaw rotation, meaning that translation parallel to the ground and rotation perpendicular to the ground are poorly constrained.

Looking into the rest of the figure, one finds that the value of *IMCE* is either noisy or does not decrease obviously (for the sweep to sweep refinement section in Fig. 7.8(f) and the sweep to map registration section in Fig. 7.8(g)). Again, this is because *IMCE* is determined by multiple



(a) Map (b) Image features

(c) Location 1 (d) Location 2 (e)

(f) *IMCE* at the first and last iterations

(g) $\mathcal{R}$ at the first and last iterations

(h) Number of constraints

(i) *ICN*, $\mathcal{D}$

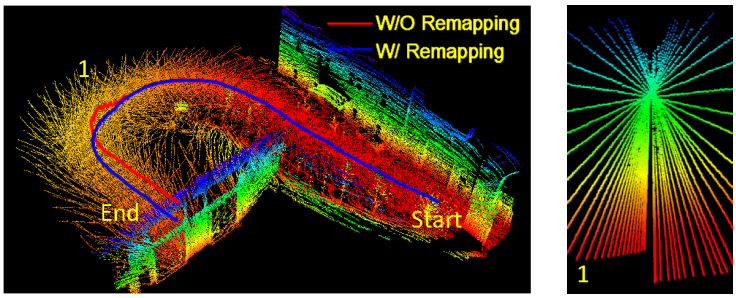
(j) *ICN*, $\mathcal{D}$

(k) Maximum eigenvalue of $\mathbf{A}^T\mathbf{A}$

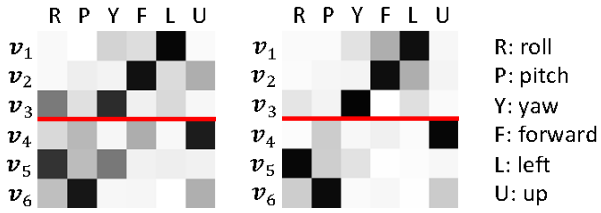
(l) Number of degenerate DOF

Figure 7.7: Test 1: Visual odometry degeneracy in feature-poor environment.

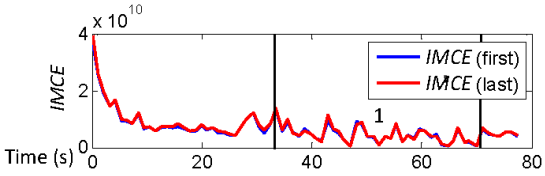
terms including squared sum of the residuals $\mathcal{R}$ and the number of constraints. Here, note that the values of *IMCE* do not differ much between the first and last iterations because the scan matching only refines motion estimates generated by the visual odometry. The value of R reduces little during the course of optimization. Fig. 7.8(h)-(i) compare *ICN* and $\mathcal{D}$. One can see an obvious drop of value between 35-70s when the degeneracy occurs. In Fig. 7.8(j), the ratio $\mathcal{D}$/*ICN* also



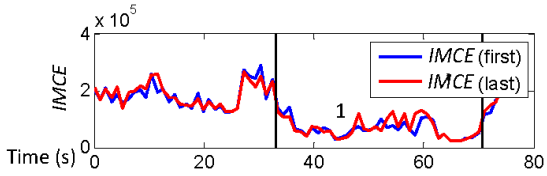
(a) Map (b) Lidar scan



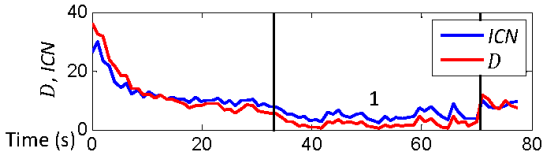
(c) Sweep to sweep (d) Sweep to map (e)



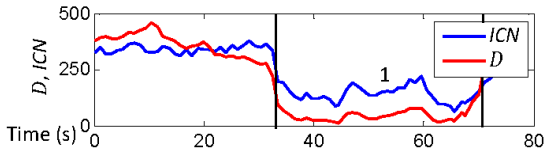
(f) *IMCE* of sweep to sweep refinement section



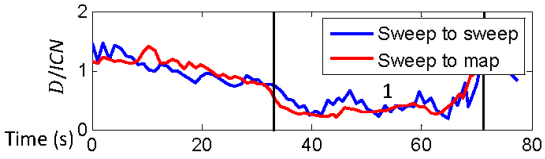
(g) *IMCE* of sweep to map registration section



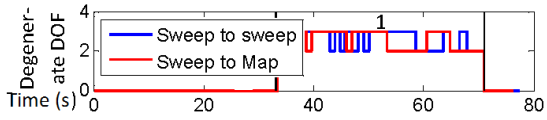
(h) *ICN*, $\mathcal{D}$ of sweep to sweep refinement section



(i) *ICN*, $\mathcal{D}$ of sweep to map registration section



(j) $\mathcal{D}$/*ICN* for both sections

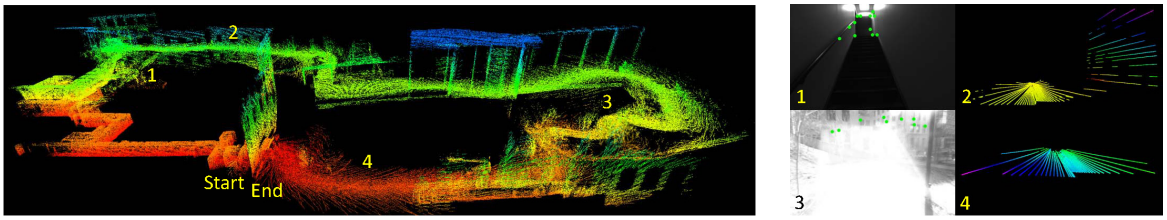


(k) Number of degenerate DOF

Figure 7.8: Test 2: Scan matching degeneracy on flat ground.

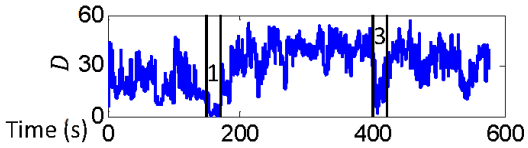
decreases slightly during this interval. In Fig. 7.8(k), one sees that the method detects three degenerate DOF corresponding to the top three rows in Fig. 7.8(c)-(d).

Finally, I conduct a larger scale test in Test 3 containing indoor and outdoor environments, as shown in Fig. 7.9. The path starts in front of a building, passes through the building and exits to the outside, climbs stairs, and follows a small trail to come back and finish at the exact starting position. The overall traveling distance is 538m. The path contains two degenerate scenes for the visual odometry in locations 1 and 3 due to undesirable lighting conditions, and two degenerate scenes for the scan matching in locations 2 and 4. In location 2, the lidar sees the flat ground and

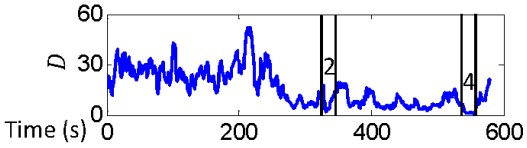


(a) Map

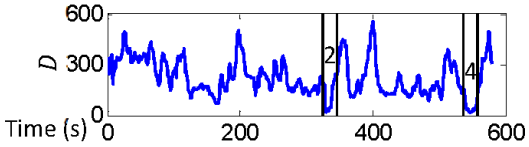
(b) Image features and lidar scans



(c) $\mathcal{D}$ of frame to frame visual odometry section



(d) $\mathcal{D}$ of sweep to sweep refinement section



(e) $\mathcal{D}$ of sweep to map registration section

Figure 7.9: Test 3: Complete test including indoor and outdoor environments. The overall path is 538m long, starting in front of a building, passing through the building with stairs and following a trail on hilly terrain to return. The path encounters four degenerate scenes labeled with 1-4.

one wall on its right side causing degeneracy in forward translation. In location 4, the lidar only sees the flat ground similar to Test 2. One observes that the value of $\mathcal{D}$ drops in locations 1 and 3 in Fig. 7.9(c) and in locations 2 and 4 in Fig. 7.9(d)-(e).

Fig. 7.10 further compares the estimated trajectories. The green curve is without solution remapping, hence jumps occur along the path. The red curve is estimated with consistent motion prior added to the motion estimation problems. The visual odometry section takes motion prior from a constant velocity model, and each scan matching section takes output from the previous section. Here, the motion prior is given as little as possible to eliminate degeneracy. However, it still causes drift at the end to be about three times as large as on the blue curve (proposed

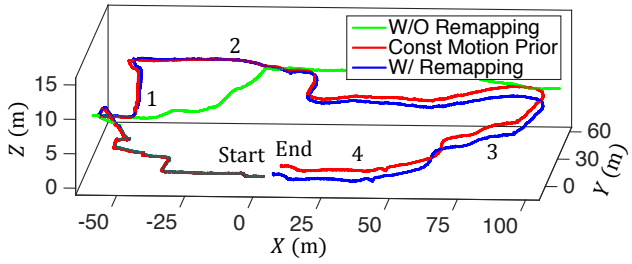


Figure 7.10: Trajectories of Test 3. The green curve is estimated without solution remapping, and jumps occur along the path. The red curve uses consistent motion prior to eliminate degeneracy. This results in more drift consequently. The blue curve uses the proposed solution remapping. The position error at the end is 0.71% of the 538m trajectory length.

method). Thanks to solution remapping, the system is able to conquer all degeneracy resulting in a 0.71% relative position error at the end of the blue curve compared to the distance traveled.

7.6 Discussion

As the solution remapping linearly combines the prediction and update, one can argue that this is a variant of the Kalman filter. The response is that a filter handles accuracy, while the proposed method is devoted to degeneracy and robustness. The difference is that filters average multiple noisy measurements to gain better accuracy. When working with degeneracy, I consider the solution to be completely unusable in the degenerate directions and simply take the prediction instead. The second response is that solution remapping can be adapted to individual iterations of nonlinear optimization. A filter only takes the final solutions to seed steps. Finally, a filter can be the following step of the proposed method taking its output for further integration.

7.7 Conclusion

Robustness of estimation is critical for state estimation and especially for autonomous vehicles. This chapter improves robustness by handling environmental degeneracy. A *degeneracy factor* is mathematically defined and derived. Degeneracy is evaluated through computation of the associated eigenvalues and eigenvectors. When degeneracy occurs, the proposed method automatically separates the state space and partially solves the problem only in well-conditioned directions. In degenerate directions, a best guess is used instead. The method is tested with a custom-built vision and lidar system in a number of challenging scenarios, for online positioning and mapping. Experimental results show that the system is able to conquer environmentally degenerate moments, to reliably estimate state, and build accurate 3D representations of the environment.

Chapter 8

Air-ground Collaborative Mapping

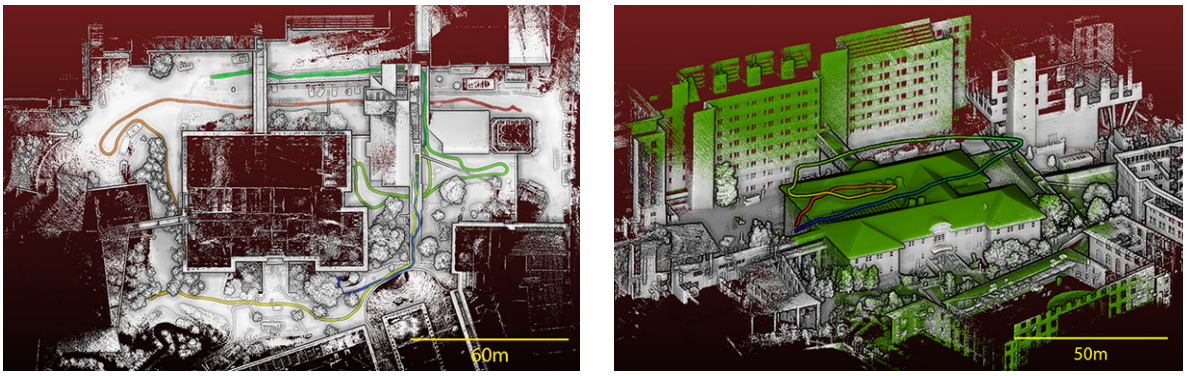
Studies are present in this chapter to enable aerial and ground-based collaborative mapping in GPS-denied environments. The work utilizes a system that incorporates a laser scanner, a camera, and a low-grade IMU in a miniature package which can be carried by a light-weight aerial vehicle. If a map is available, the system can localize on the map and merge maps from separate runs for collaborative mapping. Experiments are conducted in urban and vegetated areas. The work enables autonomous flights in cluttered environments through building and trees.

8.1 Introduction

This chapter is aimed at solving a mapping problem. In particular, I seek for collaboration between mapping from the ground and air due to each own characteristics. Ground-based mapping is not prone to limitations of space or time. Typically, a mapping device carried by a ground vehicle is suitable for mapping in large scale and can move at a high speed. On the other hand, a tight area can be mapped in a hand-held deployment. However, ground-based mapping is limited by the sensor's altitude, difficult to realize a top-down looking configuration. As illustrated in Fig. 8.1, the ground-based experiment produces a detailed map of the surroundings of a building, while the roof has to be mapped from the air. If a small aerial vehicle is used, aerial mapping is limited by time due to the short lifespan of batteries. Space also needs to be open enough for aerial vehicles to operate safely. In this chapter, experimental studies are carried out to pursue the advantages of both.

The collaborative mapping is based on the previous work [97, 99, 100] which develops a data processing pipeline for real-time ego-motion estimation and mapping. The method utilizes a laser scanner, a camera, and a low-grade IMU, processes data through multi-layer optimization. The resulting motion estimates are at a high rate (200Hz) with a low drift (typically $<0.1\%$ of the distance travel).

Benefit from the high-accuracy processing pipeline, the chapter develops a method to merge the maps from the ground and air in real-time. This is by localization of one output w.r.t. the map from the other. In the existing literature, prior map based localization often involves particle filtering [9, 28, 32, 74, 80, 89]. In addition, Nieuwenhuisen et al use multi-resolution scan matching to localize an aerial vehicle on a point cloud map [69]. Also employing scan matching,



(a) (b)

Figure 8.1: Example of air-ground collaborative mapping. (a) shows the ground-based map and sensor trajectory (colored curve starting with blue and ending with red) produced by an operator holding a sensor pack and walking around a building at 1-2m/s for 914m of travel. The ground-based map covers details surrounding the building except the roof. Then in (b), the same sensor pack is mounted to an aerial vehicle flying over the building at 2-3m/s for 269m. The green point cloud in (b) is the aerial map and the colored curve is the sensor trajectory in the air.

the method enables high-speed navigation (up to 15m/s) and large scale mapping (over 1km).

While the proposed scheme fulfills collaborative mapping, it further reduces the complexity of aerial deployments. With a ground-based map, flight paths are defined and the aerial vehicle conducts mapping in autonomous missions. In experiments, the aerial vehicle is able to accomplish challenging flight tasks autonomously.

8.2 Method

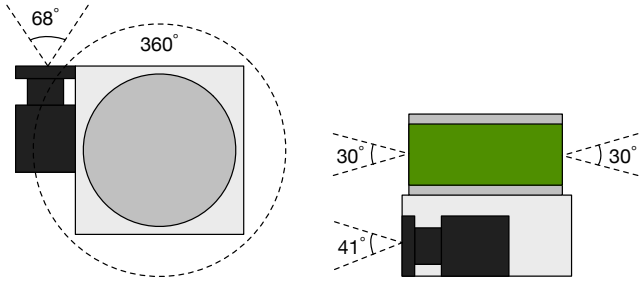
8.2.1 Sensor Configuration

Fig. 8.2 presents the sensor/computer pack utilized by the chapter to enable collaborative mapping. The processing software is not limited to a particular sensor configuration. However, introducing sensors in the front helps readers understand the technology. The sensor pack (see Fig. 8.2(a)) consists of a Velodyne Puck laser scanner generating 0.3 million points/second, a camera at 640×360 pixels resolution and 50Hz frame rate, and a low-grade IMU at 200Hz. An onboard i7 computer processes data from the sensors in real-time for ego-motion estimation and mapping. Fig. 8.2(c) and Fig. 8.2(d) illustrate the sensor field of view. An overlap is shared by the laser and camera, with which, the processing software associates depth information from the laser to image features (more discussion in the next section).

8.2.2 Odometry and Mapping

The odometry and mapping method is originally proposed in [99]. For completeness, I include an overview of the method. The software processes data from a range sensor such as a laser

(a)



(b) (c)

Figure 8.2: (a) Sensor pack including a Velodyne Puck laser scanner, a camera, and a low-grade IMU. An onboard i7 computer processes data from the sensors to conduct real-time ego-motion estimation and mapping. (b)-(c) Horizontal and vertical field of view of the laser and camera.

scanner, a camera, and an inertial sensor. Instead of combining data from all sensors in a large, full-blown problem, I parse the problem as multiple small problems, solve them sequentially in a coarse-to-fine manner. Fig. 8.3 gives a block diagram of the software system. In such a system, modules in the front conduct light processing, ensuring high-frequency motion estimation robust to aggressive motion. Modules in the back take sufficient processing, run at low frequencies to warrant accuracy of the resulting motion estimates and maps.

The software starts with IMU data processing (orange module in Fig. 8.3). This module runs at the IMU frequency to predict the motion based on IMU mechanization. The result is further processed by a visual-inertial coupled method (green module in Fig. 8.3). The method tracks distinctive image features through the image sequence and solves for the motion in an optimization problem. Here, laser range measurements are registered on a depthmap, with which, depth information is associated to the tracked image features. Since the sensor pack contains a single camera, depth from the laser helps solve scale ambiguity during motion estimation.

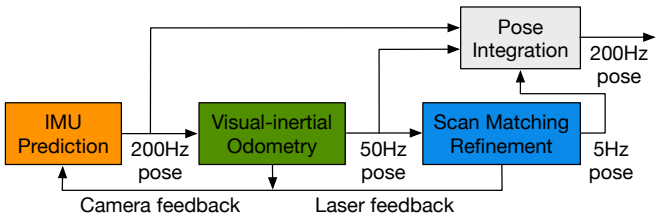


Figure 8.3: Block diagram of the laser-visual-inertial odometry and mapping software system.

The estimated motion is used to register laser scans locally. In the third module (blue module in Fig. 8.3), these scans are matched to further refine the motion estimates. The matched scans are registered on a map while scans are matched to the map. To accelerate the processing, scan matching utilizes multiple CPU threads in parallel. The map is stored in voxels to accelerate point query during scan matching. Because the motion is estimated at different frequencies, a fourth module in the system (gray module in Fig. 8.3) takes these motion estimates for integration. The output holds both high accuracy and low latency beneficial for vehicle control.

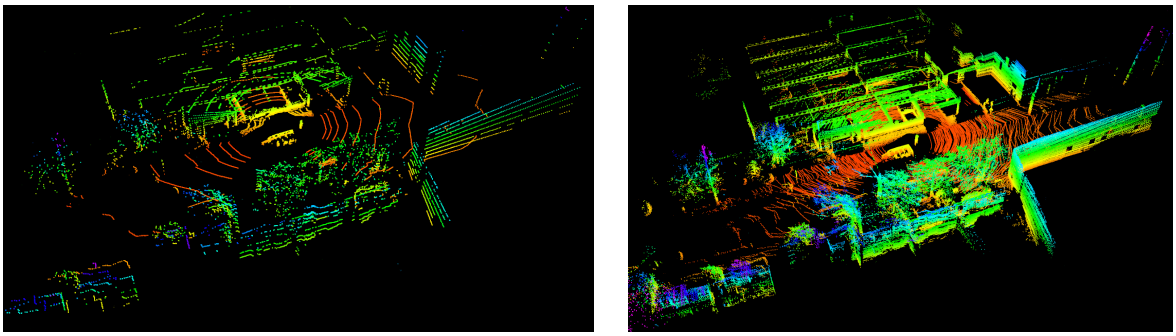
The modularized system also ensures robustness w.r.t. sensor degradation, by selecting "healthy" modes of the sensors when forming the final solution. For example, when a camera is in a low-light or texture-less environment such as pointing to a clean and white wall, or a laser is in a symmetric or extruded environment such as a long and straight corridor, processing typically fails to generate valid motion estimates. The system automatically determines a degraded subspace in the problem state space [101]. When degradation happens, the system only solves the problem partially in the well-conditioned subspace of each module. The result is that the "healthy" parts are combined to produce the final, valid motion estimates.

8.2.3 Localization and Map Merging

When a map is available, the method described in the previous section can be extended to utilize the map for localization. This is using a scan matching method similar to the blue block in Fig. 8.3. The method extracts two types of geometric features – points on edges and planar surfaces, based on the curvature in local scans. Feature points are matched to the map. An edge point is matched to an edge line segment, and a planar point is matched to a local planar patch. On the map, the edge line segments and local planar patches are determined by examining the eigenvalues and eigenvectors associated with local point clusters. The map is stored in voxels to accelerate processing. The localization solves an optimization problem minimizing the overall distances between the feature points and their correspondences. Due to the fact that the high-accuracy odometry estimation is used to provide initial guess to the localization, the optimization usually converges in 2-3 iterations.

Compared to the previous section, the difference is that the localization does not process individual scans but stacks a number of scans for batch processing. Thanks to the high-accuracy odometry estimation, scans are registered precisely in a local coordinate frame where drift is negligible over a short period of time (a few seconds). A comparison is given in Fig. 8.4, where Fig. 8.4(a) is a single scan that is matched in the previous section (scan matching executes at 5Hz), and Fig. 8.4(b) shows stacked scans over two seconds, which are matched during localization (scan matching runs at 0.5Hz). One can see the stacked scans contain significantly more structural details, contributing to the localization accuracy and robustness w.r.t. environmental changes. Additionally, low-frequency execution keeps the CPU usage to be minimal for onboard processing (localization consumes about 10% of a CPU thread).

The localization is compared to a particle filter based implementation. The odometry estimation provides the motion model to the particle filter. It uses a number of 50 particles. At each update step, the particles are resampled based on low-variance resampling [88]. Comparison results are shown in Fig. 8.5 and Table 8.1. Here, errors are defined as the absolute distances from localized scans to the map. During the evaluation, I choose a number of planar surfaces and



(a) (b)

Figure 8.4: Comparison of scans involved in odometry estimation and localization. In odometry estimation, each individual scan is processed in scan matching at 5Hz. While in localization, a number of locally registered scans are stacked and batch processed at 0.5Hz.

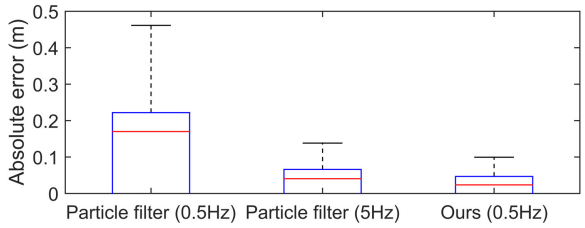


Figure 8.5: Comparison of scan matching accuracy in localization. I use the absolute distances from localized scans to the corresponding local patches on the map as the metric. Points are selected from a number of planar surfaces in Fig. 8.4. The red lines illustrate medians, the blue boxes represent 75% of the distributions, and the black lines are the maximum errors.

use the distances between points in localized scans to the corresponding planar patches on the map. Fig. 8.5 shows the error distribution. When running the particle filter at the same frequency as our method (0.5Hz), the resulting error is five times as large. While in Table 8.1, the CPU processing time is more than twice of ours. In another test, running the particle filter at 5Hz helps reduce the error to be slightly larger than our method. However, the corresponding CPU processing time increases to over 22x of ours. These results imply a particle filter based method does not take full advantage of the high-accuracy odometry estimation.

Table 8.1: Comparsion of CPU processing time in localization. When running the particle filter at 5Hz, sensor data is processed at 25% of real-time speed due to high CPU demand.

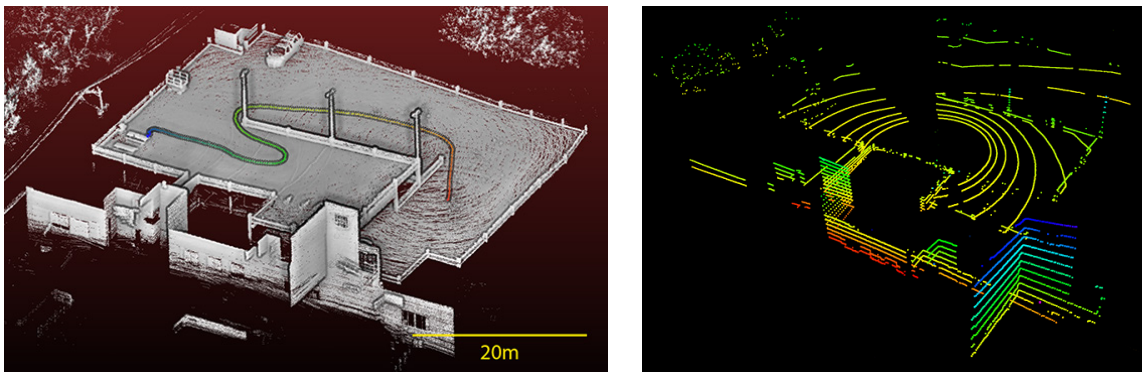
Method	Particle filter		Ours
Frequency	0.5Hz	5Hz	0.5Hz
Time per execution	493ms	478ms	**214ms**
Time per second	247ms	2390ms	**107ms**

8.3 On Sensor Orientation

In previous work [99], I studied the system performance w.r.t. sensor degradation. I concluded that the system is robust to individual sensor failures, i.e. when the laser or camera is degraded, the corresponding module is bypassed while the rest of the system is staggered to generate the solution. In this section, I will further carry out studies where the sensor pack is orientated differently. This is especially motivated by aerial mapping due to the fact that during "up and away" flights, the sensor pack has to be tilted downward in order to capture data from the ground.

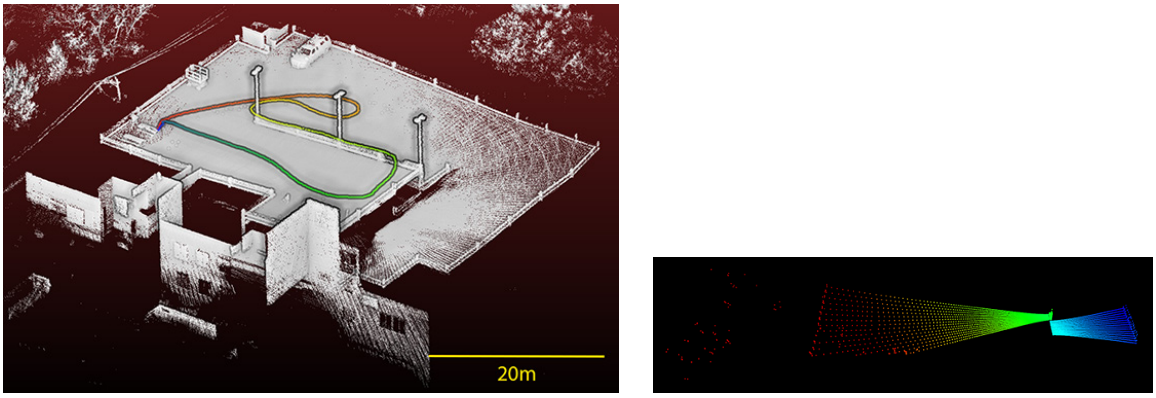
The first set of study is conducted in Fig. 8.6 and Fig. 8.7. First, the sensor pack is carried horizontally in a garage building. Fig. 8.6(a) shows the map built and sensor trajectory. Fig. 8.6(b) is a single scan. In this scenario, the scan contains sufficient structural information. When bypassing the camera processing module ((green module in Fig. 8.3)), the system produces the same trajectory as the full pipeline. On the other hand, I run another test with the sensor pack tilted vertically down toward the ground. The results are shown in Fig. 8.7. In this scenario, structural information in a scan is much sparser (see Fig. 8.7(b)). The processing fails without usage of the camera and succeeds with the full pipeline. The results indicate the camera is critical for high-altitude flights where tilting of the sensor pack is required.

The second set of study compares the drift rate w.r.t. different sensor orientations. As shown in Fig. 8.8, the sensor pack is held by an operator walking through a circle at 1-2m/s speed with an overall traveling distance of 410m. Fig. 8.8(a) shows the map built and sensor trajectory with a horizontally orientated sensor configuration. The sensor is started and stopped at the same position. The test produces 0.18m of drift through the path, resulting in 0.04% of relative position error in comparison to the distance traveled. Then, the operator repeats the path with two sensor packs held at 45° and 90° angles, respectively. The resulting sensor trajectories are shown in Fig. 8.8(b). Clearly, tilting introduces more drift, where the relative position errors are 0.6% at 45° (blue dash curve) and 1.4% at 90° (red dash-dot curve). Finally, by localizing on the map in Fig. 8.8(a), the drift is canceled and both configurations result in the black solid curve.



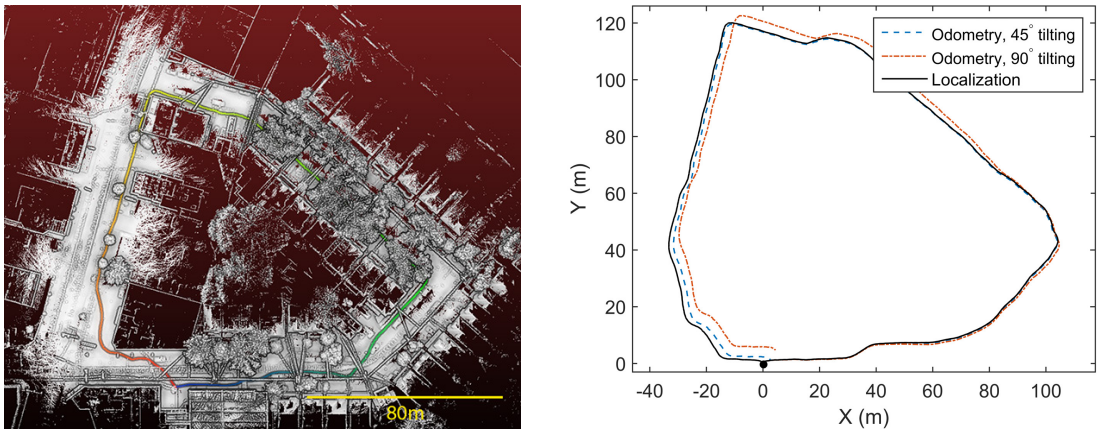
(a) (b)

Figure 8.6: Example of horizontally orientated sensor test. The processing succeeds with and without the camera, both yield the same map and sensor trajectory (colored curve starting with blue and ending with red) in (a). (b) shows a raw laser scan during the test which contains plenty of structural information for scan matching based methods to function.



(a) (b)

Figure 8.7: Example of vertically orientated sensor test. The processing succeeds only when the camera is present, which yields the map and sensor trajectory in (a). Due to the lack of structural information in laser data (see a raw laser scan in (b)), odometry estimation requires assistance from the camera. Methods only replying on scan matching are not functional.



(a) (b)

Figure 8.8: Accuracy comparison between horizontally orientated and downward tilted sensor tests. (a) shows the map and sensor trajectory (colored curve starting with blue and ending with red) of a horizontally orientated setup. The sensor pack is started and stopped at the same position, and held by an operator who walks at 1-2m/s through a circle for 410m. The odometry estimation produces 0.18m of drift resulting in 0.04% of relative position error w.r.t. the distance traveled. (b) shows results of a repeated test with the operator holding two sensor packs, one at 45° (blue dash curve) and the other at 90° (red dash-dot curve). Tilting at 45° results in 2.3m of drift and correspondingly 0.6% of relative position error, while tilting at 90° generates 5.8m of drift and 1.4% of relative position error. Finally by localizing w.r.t. the map in (a), both setups produce trajectories as the black solid curve. All trajectories start at the black dot.

Figure 8.9: DJI S1000 aircraft with sensor pack. The aircraft is built with a GPS receiver (on top of the aircraft). GPS data is not used in mapping or autonomous missions as in this chapter.

8.4 Results

8.4.1 Drone Hardware

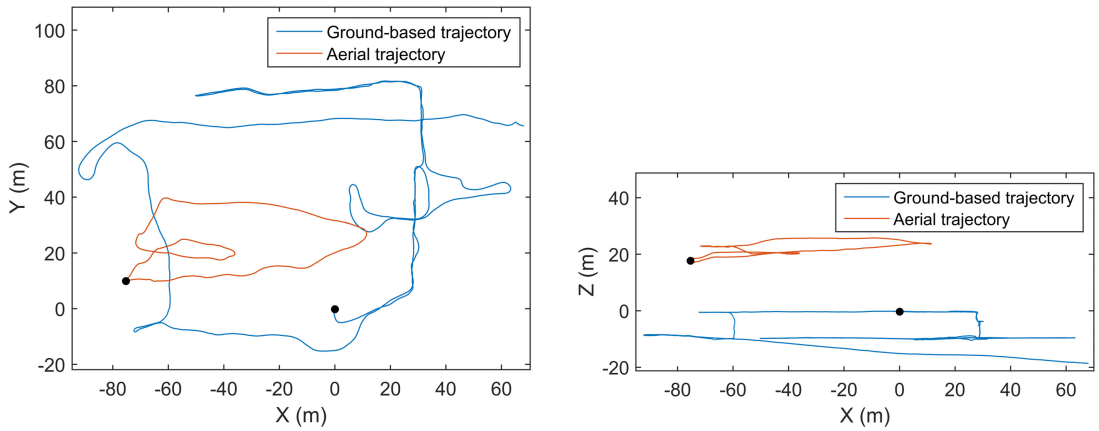
The drone platform is a DJI S1000 aircraft as shown Fig. 8.9. The aircraft weights 6.8kg (including batteries) and can carry a maximum of 4.2kg payload. The sensor/computer pack is mounted to the bottom of the aircraft, weighting 1.7kg. The bottom right of the figure shows the remote controller. During autonomous missions, the remote controller is operated by a safety pilot to override the autonomy if necessary. Note that the aircraft is built with a GPS receiver (on top of the aircraft). GPS data is not used in mapping or autonomous missions through the chapter.

8.4.2 Teleoperated Flight Results

In the first collaborative mapping experiment, an operator holds the sensor pack and walks around a building. Results are shown in Fig. 8.1. In Fig. 8.1(a), the ground-based mapping covers surroundings of the building in detail, conducted at 1-2m/s over 914m of travel. As expected, the roof of the building is empty on the map. Second, the drone is teleoperated to fly over the building. In Fig. 8.1(b), the flight is conducted at 2-3m/s with a traveling distance of 269m. The processing uses localization w.r.t. the map in Fig. 8.1(a). That way, the aerial map (green points) is merged with the ground-based map (white points). After the ground-based map is built, the take-off position of the drone is determined on the map. The sensor starting pose for the aerial mapping is known, and from which, the localization starts. Fig. 8.10 presents the aerial and ground-based sensor trajectories, in top-down and side views.

8.4.3 Autonomous Flight Results

Further, I conduct autonomous flights to realize aerial mapping. In Fig. 8.11, a ground-based map is built first by hand-held mapping at 1-2m/s for 672m of travel around the flight area. The map and sensor trajectory are shown in Fig. 8.11(a). Then based on the map, way-points are defined and the drone follows the way-points to conduct aerial mapping. As shown in Fig. 8.11(b), the colored curve is the flight path, the large colored points on the curve are the way-points, and the green points form the aerial map. In this experiment, the drone takes off inside a shed on the



(a) (b)

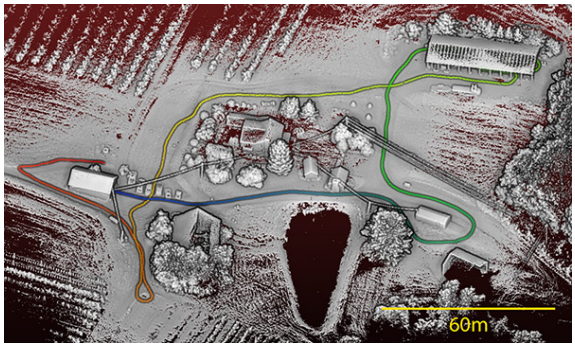
Figure 8.10: Sensor trajectories corresponding to Fig. 8.1. The ground-based mapping is at 1-2m/s with an overall distance of 914m. The aerial mapping is at 2-3m/s with 269m of travel. The trajectories start at the black dots.

left side of the figure, flies across the site and passes through another shed on the right side, then returns to the first shed to land. The speed is 4m/s crossing the site and 2m/s passing through the shed. Fig. 8.11(c) and Fig. 8.11(d) are two images taken by an onboard camera when the drone flies toward the shed on the right and is about to enter the shed. Fig. 8.11(e) shows the estimated speed during the mission.

Finally, I conduct another experiment over a longer distance. As shown in Fig. 8.12, the ground-based mapping involves an off-road vehicle driven at 10m/s from the left end to the right end, over 1463m of travel. With the ground-based map and way-points, the autonomous flight crosses the site. Upon take-off, the drone ascends to 20m high above the ground at 15m/s. Then, it descends to 2m above the ground to fly through a line of trees at 10m/s. The flight path is 1118m long as the colored curve in Fig. 8.12(b). Two images are taken as the drone flies high above the trees (see Fig. 8.12(c)) and low underneath the trees (see Fig. 8.12(d)).

8.5 Conclusion

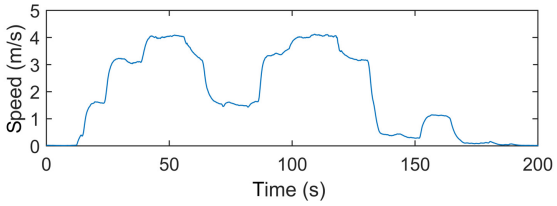
The chapter presents experimental studies on collaborative mapping from the ground and air. Utilizing a miniature sensor pack consisted of a laser scanner, a camera, and a low-grade IMU, aerial and ground-based mapping is demonstrated while the maps are merged in real-time during the flights. This benefits from a data processing pipeline with multi-layer optimization – modules in the front execute at high frequencies to handle aggressive motion and produce high-rate motion estimates, while modules in the back run at low frequencies and take sufficient computation to warrant accuracy. The system is evaluated in both teleoperated and autonomous flights, through cluttered environments and at high speeds (up to 15m/s).



(a)

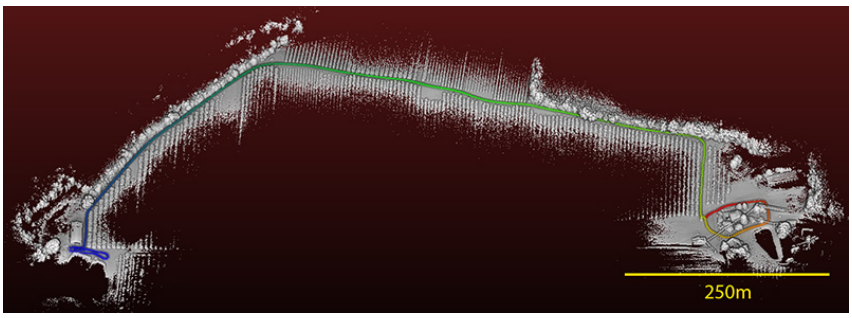
(b)

(c) (d)

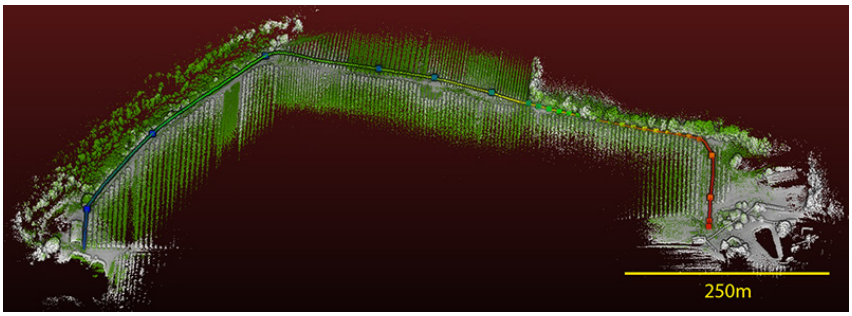


(e)

Figure 8.11: Autonomous flight result through a shed. (a) shows the ground-based map and sensor trajectory (colored curve starting with blue and ending with red) built by an operator holding the sensor pack and walking at 1-2m/s for 672m. With the ground-based map, way-points are defined and the drone executes an autonomous mission. It takes off inside a shed on the left side of the figure, navigates across the site at 4m/s, passes through another shed on the right side at 2m/s, and returns to the first shed over 390m of travel. In (b), the green point cloud is the aerial map, the colored curve is the sensor trajectory in the air, and the large points on the curve are the way-points. (c) and (d) are two images taken by an onboard camera when the drone flies between the sheds and is about to enter the shed on the right. (e) shows the estimated speed through the route.



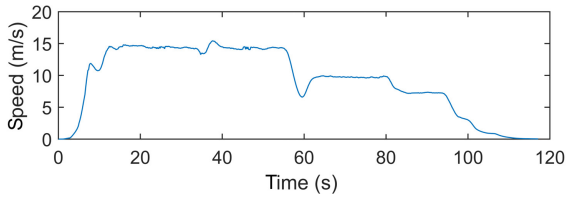
(a)



(b)

(c)

(d)



(e)

Figure 8.12: Autonomous flight result over a long-run. (a) shows the ground-based map and sensor trajectory (colored curve starting with blue and ending with red) generated by the sensor pack mounted to an off-road vehicle, driven at 10m/s for 1463m. (b) shows the result of the autonomous mission from an 1118m flight. During the mission, the drone first flies at 20m above the ground at 15m/s and then descents to 2m above the ground through a line of trees at 10m/s. The green point cloud is the aerial map, the colored curve is the flight trajectory, and the large points on the curve are the way-points. (c) and (d) are from an onboard camera when the drone flies high above the trees and low underneath the trees. (e) shows the estimated speed through the route.

Chapter 9

Conclusion

The thesis considered the problem of real-time ego-motion estimation while at the same time developing a map of the traversed environment, by leveraging range, vision, and inertial sensing. The problem was introduced in Chapter 1. Related work was summarized in Chapter 2, regarding both vision and laser based methods in the existing literature.

Chapter 3 presented a vision-based ego-motion estimation method that can be assisted by range measurements. The method registers a depthmap, with which, the method associates depth information to tracked visual features. Further, the method utilizes both features with and without association of depth information to carry out the estimation.

Chapter 4 introduced a lidar-based method which produces estimation of ego-motion as well as registration of a coherent map. The method involves two levels of processing with an odometry algorithm running at a high frequency to compute velocity estimates, and a mapping algorithm executing at a low frequency to develop accurate maps.

Chapter 5 detailed the complete processing pipeline fusing range, vision, and inertial data. The pipeline incorporates processing through multi-layer optimization. The pipeline also adopts a two-level voxel representation – voxels at the first level are for map storage, and voxels at the second level are for map retrieving during scan matching. Further, it employs an implementation using multiple CPU threads to accelerate scan matching.

Chapter 6 further compared the three methods using the KITTI odometry benchmark.

Chapter 7 focused on dealing with sensor degradation. The method determines and separates a degraded subspace from a well-conditioned subspace of the problem, and solves the problem in the well-conditioned subspace to prevent state estimation failures.

Chapter 8 explored the ego-motion estimation and mapping methods in aerial applications by merging data from the ground and air. The work further enables autonomous drone navigation with motion estimates and flight way-points w.r.t. a ground-based map.

Throughout this thesis, each proposed aspect was validated with data from real sensors.

9.1 Possible Extensions in Future Work

A number of closely related avenues for future extensions of this thesis include:

- Incorporating global positioning data such as from a GPS in the processing pipeline. Global positioning data can be helpful to cancel ego-motion estimation drift over a long distance of travel and register maps in a global coordinate frame.
- Involving dynamic vision sensors to further improve estimation robustness w.r.t. aggressive motion. A dynamic vision sensor reports data only on pixels with illumination changes, delivering both a high rate and a low latency. Direct methods may be used to realize image matching with a dynamic vision sensor for ego-motion estimation.
- Implementing parallel processing to execute on general purpose GPU or FPGA and therefore enable data processing in larger amount and higher frequencies.
- Introducing loop closures to remove ego-motion estimation drift by global smoothing.

Bibliography

[1] Robert Andersen. *Modern methods for robust regression*. Sage, 2008. 3.4.2, 4.4.4, 4.5, 5.4.2, 5.5.2

[2] Sean Anderson and Timothy Barfoot. Towards relative continuous-time SLAM. In *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013. 2.2.2, 2.2

[3] Sean Anderson and Timothy Barfoot. RANSAC for motion-distorted 3D visual sensors. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, Nov. 2013. 2.2.2, 2.2

[4] J. Artieda, J. Sebastian, P. Campoy, and et al. Viusal 3-d SLAM from UAVs. *Journal of Intelligent and Robotic Systems*, 55, 2009. 2.1.2

[5] H. Bay, A. Ess, T. Tuytelaars, and L. Gool. SURF: Speeded up robust features. *Computer Vision and Image Understanding*, 2008. 2.2.2

[6] Peter Biber and Wolfgang Straßer. The normal distributions transform: A new approach to laser scan matching. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2743–2748, 2003. 2.2.1, 2.2

[7] Michael Bosse and Robert Zlot. Continuous 3D scan-matching with a spinning 2d laser. In *IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 2009. 2.2.2, 2.2

[8] Michael Bosse, Robert Zlot, and Paul Flick. Zebedee: Design of a spring-mounted 3-D range sensor with application to mobile mapping. *IEEE Transactions on Robotics*, 28(5): 1104–1119, 2012. 2.2.2, 2.2

[9] Adam Bry, Charles Richter, Abraham Bachrach, and Nicholas Roy. Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments. *The International Journal of Robotics Research*, 34(7):969–1002, 2015. 8.1

[10] F. Caballero, L. Merino, J. Ferruz, and A. Ollero. Vision-based odometry and SLAM for medium and high altitude flying UAVs. *Journal of Intelligent and Robotic Systems*, 54 (1-3):137–161, 2009. 2.1.2, 2.1

[11] James B Campbell. *Introduction to remote sensing*. CRC Press, 2002. 2.2.1

[12] E Cheney and David Kincaid. *Numerical mathematics and computing*. Cengage Learning (6th Edition), 2007. 7.5

[13] J. Civera, O. Grasa, A. Davison, and et al. 1-point ransac for extended kalman filtering:

Application to real-time structure from motion and visual odometry. *Journal of Field Robotics*, 27(5), 2010. 2.1.2, 2.1

[14] Gianpaolo Conte and Patrick Doherty. Vision-based unmanned aerial vehicle navigation using geo-referenced information. *EURASIP Journal on Advances in Signal Processing*, 2009:10, 2009. 2.1.2, 2.1

[15] P. Corke, D. Strelow, and S. Singh. Omnidirectional visual odometry for a planetary rover. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sendai, Japan, Sept. 2004. 2.1.1, 2.1, 3.1

[16] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computation Geometry: Algorithms and Applications (3rd Edition)*. Springer, 2008. 3.4.3, 4.4.2, 4.5, 5.4.3, 5.5.1, 7.4.2

[17] F. Dellaert and M. Kaess. Square Root SAM: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Research*, 25(12): 1181–1204, 2006. 5.2.2

[18] Hang Dong and Timothy Barfoot. Lighting-invariant visual odometry using lidar intensity imagery and pose interpolation. In *The 7th International Conference on Field and Service Robots*, July 2012. 2.2.2, 2.2

[19] David Droeschel, Jorg Stuckler, and Sven Behnke. Local multi-resolution representation for 6D motion estimation and mapping with a continuously rotating 3D laser scanner. In *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, May 2014. 2.2.2

[20] I. Dryanovski, R. Valenti, and J. Xiao. Fast visual odometry and mapping from RGB-D data. In *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013. 2.1.3

[21] H. Durrant-Whyte and T. Balley. Simultaneous localization and mapping: Part i. *IEEE Robotics and Automation Magazine*, 13(2):99–110, 2006. 2.2.2

[22] H. Durrant-Whyte and T. Balley. Simultaneous localization and mapping: Part ii. *IEEE Robotics and Automation Magazine*, 13(3):108–117, 2006. 2.2.2

[23] Ben Eckart, Kihwan Kim, Alejandro Troccoli, Alonzo Kelly, and Jan Kautz. MLMD: Max-imum likelihood mixture decoupling for fast and accurate point cloud registration. In *IEEE International Conference on 3D Vision (3DV)*, Lyon, France, Oct. 2015. 2.2.1

[24] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision (ECCV)*, Zurich, Switzerland, Sept. 2014. 2.1.2, 2.1.3

[25] Jakob Engel, Jurgen Sturm, and Daniel Cremers. Semi-dense visual odometry for a monocular camera. In *IEEE International Conference on Computer Vision (ICCV)*, Sydney, Australia, Dec. 2013. 2.1.2, 2.1.3, 2.1, 3.1

[26] N. Engelhard, F. Endres, J. Hess, Jurgen Sturm, and Wolfram Burgard. Real-time 3D visual SLAM with a hand-held RGB-D camera. In *RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum*, Vasteras, Sweden, April 2011. 2.1.3

[27] Georgios D Evangelidis, Dionyssos Kounades-Bastian, Radu Horaud, and Emmanouil ZPsarakis. A generative model for the joint registration of multiple point sets. In *European Conference on Computer Vision (ECCV)*, Zurich, Switzerland, Sept. 2014. 2.2.1

[28] Maurice Fallon, Hordur Johannsson, and John Leonard. Efficient scene simulation for robust monte carlo localization using an rgb-d camera. In *IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, MN, May 2012. 8.1

[29] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, May 2014. 2.1.2, 2.1, 3.1, 3.4.1, 3.4.3, 5.4.3

[30] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In *Robotics: Science and Systems (RSS)*, Rome, Italy, July 2015. 5.3.2

[31] Francis Colas Franois Pomerleau and Roland Siegwart. A review of point cloud registration algorithms for mobile robotics. *Foundations and Trends in Robotics*, 4(1):1–104, 2015. 5.5

[32] N. Ganganath and H. Leung. Mobile robot localization using odometry and kinect sensor. In *IEEE International Conference on Emerging Signal Processing Applications (ESPA)*, Las Vegas, Nevada, Jan. 2012. 8.1

[33] A. Geiger, P. Lenz, C. Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research*, (32):1229–1235, 2013. 3.1, 4.3.1, 4.6.5, 5.2.1, 6

[34] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient large-scale stereo matching. In *Asian Conference on Computer Vision (ACCV)*, Queenstown, New Zealand, Nov. 2010. 6

[35] Andreas Geiger, Julius Ziegler, and Christoph Stiller. Stereoscan: Dense 3D reconstruction in real-time. In *IEEE Intelligent Vehicles Symposium (IV)*, Baden-Baden, Germany, June 2011. 2.1.1, 2.1

[36] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? The kitti vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, 2012. 3.1, 4.3.1, 4.6.5, 6

[37] R. Hartley and A. Zisserman. *Multiple View Geometry in computer vision*. Cambridge Press, 2000. 5.2.1

[38] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. New York, Cambridge University Press, 2004. 3.2, 3.4.1, 3.6, 4.4.3, 4.5, 7.4.2

[39] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using kinect-style depth cameras for dense 3D modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663, 2012. 2.1.3, 2.1, 2.1.4

[40] Dirk Holz and Sven Behnke. Mapping with micro aerial vehicles by registration of sparse 3d laser scans. In *The 13th International Conference on Intelligent Autonomous Systems (IAS)*, Padova, Italy, July 2014. 2.2.2

[41] Seungpyo Hong, Heedong Ko, and Jinwook Kim. VICP: Velocity updating iterative closest point algorithm. In *IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, Alaska, May 2010. 2.2.2, 2.2

[42] Radu Horaud and Fadi Dornaika. Hand-eye calibration. *The international journal of robotics research*, 14(3):195–210, 1995. 5.2.1

[43] Radu Horaud, Florence Forbes, Manuel Yguel, Guillaume Dewaele, and Jian Zhang. Rigid and articulated point registration with expectation conditional maximization. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 33(3):587–602, 2011. 2.2.1

[44] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012. 7.3.1

[45] A. Howard. Real-time stereo visual odometry for autonomous ground vehicles. In *IEEE International Conference on Intelligent Robots and Systems*, Nice, France, Sept 2008. 2.1.1, 2.1

[46] G. Hu, S. Huang, L. Zhao, A. Alempijevic, and G. Dissanayake. A robust RGB-D slam algorithm. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura, Portugal, Oct. 2012. 2.1, 2.1.4

[47] A. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. Visual odometry and mapping for autonomous flight using an RGB-D camera. In *International Symposium on Robotics Research (ISRR)*, Flagstaff, Arizona, Aug. 2011. 2.1.3, 2.1, 2.1.4, 3.6.1

[48] G. Huang, M. Kaess, and J.J. Leonard. Towards consistent visual-inertial navigation. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Hong Kong, June 2014. 2.1.2, 2.1, 5.3.1

[49] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, 24(6):1365–1378, 2008. 2.2.2, 5.2.2

[50] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *International Journal of Robotics Research*, 31(2):217–236, 2012. 1.1, 3.5

[51] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J.J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *International Journal of Robotics Research*, 31(2):217–236, 2012. 2.2.2

[52] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for RGB-D cameras. In *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013. 2.1.3, 2.1, 2.1.4, 3.4.1, 3.6.1

[53] B. Kitt, J. Rehder, A. Chambers, and et al. Monocular visual odometry using a planar road model to solve scale ambiguity. In *Proc. European Conference on Mobile Robots*, September 2011. 2.1.2, 2.1

[54] G. Klein and D. Murray. Parallel tracking amd mapping for small AR workspaces. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, Nara, Japan, Nov. 2007. 2.1.2, 2.1, 3.1

[55] Stefan Kohlbrecher, Oskar von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable slam system with full 3D motion estimation. In *IEEE International Symposium on Safety, Security, and Rescue Robotics*, Kyoto, Japan, September 2011. 2.2.2

[56] K. Konolige, M. Agrawal, and J. Sol. Large-scale visual odometry for rough terrain. *Robotics Research*, 66:201–212, 2011. 2.1.1, 2.1, 3.1

[57] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual-inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015. 2.1.2, 2.1, 5.3.2

[58] Mingyang Li and Anastasios I Mourikis. High-precision, consistent ekf-based visual–inertial odometry. *The International Journal of Robotics Research*, 32(6):690–711, 2013. 2.1.2, 2.1, 5.3.1

[59] Yangming Li and Edwin Olson. Structure tensors for general purpose LIDAR feature extraction. In *IEEE International Conference on Robotics and Automation*, Shanghai, China, May 9-13 2011. 4.4.1

[60] Victor Lu and John Hart. Multicore construction of k-d trees for high dimensional point data. In *International Conference on Advances in Big Data Analytics (ABDA)*, Las Vegas, NV, July 2014. 5.5.4

[61] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence*, Vancouver, Canada, Aug. 1981. 3.6, 5.8.1, 7.4.2

[62] M. Maimone, Y. Cheng, and L. Matthies. Two years of visual odometry on the mars exploration rovers. *Journal of Field Robotics*, 24(2):169–186, 2007. 2.1.1, 2.1

[63] Colin McManus, Winston Churchill, Ashley Napier, Ben Davis, and Paul Newman. Distraction suppression for vision-based pose estimation at city scales. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3762–3769, 2013. 7.5

[64] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *The AAAI Conference on Artificial Intelligence*, Edmonton, Canada, July 2002, pp. 99–110. 2.2.2

[65] Frank Moosmann and Christoph Stiller. Velodyne SLAM. In *IEEE Intelligent Vehicles Symposium (IV)*, Baden-Baden, Germany, June 2011. 2.2.2, 2.2

[66] Richard Murray and Shankar Sastry. *A mathematical introduction to robotic manipulation*. CRC Press, 1994. 3.4.1, 3.4.1, 4.4.3, 5.4.1, 5.4.2

[67] Richard Newcombe, Andrew Davison, Shahram Izadi, Pushmeet Kohli, Otmar Hilliges, Jamie Shotton, David Molyneaux, Steve Hodges, David Kim, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011. 2.1.3

[68] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. DTAM: Dense tracking and mapping in real-time. In *IEEE International Conference on Computer Vision (ICCV)*, Barcelona, Spain, Nov. 2011. 2.1.2, 2.1.3, 2.1, 3.1

[69] Matthias Nieuwenhuisen, David Droeschel, Marius Beul, , and Sven Behnke. Au-

tonomous navigation for micro aerial vehicles in complex gnss-denied environments. *Journal of Intelligent and Robotic Systems*, 2015. 8.1

[70] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry for ground vechicle applications. *Journal of Field Robotics*, 23(1):3–20, 2006. 2.1.1, 2.1

[71] J. Nocedal and S. Wright. *Numerical Optimization*. New York, Springer-Verlag, 2006. 5.4.2, 5.5.2, 5.7, 7.2, 7.4.2

[72] Navid Nourani-Vatani and Paulo Borges. Correlation-based visual odometry for ground vehicles. *Journal of Field Robotics*, 28(5), 2011. 2.1.2, 2.1

[73] A Nuchter, K. Lingemann, J. Hertzberg, and H. Surmann. 6D SLAM–3D mapping outdoor environments. *Journal of Field Robotics*, 24(8-9):699–722, 2007. 2.2.2

[74] Henri Nurminen, Anssi Ristimäki, Simo Ali-Löytty, and Robert Piché. Particle filter and smoother for indoor localization. In *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, Montbliard, France, Oct. 2013. 8.1

[75] Taragay Oskiper, Zhiwei Zhu, Supun Samarasekera, and Rakesh Kumar. Visual odometry system using multiple stereo cameras and inertial measurement unit. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Minneapolis, MN, June 2007. 2.1.1

[76] L. Paz, P. Pinies, and J. Tardos. Large-scale 6-DOF SLAM with stereo-in-hand. *IEEE Transactions on Robotics*, 2008. 2.1.1

[77] François Pomerleau, Francis Colas, Roland Siegwart, and Stéphane Magnenat. Comparing ICP variants on real-world data sets. *Autonomous Robots*, 34(3):133–148, 2013. 2.2.1, 2.2, 7.4.2

[78] A. Pretto, E. Menegatti, M. Bennewitz, and et al. A visual odometry framework robust to motion blur. In *IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 2009. 2.1.2, 2.1

[79] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. ROS: An open-source robot operating system. In *Workshop on Open Source Software (Collocated with ICRA 2009)*, Kobe, Japan, May 2009. 5.8.1

[80] Alessandro Redondi, Marco Chirico, Luca Borsani, Matteo Cesana, and Marco Tagliasacchi. An integrated system based on wireless sensor networks for patient monitoring, localization and tracking. *Ad Hoc Networks*, 11(1):39–53, 2013. 8.1

[81] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*, Quebec City, Canada, June 2001. 2.2.1, 2.2, 4.6.1

[82] D. Scaramuzza. Absolute scale in structure from motion from a single vehicle mounted camera by exploiting nonholonomic constraints. In *IEEE International Conference on Computer Vision*, Kyoto, Japan, Sept. 2009. 2.1.2, 2.1

[83] Davide Scaramuzza. 1-point-ransac structure from motion for vehicle-mounted cameras by exploiting non-holonomic constraints. *International Journal of Computer Vision*, 95: 7417, 2011. 2.1.2, 2.1, 3.1

[84] Sebastian Scherer, Joern Rehder, Supreeth Achar, Hugh Cover, Andrew Chambers, Stephen Nuske, and Sanjiv Singh. River mapping from a flying robot: state estimation, river detection, and obstacle mapping. *Autonomous Robots*, 32(5):1–26, May 2012. 2.2.2, 2.2

[85] Todor Stoyanov, Martin Magnusson, and Achim J Lilienthal. Point set registration through minimization of the l2 distance between 3d-ndt models. In *IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, Alaska, May 2010. 2.2.1

[86] J. Sturm, E. Bylow, C. Kerl, F. Kahl, and D. Cremer. Dense tracking and mapping with a quadrocopter. In *Unmanned Aerial Vehicle in Geomatics (UAV-g)*, Rostock, Germany, 2013. 2.1.3, 2.1, 2.1.4, 3.4.1

[87] Jean-Philippe Tardif, Yanis Pavlidis, and Kostas Daniilidis. Monocular visual odometry in urban environments using an omnidirectional camera. In *International Conference on Intelligent Robots and Systems (IROS)*, Nice, France, Sept. 2008. 2.1.1

[88] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. Cambridge, MA, The MIT Press, 2005. 2.2.2, 8.2.3

[89] Gian Tipaldi, Daniel Meyer-Delius, and Wolfram Burgard. Lifelong localization in changing environments. *The International Journal of Robotics Research*, 32(14):1662–1678, 2013. 8.1

[90] Chi Hay Tong and Timothy Barfoot. Gaussian process gauss-newton for 3D laser-based visual odometry. In *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013. 2.2.2, 2.2

[91] Lloyd Trefethen and David Bau III. *Numerical linear algebra*. SIAM, 1997. 7.2, 7.5

[92] Cihan Ulaş and Hakan Temeltaş. 3d multi-layered normal distribution transform for fast and long range scan matching. *Journal of Intelligent & Robotic Systems*, 71(1):85–108, 2013. 2.2.1, 2.2

[93] Ranjith Unnikrishnan and Martial Hebert. Fast extrinsic calibration of a laser rangefinder to a camera. Technical report, Technical Report #CMU-RI-TR-05-09, Robotics Institute, Carnegie Mellon University, 2005. 5.2.1

[94] George Vogiatzis and Carlos Hernandez. Video-based, real-time multi-view stereo. *Image and Vision Computing*, 29(7):434–441, 2011. 3.4.3, 5.4.3

[95] S Weiss, M Achtelik, S Lynen, M Achtelik, L Kneip, M Chli, and R Siegwart. Monocular vision for long-term micro aerial vehicle state estimation: A compendium. *Journal of Field Robotics*, 30(5):803–831, 2013. 2.1, 3.1

[96] T. Whelan, H. Johannsson, M. Kaess, J. Leonard, and J. McDonald. Robust real-time visual odometry for dense RGB-D mapping. In *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013. 2.1.3

[97] Ji Zhang and Sanjiv Singh. LOAM: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems Conference (RSS)*, Berkeley, CA, July 2014. 5.5, 5.5.1, 8.1

[98] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: Low-drift, robust, and

fast. In *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, May 2015. 7.4.2

[99] Ji Zhang and Sanjiv Singh. Enabling aggressive motion estimation at low-drift and accurate mapping in real-time. In *IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, May 2017. 8.1, 8.2.2, 8.3

[100] Ji Zhang, Michael Kaess, and Sanjiv Singh. Real-time depth enhanced monocular odometry. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Chicago, IL, Sept. 2014. 5.4, 8.1

[101] Ji Zhang, Michael Kaess, and Sanjiv Singh. On degeneracy of optimization-based state estimation problems. In *IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, May 2016. 5.7, 8.2.2

[102] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000. 5.2.1

[103] Robert Zlot and Michael Bosse. Efficient large-scale 3D mobile mapping and surface reconstruction of an underground mine. In *The 7th International Conference on Field and Service Robots*, July 2012. 2.2.2, 2.2