University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Kruno Lenac

SIMULTANEOUS MOBILE ROBOT LOCALIZATION AND THREE-DIMENSIONAL MODELING OF UNKNOWN COMPLEX ENVIRONMENTS IN REAL TIME

DOCTORAL THESIS

Zagreb, 2017

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Kruno Lenac

SIMULTANEOUS MOBILE ROBOT LOCALIZATION AND THREE-DIMENSIONAL MODELING OF UNKNOWN COMPLEX ENVIRONMENTS IN REAL TIME

DOCTORAL THESIS

Supervisor: Professor Ivan Petrović, PhD

Zagreb, 2017

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Kruno Lenac

ISTODOBNA LOKALIZACIJA MOBILNIH ROBOTA I TRODIMENZIONALNO MODELIRANJE NEPOZNATIH SLOŽENIH PROSTORA U STVARNOME VREMENU

DOKTORSKI RAD

Mentor: Prof. dr. sc. Ivan Petrović

Zagreb, 2017.

Doctoral thesis was written at the University of Zagreb, Faculty of Electrical Engineering and Computing, Departement of Control and Computer Engineering.

Supervisor: Professor Ivan Petrović, PhD

Thesis contains 147 pages

Thesis no.:

ABOUT THE SUPERVISOR

IVAN PETROVIĆ received B.Sc., M.Sc. and Ph.D. degrees in electrical engineering from the University of Zagreb, Faculty of Electrical Engineering and Computing (FER), Zagreb, Croatia, in 1983, 1989 and 1998, respectively.

For the first ten years after graduation he was with the Institute of Electrical Engineering of Končar Corporation in Zagreb, where he had been working as a research and development engineer for control and automation systems of electrical drives and industrial plants. From 1994 he has been working at the Department of Control and Computer Engineering at FER, where he is currently a Full Professor with tenure. He has actively participated as a collaborator or principal investigator on more than 30 national and 20 international scientific projects, where from them six are funded form FP7 and Horizon 2020 framework programmes. He is also co-director of the Centre of Research Excellence for Data Science and Cooperative Systems. He published more than 50 papers in scientific journals and more than 180 papers in proceedings of international conferences in the area of control engineering and automation applied to control mobile robots and vehicles, power systems, electromechanical systems and other technical systems.

Professor Petrović is a member of IEEE, Croatian Academy of Engineering (HATZ), chair of the Technical committee on Robotics of the International Federation of Automatic Control (IFAC), a permanent board member of the European Conference on Mobile Robots, an executive committee member of the Federation of International Robot-soccer Association (FIRA), and a founding member of the iSpace Laboratory Network. He is also a member of the Croatian Society for Communications, Computing, Electronics, Measurements and Control (KoREMA) and Editor-in-Chief of the Automatika journal. He received the award "Professor Vratislav Bedjanič" in Ljubljana for outstanding M.Sc. thesis in 1990 and silver medal "Josip Lončar" from FER for outstanding Ph.D. thesis in 1998. For scientific achievements he received the award "Rikard Podhorsky" from the Croatian Academy of Engineering (2008), "National Science Award of the Republic of Croatia" (2011), the gold plaque "Josip Lončar" (2013) and "Science Award" from FER (2015).

O MENTORU

IVAN PETROVIĆ diplomirao je, magistrirao i doktorirao u polju elektrotehnike na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva (FER), 1983., 1989. odnosno 1998. godine.

Prvih deset godina po završetku studija radio je na poslovima istraživanja i razvoja sustava upravljanja i automatizacije elektromotornih pogona i industrijskih postrojenja u Končar - Institutu za elektrotehniku. Od svibnja 1994. radi u Zavodu za automatiku i računalno inženjerstvo FER-a, gdje je sada redoviti profesor u trajnome zvanju. Sudjelovao je ili sudjeluje kao suradnik ili voditelj na više od 30 domaćih i 20 međunarodnih znanstvenih projekata, od čega šest projekata iz programa FP7 i Obzor 2020. Nadalje, suvoditelj je Znanstvenog centra izvrsnosti za znanost o podatcima i kooperativne sustave. Objavio je više od 50 znanstvenih radova u časopisima i više od 180 znanstvenih radova u zbornicima skupova u području automatskog upravljanja i estimacije s primjenom u upravljanju mobilnim robotima i vozilima te energetskim, elektromehaničkim i drugim tehničkim sustavima.

Prof. Petrović član je stručne udruge IEEE, Akademije tehničkih znanosti Hrvatske (HATZ), predsjednik tehničkog odbora za robotiku međunarodne udruge IFAC, stalni član upravnog tijela European Conference of Mobile Robots, član izvršnog odbora međunarodne udruge FIRA, suutemeljitelj međunarodne udruge „The iSpace Laboratory Network". Član je i upravnog odbora Hrvatskog društva za komunikacije, računarstvo, elektroniku, mjerenja i automatiku (KoREMA) te glavni i odgovorni urednik časopisa Automatika. Godine 1990. primio je u Ljubljani nagradu „Prof. dr. Vratislav Bedjanič" za posebno istaknuti magistarski rad, 1998. srebrnu plaketu "Josip Lončar" FER-a za posebno istaknutu doktorsku disertaciju, a za znanstvena je postignuća dobio 2008. godine nagradu „Rikard Podhorsky" Akademije tehničkih znanosti Hrvatske, 2011. godine „Državnu nagradu za znanost", 2013. godine zlatnu plaketu "Josip Lončar" FER-a te 2015. godine nagradu za znanost FER-a.

ABSTRACT

The two key prerequisites for autonomous navigation of any mobile robot are its capability to infer (1) what the environment around it looks like and (2) where it is in the environment. To infer *where it is* a robot localization algorithm is needed, while to infer what the environment looks like a mapping algorithm is needed. The difficulties with estimating the environment map and the robot location arise from the fact that they are strongly interrelated. For this reason the present thesis focuses on development of the Simultaneous Localization and Mapping (SLAM) algorithms which allow mobile robots to build the map of completely unknown environments, and at the same time estimate their location within. The main problems of today's state of the art SLAM algorithms are: (i) to find appropriate map representation, (ii), computation speed, (iii) to achieve long-term autonomy, and (iv) to distribute the SLAM tasks over several robots. In this thesis all four problems are addressed.

For the map representation, the algorithm is presented which allows modeling of 3D complex environments using as small as possible number of planar surface segments. Moreover, the algorithm also allows for fast update of the entire map when the SLAM optimization is completed. Computation speed is addressed through the derivation of the Exactly Sparse Delayed State Filter (ESDSF) on Lie groups (LG-ESDSF). LG-ESDSF represents the states on Lie groups, and performs filtering equations in the pertaining Lie algebra, which allows it to respect the geometry of the state space. Because of this LG-ESDSF achieves accuracy comparable to the state-of-the-art SLAM algorithms while maintaining high computation speeds. A solution for long-term SLAM is also presented in the context of LG-ESDSF. It detects which states have negligible information gain and removes them from the state space, but at the same time preserves the sparsity of the information matrix, thus allowing long-term real-time performance possible. Finally, the distribution of the SLAM tasks over several agents is solved by the introduction of the cooperative SLAM algorithm. The proposed algorithm allows each agent to compute only simple computing tasks (trajectory optimization in our case), and therefore they can have low computational power and still work in real-time, while the central server performs complex computing tasks (global map building in our case). Moreover, server collects information from all the robots and sends the update information based on data from one agent to the other agents. Agents then use this information to further increase their trajectory accuracy.

KEYWORDS: simultaneous localization and mapping, filtering SLAM, graph-based SLAM, exactly sparse delayed state filter, 3D planar map, Lie groups, cooperative SLAM, long-term

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

BA	Bunde Adjustment
BoW	Bag of Words
CLT	Chow Liu Tree
CS	Cloud Server
DoF	Degrees of Freedom
EIF	Extended Information Filter
EKF	Extended Kalman Filter
FOV	Field Of View
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LIDAR	Light Detection And Ranging
LMR	Local Map Registration
NDT	Normal Distributions Transform
PF	Particle Filter
PSS	Planar Surface Segments
RSSI	Received Signal Strength Indicator
SEIF	Sparse Extended Information Filter
SIFT	Scale Invariant Feature Transforms
SLAM	Simultaneous Localization And Mapping
SURF	Speeded Up Robust Feature
ToF	Time of flight
UAV	Unmanned Aerial Vehicle
VI	Visual Inertial

1

Introduction

Do or do not, there is no try.

Master Yoda

In this chapter, brief introduction and motivation behind the work done in this thesis is presented. We start with explanation why mobile robotics is important in general with the emphasis on the importance of the autonomy of mobile robots. Then, we explain what exactly the *simultaneous mobile robot localization and three-dimensional modeling of unknown complex environments in real time* means in the context of mobile robots and why it is essential for achieving full autonomy in completing any task. Main challenges in achieving solution to this problem are presented afterwards, followed with the discussion of various approaches. After this, original scientific contributions of the thesis are presented with more detailed elaborations. Finally outline and structure of this thesis is sketched, with short description and main aspects covered in each of the chapters.

1.1 Motivation and problem statement

Mobile robots, in the next decades, will transform our everyday lives, as well as industrial processes, similar to the impact that the Internet, cell phones and computers had in the past two decades. Starting from our homes, where they could help us in everyday chores, to search and rescue operations where they could replace us in dangerous tasks, and finishing with factories where, even nowadays, they increase safety, production speed, and performance. Accordingly, many globally relevant agencies and institutes, such as McKinsey Global Institute [2], World Economic Forum [3] and European Commission [4], have highlighted the significance of the field of autonomous, next-generation robotics, and its potential to transform life, business and the global economy. Therefore, the main challenges in mobile robotics in the coming decades will be how to allow these mobile robots to cooperate with humans in arbitrary settings and how to develop their autonomy to allow them to perform these complex tasks.

However, regardless of the given task and its complexity, every autonomous mobile robot must have at its disposal (i) a map of the environment, and (ii) its location within that map. Although the environment map can mean different things, in the context of this thesis map will refer to representing complex 3D environments in a way recognizable to the mobile robot. Complex environment does not necessarily mean environment made of complex

shaped objects, but also environments with moving objects and surfaces that are hard to detect using sensors, like reflective or texture-less surfaces. Since fully autonomous mobile robots should not rely on a pre-built environment map, they should be able to simultaneously build the map of the environment and localize within that map. This problem is known in mobile robotics as Simultaneous Localization and Mapping (SLAM) problem and its solution is one of the fundamental enabling technology for operation abilities of advanced, autonomous and cooperative robotic systems in a wide range of future applications and in diverse operating environments (on the ground, in the air, underwater). The special importance of the SLAM has also been highlighted by the European Commission in [4], as part of the Perception and Navigation sections, where SLAM was identified through the following challenges: (i) dynamic map building, (ii) cloud based localization and (iii) localization in dynamic environment.

Due to its importance, today many different solutions to the SLAM problem exist, however they all need to cope with the fundamental SLAM paradox: in order to build a map, robot location has to be known, while in order to estimate the robot location, the map needs to be known. Because of this, although conceptually identified, SLAM had been considered insolvable for many years. However, with the advancements in mathematics, probabilistic theory and scientific discoveries in robotics in general, the following crucial fact was discovered: *As the robot moves through the environment and records features from the environment, connections made between those measurements can be accurately established no matter the error in the absolute robot or feature poses. Moreover, this connections can then be used to increase global accuracy of all poses which leads to the convergence of the SLAM solution*. Once this was realised the first SLAM solution was soon presented and it marked one of the most important events in the history of mobile robotics.

However, early SLAM solutions suffered from various problems like inability to work in real time, poor map representations, slow convergence and high sensitivity to disturbances. Even today, many SLAM solutions suffer from one or more of these problems. The main reason is that SLAM combines the knowledge from large number of different fields in robotics, probably the largest of any other problem in mobile robotics, which requires large number of different algorithms to produce accurate results in the same time. If any one fails, the entire system is compromised. Nevertheless, modern SLAM solutions are capable of accurately localizing a mobile robot and map the environment in real time over long periods without losing accuracy more than the required threshold for the safe autonomous robot operation. They can be grouped based on many different characteristics, of which the main are, how they perform optimization of map and robot poses, what state parametrization they use, what sort of the map representation they have and how long they can operate in real time. In this thesis, all of this aspects are discussed and the scientific contributions are presented that achieve step forward in each of these categories. The resulting SLAM solution is capable of working fast in large-scale environments requiring low computational and memory resources, and can be run on multiple different robots at the same time.

1.1.1 *Original scientific contributions*

The original scientific contributions of this thesis deal with enabling the SLAM algorithm to build efficient map of the environment which is easy to share and process and to optimize the map and the robot poses in order to maximize their accuracy. Also, they allow SLAM to remain fast in the long-term and to be able to work simultaneously on heterogeneous agents. The contributions with short elaborations are given in the sequel.

- Method for building a global three-dimensional map of large environments in real time by connecting planar features segmented from the local maps.

  The developed method represents the environment with as small as possible number of planar surface segments. Planar segments were chosen since they are predominant in indoor and urban environments and they are much more compact features than points. This is especially important when dealing with large point clouds, like those generate by 3D LIDAR, that can have tens of thousands of points. Using planar segments allows much faster registration of point clouds and map update operations which is essential for SLAM. The pertaining algorithm uses probabilistic approach to merge planar segments from different local maps, which consist of planar segments extracted from one point cloud. This drastically reduces the number of features required to build the global environment map. Moreover, algorithm allows fast update of each global surface when the need arises due to the SLAM optimization of robot's trajectory.

- Algorithm for simultaneous localization and three-dimensional modeling of unknown complex environments based on exactly sparse delayed state filter implemented on Lie groups.

  This contribution presents a novel derivation of the Exactly Sparse Delayed State Filter (ESDSF) which is used for prediction and optimization of robot's trajectory and consequently the map associated with that trajectory. Although 10 years ago, state-of-the-art SLAM solutions relied on filter for optimization, today's approaches mainly use graph optimization approach which, in essence, means solving SLAM as non linear least squares problem. This was mainly due to the fact that filtering approaches relied on state orientation representation within Euclidean frameworks which does not represent the natural way of characterizing uncertainties and relations between the state vector elements. In contrast to filtering based approaches, graph optimization SLAM approaches relied more on using the insights of Lie groups and Lie algebras within the framework. By representing the states on Lie groups, and performing filtering equations in the pertaining Lie algebra, they were able to respect the geometry of the state space, thus achieving better estimation accuracy of both the mean and the covariance. In this contribution, ESDSF was also derived on the Lie groups (LG-ESDSF) and by doing so it was able to achieve state-of-the-art performance while requiring much less time for optimization than the graph optimization SLAM solutions.

- Method for allowing simultaneous localization and three-dimensional modeling of unknown complex environments based on exactly sparse delayed state filter to work

in real-time for long-term.

Once the mapping is complete, and robot continues to move in the same environment, new states are continuously added into the SLAM state-space. After some time this generally leads to slow computation speeds and renders SLAM unable to continue operating in real-time. The proposed method allows the SLAM algorithm based on ESDSF to continue working in large scale environments, even long after the mapping is completed. The main advantage is that the SLAM does not need to be stooped when the map is built and reinitialized when the need for optimization arises. Instead, it can continue to map new environments, if they become reachable, and optimize map and robot poses continuously. This is achieved by detecting which states added into ESDSF contain much of the already present information and removing those states without destroying the sparsity of the ESDSF information matrix. The preservation of sparsity is crucial since it is the main reason why ESDSF based SLAM can perform its calculations significantly faster than standard EIF or EKF based SLAM solutions.

- Algorithm for simultaneous localization of heterogeneous multi-agent system and three-dimensional modeling of unknown complex environments.

  This contribution uses all three previous contributions to create highly efficient SLAM solution and extends it by adding the ability to simultaneously work on heterogeneous agents that have low computational power. This is achieved by emigrating the complex computing operations from the agents and performing them on a standalone server which communicates with the agents using wireless connection. Agents still have the ability to build local maps and perform trajectory update, while the global map is built by the server. Thanks to the planar representation of local maps, they can be easily transferred wirelessly to the server due to their small size. The server receives local maps and agents trajectories and builds the global map the same way as the SLAM for a single agent does. Moreover, the server uses this information to send update information to one agent, based on the information from another agent. Cooperative SLAM system built this way is event triggered which makes it immune to synchronization issues and is also robust to agents, communication or server failures. In the case of an agent failure the entire system can continue functioning normally, except no new information will be sent by the failed agent. In the case of a complete server failure or communication failure each agent still has its trajectory and local map which it can use to navigate safely. If communication with the server is re-established the data will be synced and in time the system will continue to function as if no communication loss had occurred.

1.2 OUTLINE OF THE THESIS

The thesis consists of seven chapters; this introduction chapter being first of them. The second chapter serve as an introduction to the concepts of mapping and localization in mobile robotics. The third chapter gives an overview of the Simultaneous Localization and Mapping (SLAM) solution. The development of the SLAM problem solutions is presented, from the

derivation of the first successful solution to the current state-of-the-art algorithms. After that, the main components of a general SLAM system are presented and related work for each of these components is given. The chapters 4-6 deal with scientific contributions. Each chapter begins with the introduction to the problem with brief description of how it was solved. Then, detailed derivation of the contribution is presented and finally experimental results, proving successful implementation, are presented. Experiments are conducted on relevant publicly available datasets which contain real world data. Each chapter is concluded with the summary. In the last, seventh, chapter the conclusion and the ideas for possible future work are given. Hereafter, short content description of each chapter is given.

⋆ CHAPTER 2. This chapter deals with the basics of localization and mapping, introducing key concepts for both. It presents different aspects of pose and map representation and different localization methods that perceived SLAM solutions. These methods are based on a-priori knowledge about the environment in which the robot will move. They require either artificially placed beacons or previously built maps of the environment. This makes them accurate, but suitable only for specific scenarios and robot's that do not require autonomy.

⋆ CHAPTER 3. In this chapter, SLAM problem is discussed in detail. Two main components of each SLAM are presented, i.e. its back-end and its front-end. Afterwards, main parts of every SLAM front-end are explained. They include: (i) odometry used for the robot motion prediction, (ii) loop closing detection, used for generating pose constraints, and finally (iii) method for generating pose constraints used for optimization. For each part two main groups are described. One contains the algorithms based on visual sensors and the other contains algorithms based on laser sensors. For each group and for each part, basic functionality and usage in SLAM is explained and related work is given. The rest of the chapter deals with SLAM back-end. Two main groups are presented, one consists of SLAM back-end based on filtering and the other consists of SLAM back-ends based on graph optimization. For each group the relevant research and approaches are presented. However, since the contributions of these thesis are based on filtering approach it is covered in more detail.

⋆ CHAPTER 4. This chapter introduces complete SLAM solution based on the Exactly Sparse Delayed State Filter (ESDSF) and planar representation of the environment. First, introduction and detailed explanation of every ESDSF step is given, together with its computational complexity analysis. Then, the algorithm for segmenting 3D point clouds into local maps that consist of planar surface segments is presented. Afterwards, global map construction algorithm is explained in four sections. The first section discusses in details the representation of the global map, while the second section deals with the algorithm for merging the planar segments from different local maps into global planar surfaces. The third section explains the method for calculating parameters of the global planar surfaces and the forth section gives the procedure for efficiently updating the global map, after the SLAM trajectory optimization occurred. Finally, the extension of the SLAM solution with the active SLAM component, which allows the SLAM to influence the robot's movement, is presented. The chapter concludes with extensive experimental testing of all components using four different datasets.

⋆ CHAPTER 5. This chapter begins with the introduction of Lie groups and their main aspects required for the derivation of the ESDSF on Lie groups (LG-ESDSF). Then, for each step of the ESDSF: (i) augmentation, (ii) marginalization and (iii) update, its derivation using the Lie group theory is given in details. It is also carefully explained what the main differences of each step in comparison to the classic ESDSF are. The derivation concludes with the analysis of computational complexity of the LG-ESDSF. Afterwards a solution is presented which allows LG-ESDSF to work over long periods of time in the same environment after the mapping is complete. Main problems with this situation are explained and procedure of removing states that hold little new information without the loss of sparsity of the information matrix is covered in detail. The chapter concludes with the experimental results. For this purpose the LG-ESDSF is coupled with two state-of-the-art SLAM front-ends, one for 3D LIDARs and one for stereo cameras. It is then compared to two state-of-the-art SLAM algorithms and one state-of-the-art SLAM back-end, all based on graph optimization. Finally the long-term solution is tested. All experiments are conducted using two acclaimed and very different publicly available datasets: one recorded with unmanned aerial vehicle and the other with the ground vehicle.

⋆ CHAPTER 6. In this chapter, a cooperative SLAM algorithm is proposed. SLAM back-end is based o LG-ESDSF described in Chapter 5 and its front-end on the same planar based solution described in Chapter 3. The chapter begins with the introduction to the cooperative SLAM, and main components of the proposed solution are explained afterwards. Then, the separation of the global map building from the trajectory estimation is explained, which is crucial to the fast execution of cooperative SLAM, is explained. Next, the algorithm which allows the server computer to use information of one agent to improve the accuracy of another is presented. Finally, robustness of the proposed solution is analysed and the entire algorithm is tested. Datasets used for testing are the same as for the other experiments done in the thesis, however, they are modified to accommodate for the multi agent environment.

⋆ CHAPTER 7. This chapter concludes the thesis, gives the summary of scientific contributions and the outlook to the future breakthroughs in the SLAM solutions.

2

Localization and mapping in mobile robotics

In the field of mobile robotics changes are occurring every day and few things are constant. However, there are some facts that will remain the same regardless of the advancements and technological breakthroughs. One of them is that every autonomous mobile robot, regardless of its purpose, must be able to answer two key questions at any time:

1. Where is it in the environment?
2. What the environment around it looks like?

The answer to the first question is given by the localization algorithm and the answer to the second question is given by the mapping algorithm. In mobile robotics, localization refers to estimation of robot's pose in 2D (3 DoF) or 3D (6 DoF) with respect to some reference coordinate frame, while mapping stands for estimation and representation of environment using information from the robot's sensors. The main problem with answering these questions is the fact that it is impossible to answer them separately. Mobile robot cannot determine its location without the map and it cannot build the map without knowing its location.

Researchers and engineers have tried to solve this problem over the past several decades continuously improving the accuracy and robustness of mobile robots localization and mapping algorithms. Due to the complexity of the problem itself, even today we cannot say that it has been completely solved. However today's state-of-the-art methods allow mobile robots to continuously and autonomously complete tasks in large and complex environments over long periods.

2.1 Map representation

Regardless of the technique used to build the map and its intended purpose, every map consists of features and landmarks. Feature is any part of the map whose pattern is different than that of its neighbours. Features that are used for localization are referred to as landmarks. Depending on the way features are represented in a map we can separate maps into three main groups: metric maps, topological maps and semantic maps, Fig. (2.1).

In metric maps, with a given distance metric, distance can be calculated between any two features in a map. Depending on the way features are represented, metric maps can be divided into continuous metric maps and discrete metric maps. Continuous metric maps

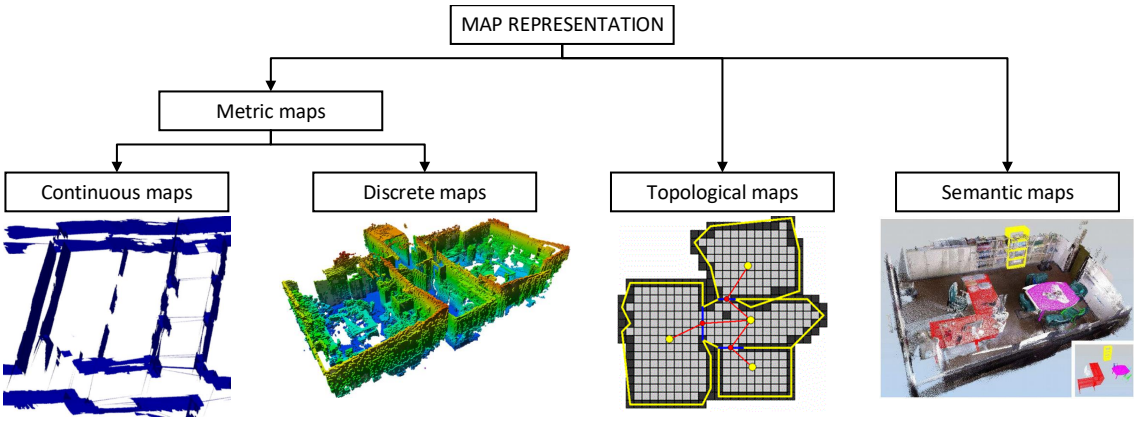

Figure 2.1: Different map representations in robotics.

consist of geometric features like lines, points and planes. Discrete metric maps decompose the environment into cells. Each cell has a predefined size which is determined by the map's maximum resolution. The most commonly used cells are squared and are referred to as grids in a 2D map and voxels in a 3D map.

Topological maps consist of nodes and edges that connect them. Each node represents a distinct part of the environment and can be as small as a single feature. However, in most cases, nodes represent metric maps, and edges between nodes mean that metric maps represented by those nodes are directly accessible from one another. The good example of a topological map would be a building with several floors. Each floor would represent one node in a topological map and nodes representing neighbouring floors would be connected with edges.

Semantic maps in robotics are most commonly combined with metric and/or topological map representation. Semantic maps add higher abstraction level of the environment which allow mobile robots to perceive environment not just as a set of features, but as a set of objects, like doors, walls, cars, trees, etc. Such environment representation allows much easier human-robot interaction and more accurate localization. However, semantic representation is computationally very demanding and there is still lack of algorithms that could accurately classify more complex objects. A survey on semantic mapping can be found in [5]

More details on different map representations can be found in [6], while solutions that use different combinations of mapping approaches are surveyed in [7].

2.2 POSE REPRESENTATION

Pose of a mobile robot consists of rotation and position expressed in relation to some fixed reference frame. Pose can be defined in 2D and 3D. In the case of a 2D pose a rotation is defined by a single angle and a position is defined by two coordinates. In the remainder of this thesis we will concentrate only on 3D pose representation. Although position representation is straightforward, four major 3D orientation representations exist: Euler angles, Euler axis, quaternions and rotation matrices. Here, only main properties of each representation are given, while some more details are presented later in the thesis.

The Euler angles are the three angles of rotation introduced by Leonhard Euler which

are used to describe the rotation of the robot's coordinate frames with respect to some fixed coordinate frame. Euler angles are most commonly denoted as ϕ, θ and ψ for rotations around x, y and z axes, respectively. The rotations can be defined in extrinsic and intrinsic way. Intrinsic rotations represent rotation of a mobile robot around its own coordinate frame, while extrinsic rotations represent rotation of o mobile robot around the coordinate axes of a fixed frame. Euler angles are most straightforward way to express rotation and they are easiest to visualize, but they suffer from a well known problem called gimbal-lock.

The Euler axis representation of rotation is based on Euler's rotation theorem, which says that any rotation or sequence of rotations of a rigid body in 3D space is equivalent to a pure rotation about a single fixed axis. Euler axis θ is defined as:

$$\theta = \alpha e \tag{2.1}$$

where e represents unit vector that defines the fixed axis around which the rotation is being made, while scalar α defines the amount of rotation in radians. Since e is a unit vector, it is defined only by two coordinates. If we want to rotate vector $v = [v_x\ v_y\ v_z]$ around the axis of rotation defined by the unit vector e for an angle α, the resulting vector v' would be

$$v' = \cos(\alpha)v + \sin\alpha v(e \times v) + (1 - \cos\alpha)(e \cdot v)e \tag{2.2}$$

Quaternions have first been described by Irish mathematician William Rowan Hamilton. They were developed as a number system that extends the complex numbers. In general quaternions are represented as

$$q = a + v \tag{2.3}$$

where a is a scalar part and $v = [bi + cj + dk]$ is a vector part. However, when we normalize quaternion and get unit quaternion, these unit quaternions can be used to represent orientation. In this case they are referred to as orientation quaternions. Orientation quaternions also have the basis in Euler's theorem, but they encode orientation differently. Orientation quaternion describing orientation around axis defined by unit vector e by angle ϕ is defined as:

$$q = \cos\left(\frac{\theta}{2}\right) + e\sin\left(\frac{\theta}{2}\right) \tag{2.4}$$

In order to rotate vector $p = [p_x,\ p_y\ p_z]$ using quaternion q we need to evaluate conjugation of p by q

$$p' = qpq^{-1} \tag{2.5}$$

Compared to Euler angles, quaternions are easier to compose and they avoid the gimbal-lock problem and compared to rotation matrices they are more compact and numerically stable.

Rotation matrices are 3×3 matrices whose columns or rows (depending on the convention) are three unit vectors (u, v and w) that form base of a coordinate frame with new orientation

$$R = \begin{bmatrix} u_x v_x w_x \\ u_y v_y w_y \\ u_z v_z w_z \end{bmatrix} \tag{2.6}$$

Although rotation matrix has 9 elements it has only 3 degrees of freedom, as dictated by the Euler's rotation theorem. Because of this, rotation matrix has the following properties:

1. R is real, orthogonal matrix
2. Eigenvalues of R are $\{1,\ \cos\theta + i\sin\theta,\ \cos\theta - i\sin\theta\}$
3. Determinant of R is $|R| = 1$
4. Trace of R is $\mathrm{Tr}(R) = 1 + 2\cos(\theta)$
5. Inverse of R is equal to its transpose $R^{-1} = R^T$

Since the set of all $n \times n$ orthogonal matrices with determinant 1 forms a group known as special orthogonal group $\mathsf{SO}(n)$, rotation matrices form special orthogonal group known as $\mathsf{SO}(3)$.

2.3 LOCALIZATION METHODS

The main difference between various localization methods is in the amount of required prior knowledge about the environment. The more knowledge is required the simpler the method. However, amount of required prior knowledge negatively effects the level of autonomy that the mobile robot can achieve.

Based on the level of autonomy we can divide all localization methods in three main groups:

1. The simplest and least autonomous methods rely on artificial beacons placed in the environment.
2. Higher degree of autonomy is achieved when the localization method requires only the environment map without any artificially placed beacons.
3. The highest level of autonomy is achieved when neither map nor beacons are required.

In the next two sections, the introduction and brief explanation for the first two groups will be given, while the remainder of the thesis will focus on the third group.

The main property that the first and the second group have in common is that they do not estimate environment map, but instead use given map and/or artificially placed information in the environment to estimate location of the mobile robot.

2.3.1 *Localization based on artificially placed beacons*

Localization techniques based on artificial beacons can be divided into two main groups depending on where actual localization algorithm is being performed. First group consists of localization methods that rely o active beacons to send information used for localization in the environment. The data is then intercepted by the receiver on a mobile robot and used to calculate its location. Most well known system of this type is Global Positioning System (GPS) [8]. GPS consists of satellites in Earth's orbit which constantly send localization data. Data are received by the GPS receiver placed on a robot which calculates robot's location on the Earth's surface. Although, when developed, GPS was the first system of this type, today we have similar systems developed by the European Union (EU) called GALILEO and by

the Russian Federation called Global Navigation Satellite System (GLONASS). CRICKET system [9] developed at MIT works by the same principle as GPS, i.e. uses a network of emitters and receivers to determine the robot location. However, in contrast to GPS, CRICKET works on a much smaller scale and is cheap and easy to setup in many different environments. CRICKET system is also scalable and can work with different number of emitters, with more emitters increasing its precision. One more similar approach is used in [10] where emitters send out RF signal, and the receiver on the mobile robot triangulates its position using Time of Flight (ToF) principle. Receiver can accurately calculate ToF thanks to its implementation using high performance multi core DSP processors. Approach similar to CRICKET system is also presented in [11] in which a sensor network of Zigbee [12] devices is used as emitters, and the receiver on the robot measures its location using the Received Signal Strength Indicator (RSSI) method. RSSI works by comparing how much of the original signal strength was lost while the signal was travelling from emitter to receiver.

Another major group of localization methods that use artificial beacons consists of methods that require mobile robot to be equipped only with the recognizable marker. Marker is then identified by several beacons placed in the environment, this information is shared between beacons and location is estimated. Usually, location information is then wirelessly transmitted to the mobile robot. Today two mostly used systems of such type are VICON and Optitrack motion capture systems. They both use IR emitters and specially designed markers which reflect IR light back to the receiver placed in each emitter. Using ToF principle, distance of the marker from each emitter is calculated and final robot location is triangulated on a server connected with each beacon. Besides specifically designed motion capture systems there are a lot solutions that use standard sensors like cameras [13] to create a sensors network used to localize a robot. In these cases markers are designed specifically to accommodate characteristics of a chosen sensor network in order to allow maximum visibility and ease of detection.

All localization methods based on artificial beacons, although easy to implement and highly accurate, have serious limitations which prevents them from replacing localization methods based on different principles. The most important limitation is the requirement to cover the entire area where the robot is going to move, with enough beacons to allow accurate localization. The best example of such problem is GPS, where often its signal in urban environments and under some weather conditions becomes too weak for accurate pose estimate. The similar problems occur when using systems like VIKON and Optitrack. The easiest solution to the localization problem with such systems would be to place more beacons to cover larger areas. However, the main disadvantage of such solution is its price. Although not nearly expensive as launching new satellites, VIKON, Optitrack and beacons used in other methods become very expensive when required in larger areas. Moreover, it is often not possible to place beacons in the required areas, for example when robot is moving through urban or forest environments. One more problem with those methods is the requirement for precise pose estimation between placed beacons, which is solved using specifically designed calibration protocols. Although, in theory system will require only initial calibration, any change in the environment or addition of new emitters will require the repeat of the process in order to maintain the accuracy.

For these reasons localization methods based on artificially placed beacons, although irre-

placeable in some applications, can never become universal localization methods. However, in some cases their limitations are abbreviated by combining them with other localization methods. The main idea behind this approach is that other localization methods will work accurately enough while the beacon signal strength is too low for localization. Once the beacon signal becomes available again, it will correct the accumulated error.

2.3.2 *Localization based on known map*

Localization techniques that use existing map all work on the same principle. They compare current measurements taken from the robot sensors to the existing map in the memory. Environment map stored in the memory is referred to as the global map, while the map constructed from the robot measurements is known as local map. Once the best fit between the local map and the global map is calculated, a pose is estimated between the local reference frame of the local map and the global reference frame of the global map. The main difference between these methods refers to features used to find the fit between the maps and the sensors used to build both maps. The main problem with these methods is the requirement to build the global map prior to localization.

Global maps can be built by the same sensors setup that is going to be used for robot localization by entirely different combination of sensors. When the robot is building the global map, usually, only measurements are recorded from the environment and then the global map is built offline. In the process of constructing a global map, pose that the robot had while taking measurements must be known. This can be solved by using some slow accurate matching algorithm that will match consecutive measurements to determine their relative location. It is a viable solution in this scenario because while constructing the global map, accuracy is much more important than the time required. Another solution to estimating robot location for offline map construction is to use beacon based localization. Since the map is built offline, one can move beacons from one section to the other, thus drastically reducing the cost. After the measurement locations are estimated beacons can be completely removed, so there is no need for multiple calibrations. After the global map is built, features used for the localization are extracted and stored in a way so they can be matched fast with features from the local maps. Global map can also be built using different sensors than the ones robot will use. For example we can use high accuracy offline 3D laser to build global map and then equip robots with stereo cameras or even 2D lasers depending on the requirements.

Work presented in [14] builds a global map from range measurements. It than reduces the global map by determining which areas are reachable by the used robot. After that features are extracted from range measurements by grouping individual points and classifying groups by their median range value. The features extracted from the global map are then compared to the features from the local map and current location is determined. In [15] authors present a solution that allows robot localization even in hand drawn 2D maps built by non-expert user. The approach is based on Monte Carlo localization (MC) [16], with two extensions. First, they augment state-space of the robot with an additional variable that represents local deformation of the hand drawn map. Second, they localize the robot in the pixel coordinate frame of the map, instead of the world coordinate frame. The authors of work presented in

[17] describe a method which uses textured occupancy grid map. They have demonstrated how a textured occupancy grid map can be combined with an extremely simple monocular localization algorithm to produce a viable localization solution comparable to localization results achieved when using a laser. Added benefit is that textured occupancy grid map can be used for localization by humans, cameras and lasers. Solution presented in [18] is a more general probabilistic solution which can work with different features extracted from camera image, laser range image or even sonars. It performs an efficient global search of the pose space that guarantees that the best position is found according to the probabilistic map agreement measure.

Although localization based on known map offers much more flexibility than localization based on beacons, it is still very limiting regarding the autonomy perspective. There are very few solutions that can cope with changing maps and there is still a requirement to enter and measure the environment prior to letting the robot operate in it. The only truly autonomous approach is to let the mobile robot to build the map itself first time it enters the environment.

2.3.3 *Localization and map building in unknown environments*

As stated before, when building the map at the same time while performing the localization, the main problem is that you do not know what do do first. You need a map to localize a robot within and you need pose to build a map. This is why such solutions were presumed unviable for the long time in mobile robotics. However some 20 years ago first solutions to the problem were presented. Since then numerous solutions have been presented and all are known under the same name: Simultaneous Localization And Mapping (SLAM). SLAM solutions will be in the focus of the remainder of the thesis with the next chapter providing introduction to SLAM basics.

3

Simultaneous Localization And Mapping

In this chapter Simultaneous Localization And Mapping (SLAM) algorithms will be described. First, the problem definition will be given, following with the overview of the SLAM development over the last two decades. Afterwards, the main components of every SLAM solution will be explained.

3.1 PROBLEM DEFINITION

As opposed to the problem described in the previous chapter, where only the robot location was estimated, in SLAM we need to estimate both the robot location and the location of all map landmarks. The SLAM problem can be represented as a proposal distribution

$$P(X_k, m|\Omega_{1\dots i}, z_{1\dots j}) \tag{3.1}$$

where X_k represents robot pose at step k, m represents all map landmarks, $z_{1\dots j}$ represents measurements recorded up to step k and $\Omega_{1\dots i}$ represents odometry inputs up to step k. More information about odometry can be found in Sec. 3.3.2. Notice that index j is different than index k since number of measurements does not need to be the same as the number of odometry inputs.

In the general form, the SLAM problem is unobservable since there is no known starting point, so the problem statement (3.1) is always written as

$$P(X_k, m|\Omega_{1\dots i}, z_{1\dots j}, X_1) \tag{3.2}$$

where X_1 represents initial starting point of the robot. It is not important what the exact value of X_1 is, only that it is fixed. The observability of SLAM will not be further discussed in the present thesis, however, more detailed analysis of SLAM observability and convergence can be found in [19].

Regardless of the SLAM solution, in order to estimate P at time step k, we always need to define a measurement model and a motion model. Motion model is used only to estimate new robot pose based on previous robot poses and odometry inputs. Motion model is defined as:

$$P(X_k|X_{1\dots k-1}, \Omega_k)\,. \tag{3.3}$$

If we assume that the robot motion between pose X_i to pose X_{i+1} acts as a first order Markov chain, we can write (3.3) as

$$P(X_k|X_{k-1}, u_k)\,. \tag{3.4}$$

Measurement model is defined as

$$P(z_k|X_k, m)\,. \tag{3.5}$$

Measurement model is used to estimate predicted value of real world measurement based on the current robot pose X_k, and poses of map landmarks m. The difference between the measured and the predicted value is used to optimize both, landmark and robot poses.

As can be seen from the measurement model (3.5), measurement probability is correlated to both robot pose and map landmarks. However, map landmarks and robot pose are also correlated, therefore we have the following inequality

$$P(X_k, m|z_k) \neq P(X_k|z_k)P(m|z_k)\,. \tag{3.6}$$

The absolute poses of the map landmarks have two sources of uncertainty, one comes from the sensor itself and landmark extraction algorithm, and the other is from the uncertainty of the robot pose at the time when the measurement was taken. The inequality (3.6) exists because of the latter. This means that we cannot estimate the poses of map landmarks separately from the robot pose. Moreover, since the uncertainty of the robot pose increases with every new pose estimated, so does the uncertainty of the map landmarks extracted from those poses. At first, it would seem that due to this error accumulation in both landmarks and robot poses, the solution to the (3.2) diverges in time. However the SLAM problem is solvable due to the following observation: *the accuracy of relative poses between landmarks is unaffected by the accuracies of landmark global poses and the accuracy of the robot global pose*. Figure 3.1 shows an example of four robot poses and map landmarks extracted from measurements taken at those poses. Landmark m^i was extracted from measurements z_{k-2} and z_{k-1}, while landmark m^j was extracted from z_{k-1} and z_k. Although absolute poses of landmarks m^i and m^j can have large errors, their relative pose can be calculated accurately, since they are both extracted from the same measurement z_{k-1}. In probabilistic term this means that although probability density of $P(m^i)$ and $P(m^j)$ is dispersed, probability density of $P(m^i, m^j)$ is highly concentrated. Moreover, when m^j is again extracted from z_k, it is matched with m^j extracted from z_{k-1} and this information can be used for correcting both accuracy of landmark pose m^i and robot pose X_k. Consequently, as all map landmarks extracted from the same measurement are correlated, their poses are also updated. As the robot continues to move and new landmarks are extracted, they are matched with previously extracted landmarks, and both poses of all map landmarks and the robot pose are corrected.

The SLAM problem can best be visualised using a set of springs. All map landmarks and robot poses are connected with each other using springs with variable elasticity coefficient. When the robot takes new measurement and new relative poses between landmarks are estimated, springs connecting those landmarks become stiffer, as do all other springs in the system. However, springs that connect landmarks that are farther away are less effected. As the robot continues to take new measurements, springs become more and more stiff, and ideally eventually lose all elasticity. In reality this will never happen, but we can say that the more stiff the springs are, the more certain we are in the pose of every landmark. All SLAM solutions try to do just that, minimize the uncertainty of map and robot poses. However they do this in many different ways. In the next chapter, development of different

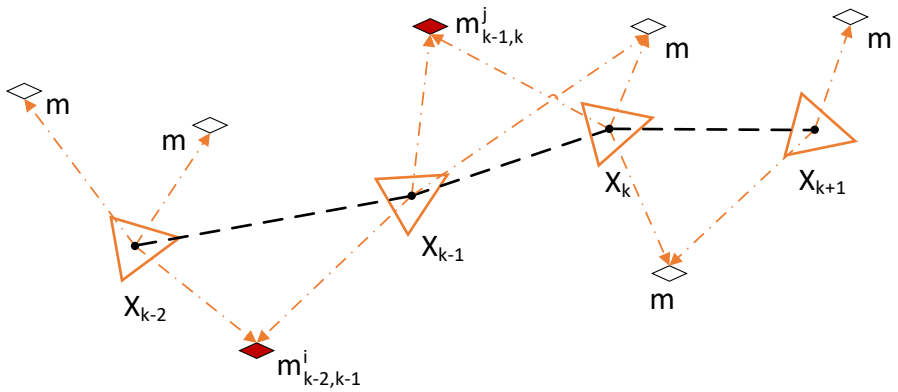

Figure 3.1: SLAM problem.

apporaches to solving the SLAM problem will be presented, and more details about more relevant ones will be given. For more information on the fundamentals of the SLAM the reader is referred to [20, 21].

3.2 DEVELOPMENT OF SLAM SOLUTIONS

The first formulation of the SLAM problem was in 1986. at the robotic conference in San Francisco. There, problems regarding simultaneous localization and mapping were recognized and many researchers started to work on finding the theoretical solution. After a few years many came to the conclusion that solution to the problem would require computation power unavailable at the time. Moreover, many also concluded that errors which occur in landmarks and robot poses will not converge with time, but would instead diverge, so they rendered the entire solution infeasible. Because of this conclusions, work on the problem solution was abandoned. However, after some time, it was realised that if estimating the robot pose and poses of map landmarks can be realised as a single estimation problem it does converge to a single solution. Most importantly, it was discovered that the number of correlations between map landmarks should not be minimised, but maximized instead in order to achieve better accuracy. Finally, at the International symposium on robotics held in 1995. SLAM acronym was formulated and first converging solution was presented. It marked a new age in robotics and opened doors to more advanced autonomous robotic systems.

From the first solution to the SLAM problem until today, a large number of different SLAM algorithms have been presented [22]. In general, any SLAM algorithm can be divided in two main parts: (i) the SLAM *front-end* and (ii) the SLAM *back-end*. The SLAM front-end is responsible for dealing with sensor data abstractions, e.g., extracting features and pose constraints, while the SLAM back-end is responsible for estimation and optimization of both robot and map landmarks poses based on the SLAM front-end pose constraints.

Main differences between SLAM front-ends comes from the sensor configuration (mono/stereo, color/greyscale cameras, 2D/3D LIDARs, radars etc.) and from the environment (air, water, forest, urban etc.) they were developed for. More details about different sensor configurations and applicable SLAM front-ends can be found in Sec. 3.3. Besides this, SLAM front-ends can further be divided based on two aspects. First important aspect

is whether or not they can successfully function in dynamic environments. Most SLAM front-ends completely disregard the possibility of moving objects in the scene and assume a completely static scenery. However, several works have addressed moving objects and taken them into account during map and location estimation [23, 24, 25], thus allowing for more robust performance when working in dynamic environments. Second differentiating aspect between SLAM front-ends comes from their ability to influence the robot motion. Although in general, SLAM algorithms are passive and do not influence robot navigation, some SLAM algorithms address this fact in order to maximize location and map accuracy, and are referred to as active SLAM algorithms [26, 27, 28, 29].

SLAM back-ends can be divided in two main groups depending on the approach to the state optimization: (i) filtering based (e.g. [30]) and (ii) graph-optimization based SLAM back-ends (e.g. [31]). Furthermore, filtering based SLAM back-ends can be divided into two additional groups based on the states used for the map and location estimation. The first group are the feature-based SLAM back-ends that estimate the current robot pose and poses of extracted map landmarks. Given that, map landmarks and the robot location are correlated and must be updated in the same step. The second group of filtering based approaches is called pose graph SLAM back-ends. These approaches estimate a discrete robot trajectory, while map landmarks are correlated only to one of the discrete trajectory states. The result is that map landmarks are no longer correlated to each other, but only to a single trajectory state which then allows trajectory estimation independent of the environment map estimation.

Even though the graph optimization approach to SLAM was known and well defined in the early stages of SLAM development, first SLAM solutions were nevertheless based on the filtering approach. The main reason behind this, at the time, was the inability to compute graph optimization in the time required for successful SLAM operation. The first filtering based SLAM solutions utilized the extended Kalman filter (EKF); however, issues were detected stemming from the linearization of both process and measurement model and from the increasing number of states when new landmarks are extracted and added as new filter states. Through time, many efficient implementations of the EKF-SLAM were presented, such as [32, 33, 34, 35, 36, 37, 38, 39], but the core problems remained. The first fundamentally different approach to EKF-SLAM was presented in the form of a particle filter (PF) SLAM, dubbed Fast-SLAM in [40]. The main advantage of the PF is that there is no need to linearize the system model, while the main drawback is high dimensionality of the state space. This was solved in Fast-SLAM by applying Rao-Blackwell marginalization. The improved version of the Fast-SLAM, dubbed Fast-SLAM 2.0, was later presented in [41].

In order to solve the issue of high dimensionality of the EKF-SLAM state-space, researches turned to the information form of the EKF called the extended information filter (EIF). The main advantage of using EIF in SLAM is that with a large numbers of landmarks in the state-space, most of the off-diagonal elements of the information matrix are close to zero. One of the first successful solutions for the EIF-SLAM was presented in [1] and more were presented thereafter in [42, 43]. However, disregarding elements that are close to zero leads to inevitable introduction of estimation errors, which in most cases for a well designed system are small, but still do exist. In [44] authors presented a solution to

the information-form SLAM that has an exactly sparse information matrix and thus no approximation error occurs. Therein, they named the developed information filter the exactly sparse delayed state filter (ESDSF).

The sparsity of the SLAM matrix was also a key insight that allowed developing of new direct linear solvers for the SLAM problem using graph optimization techniques, such as in [45]. New solvers allowed the refinement process to complete tens of times faster than before, which opened a whole new research area for SLAM algorithms. One of the first successful alternatives to filtering approaches, dubbed $\sqrt{\text{SLAM}}$, was presented in [46]. $\sqrt{\text{SLAM}}$ used a smoothing approach to solve SLAM and achieved better performance in both computation time and accuracy than the contemporary existing EKF-SLAM solutions. In [47] authors presented a method for optimizing large pose graphs called the sparse pose adjustment (SPA). SPA is similar to $\sqrt{\text{SLAM}}$, but with the main differences in (i) the efficient construction of the linear subproblem, by employing ordered data structures, and (ii) in using the Levenberg–Marquardt (LM) algorithm instead of the nonlinear least squares. A graph optimization solution was presented in [48] which used an efficient version of the sparse bundle adjustment (SBA). Therein, relations among cameras are also sparse and, by combining the proposed method with direct sparse Cholesky solvers, authors outperformed the standard SBA implementations. A stereo SLAM solution, named S-PTAM, was presented in [49] and used a parallel estimation process of the map and robot poses, thus enabling fast computation of robot location, while building map and refining the graph in the background. S-PTAM uses the LM optimization for refinement, while binary features are used for describing visual point landmarks. A graph based SLAM solution was also presented in [50], where to speed up computation and allow execution in large scale environments, authors divided the global graph into subgraphs optimized independently using the LM algorithm. Subgraphs are then matched and combined into global graph using loopy belief propagation algorithm called large scale relative similarity averaging.

In parallel to various complete SLAM solutions that used graph optimization and comprised of both the front-end and the back-end, researchers started to develop universal graph optimization solutions. These solutions can be used as SLAM back-ends for trajectory and map optimization, as well as for any optimization problem that can be formulated using a graph structure. Currently, there exists several such solutions including GTSAM of [51], Ceres of [52] and SLAM++ of [53], but the two solutions most commonly used in combination with different SLAM front-ends are iSAM of [54] and g^{2}o of [55]. iSAM uses a fast and incremental QR matrix factorization. By updating only the QR factorization of the sparse smoothing information matrix, it recalculates only those matrix elements that change drastically, thus increasing the computation speed. Another positive aspect of such matrix factorization is an easy and fast access to estimation uncertainties. Upgrade of the iSAM, called iSAM2, was presented in [56] which introduced a new data structure called the Bayes tree.

Of all the aforementioned solutions, g^{2}o is the most general optimization framework for nonlinear least squares problems that can be represented as a graph. It was developed to utilize sparse connectivity in the graph, and also to take the advantage of the special structures that occur in the graph when being built by specific algorithms like the graph SLAM or bundle adjustment (BA) SLAM. As a graph optimization solution, g^{2}o is highly

efficient and computationally fast thanks to using advanced methods for solving sparse systems and advanced features of modern processors as well as maximally optimizing processor memory and cache usage. Similarly to the other solutions, g^{2}o is also publicly available, but currently enjoys better community support and offers a large database of tutorials and examples. Therefore, it is not surprising that the two current state-of-the-art visual SLAM solutions, namely ORB-SLAM and LSD-SLAM, both use g^{2}o as their SLAM back-end.

LSD-SLAM was first presented for mono-cameras in [57] and afterwards a stereo-solution in [58] was introduced. It employs a direct and featureless method which minimizes photometric errors between images in order to estimate the pose. Main novelties included were direct tracking method which operated on $\mathsf{Sim}(3)$ and probabilistic solution to include the effect of noisy depth values into tracking. ORB-SLAM was also first introduced for mono cameras in [59], and later for stereo and RGB-D cameras in [60]. It uses ORB features for mapping, loop closing and tracking, and employs covisibility graph and survival of the fittest strategy to allow for real-time execution over long periods of time in large-scale environments.

The next section describes in more details key aspects of SLAM front-ends, while the final section of this chapter gives details of different SLAM back-ends.

3.3 SLAM FRONT-END

As stated in the previous section, SLAM front-end is responsible for processing sensor data and generating pose constraints used in the SLAM back-end to estimate robot and landmark poses. In this section, first most commonly used sensor setups in SLAM systems today will be described. Then odometry algorithms based on those sensor setups and used to evaluate SLAM motion model will be discussed. After this, algorithms for loop closing, key component of every SLAM front-end, will be presented and in the final section special type of SLAM front-ends that can influence robot motion in order to increase accuracy will be described.

3.3.1 *Commonly used sensor setups*

Although today there are a lot of SLAM front-ends for many different kinds of sensor setups, most commonly used are mono grayscale camera in combination with inertial measurement unit (IMU), two grayscale cameras in stereo configuration, 3D light detection and ranging (LIDAR) sensors and depth cameras in combination with RGB camera usually called RGBD sensor. These four sensor setups are shown in Fig. 3.2.

The main problem when using mono cameras for SLAM is that it is not possible to estimate metric scale of the poses. To solve this, mono camera is coupled with the IMU sensor whose accelerometers allow scale estimation, while data from the mono camera ensures that the error introduced by the accelerometers remains low. One of the best mono SLAM solutions that use IMU is presented in [61] which uses ORB-SLAM as a basis. It allows constant time local Bundle Adjustment (BA), and by not marginalizing past states, it is able to reuse them. Its back-end uses a lightweight pose-graph optimization, followed by

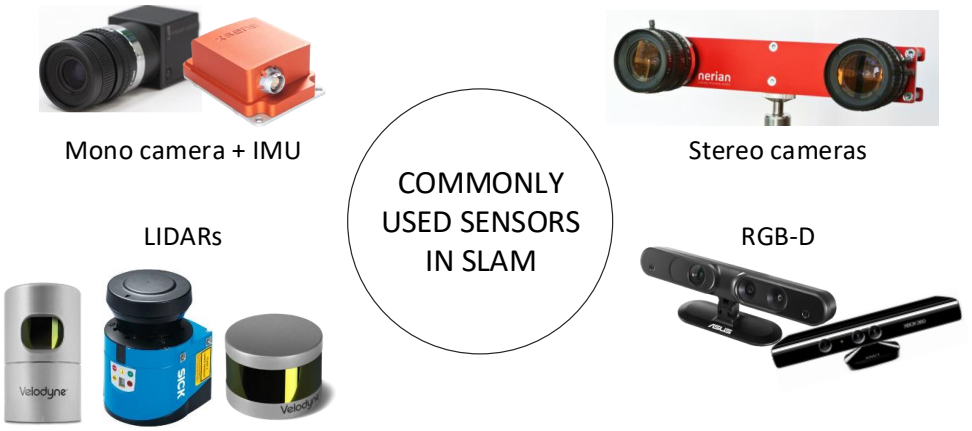

Figure 3.2: Commonly used sensor setups in SLAM.

full BA in a separate thread, not to interfere with real-time operation. Other mono SLAM solutions include mono LSD SLAM [57] and mono ORB SLAM [58]. Main advantages of using mono cameras are in their cheap price and small size, while the drawbacks are the requirement for both very precise hardware synchronization and estimation of calibration parameters between the IMU and the camera.

Accuracy of SLAM solutions based on stereo cameras greatly depends on the correct calibration of extrinsic and intrinsic camera parameters. However, once the stereo cameras are calibrated there is no need for additional sensors. Stereo SLAM solutions are currently the most used among all other SLAM algorithms. The main reason for this is in their small size, mass and price which allows them to be placed on many different robot configurations, including autonomous flying vehicles (UAVs). Besides already mentioned stereo LSD SLAM [58] and stereo ORB SLAM [60], stereo SLAM solutions are discussed in [62, 63, 64].

The release of Microsoft Kinect introduced a new type of sensor, which offered high frequency and accurate 3D depth data without using stereo cameras and IMU. Most importantly, unlike most stereo and mono camera algorithms, its depth data was dense. Since it also contained an RGB camera, this type of sensor was named RGBD sensor. Main limitations of Kinect and all other RGBD sensors that followed, was their limited range (up to 4m) and their susceptibility to sunlight which rendered them far less accurate in outdoor environments. However, due to their low price and accuracy in indoor environments, many 3D RGBD SLAM solutions were developed [62, 65, 66, 67, 68].

LIDAR sensors, in comparison to cameras, have several advantages, main being longer range, greater accuracy and larger Field Of View (FOV). On the other hand their main drawback is that they are much more expensive, especially those that have higher resolutions. The price also greatly varies depending on whether we want 3D or 2D LIDAR. 2D SLAM solutions that use 2D LIDAR sensors [69, 70] offer great accuracy, but have limitations on the scenarios they can work in. There are also SLAM solutions [37, 36, 71, 72] that work in 3D and use 2D LIDAR mounted on a pan-tilt unit, which rotates the LIDAR and provides 3D information. Although able to work in 3D, the speed of these SLAM solutions is limited by the speed of the pan-tilt unit. The SLAM solutions that use 3D LIDAR sensors offer the best speed among all LIDAR based SLAM algorithms, however, due to the highly expensive sensor and requirement to process vast amount of data in real time, these SLAM solutions

have only recently become viable and are still largely in development.

3.3.2 *Odometry*

In general, term odometry refers to estimating robot pose in reference to a starting point. The main difference between SLAM and odometry is that odometry does not optimize past states, but only uses current robot pose and information from the sensor to estimate robot's pose in the next step. Due to this, pose errors are cumulative and using only odometry to estimate robot's pose is not accurate enough in large-scale environments. However, accurate odometry is crucial in every SLAM system since it is used to calculate robot motion for the SLAM motion model (3.4). Although SLAM is capable to reduce accumulated odometry error, the less amount of pose error is introduced by the odometry, the more accurate will the final SLAM estimate be.

The most basic and most common odometry algorithm is so called wheel odometry, which uses encoders to measure wheel or joint rotation in wheel and legged ground robots. By knowing exact physical dimensions of wheel or joint, algorithm can estimate the pose change between two encoder measurements. However, due to the slippage and many other interferences from the environment, measuring robot pose this way results in the accumulation of large pose errors over time. Also, this type of odometry cannot be used in the UAVs. Because of this many advanced odometry algorithms have been developed that rely on different types of sensors. In SLAM, two mostly used odometries are so called vision odometry which relies on camera measurements, and laser odometry which relies on LIDAR data to estimate the robot's pose. As explained in the Sec. 3.1, in order to maintain accuracy, SLAM has to make connections between map landmarks from different measurements. Since both visual and laser odometries have to do the same, but only between consecutive measurements, it is often the case that features used by the SLAM are the same as the ones used in odometry algorithms. Using same type of features means that only one feature extraction algorithm is required which speeds up and simplifies the process.

Because of this tight connection between SLAM and odometry, odometry solutions are often presented together with SLAM solutions. In most cases odometry algorithm is introduced first and then the global optimization and loop closing detection is added to complete the full SLAM solution. This is why most already mentioned works on SLAM algorithms also introduce a novel odometry algorithm.

Visual odometry. Visual odometry algorithms can be divided into two main groups regarding the sensors used, and in two additional groups regarding their approach to feature selection. Based on the sensors there are visual odometry algorithms that use stereo cameras and algorithms that use mono camera with or without IMU. With regards to the feature selection there are visual odometry algorithms that use extracted features from images like Harris corners [73], Scale Invariant Feature Transforms (SIFT) [74] and Speeded Up Robust Features (SURF) [75], and those that use only pixel densities of the entire camera image. The latter are referred to as direct visual odometry algorithms. As stated in the [59] the main advantage of direct approach is that, since they do not use features, they are more robust to effects such as blur and low texture in camera images. However, they are

more sensitive to illumination changes and moving objects due to the limit in the baseline matches introduced by high non-convexity of the photometric error.

One of the first highly accurate feature-based visual odometry solution that relies only on mono camera was presented in [38] which also included full SLAM solution called MonoSLAM. Main contribution of this paper was in the way features were selected. Special strategy was developed which focused only on the most relevant image parts for feature extraction. This allowed both reduction in the number of features required and increased accuracy in comparison to other methods which treated all image segments the same. One also very important work, which set the basis for future mono visual odometry solutions called PTAM, was presented in [76]. Although original, PTAM was usable only in small spaces and was primarily used for virtual reality (VR) applications. Thereafter many variations followed which allowed usage in much larger areas. Today we have several state-of-the art methods for feature-based visual odometry using mono cameras of which some are presented in [59, 49]. Among the direct visual odometry methods that use only mono cameras, the most relevant works are presented in [57], which also includes full SLAM solution mono LSD SLAM and fast semi-direct monocular visual odometry (SVO) presented in [77].

Stereo visual odometry approaches in most cases evolve from the mono visual odometry algorithms. The good examples are odometries presented together with a full SLAM solution ORB SLAM and LSD SLAM. Shortly after mono solution was published, stereo solution was also presented in [58] for ORB SLAM and in [60] for LSD SLAM. SVO visual odometry was also released for stereo cameras in [78].

Due to the low cost of IMU sensors, many mono and stereo visual odometry approaches use them in combination with the cameras. In this case odometry algorithms are referred to as visual-inertial (VI) odometries. One of the today's state-of-the-art VI odometry algorithms is modified version of SVO presented in [79] which preintegrates inertial measurements between selected keyframes into single relative motion constraints to further increase its accuracy. Another, highly accurate example of VI odometry solution developed for both mono and stereo cameras can be found in [80]. In [81] VI solution only for mono cameras was presented while solution presented in [82] is dedicated only to stereo cameras in combination with IMU.

Among all the mentioned visual and visual-inertial odometry solutions, solution developed within our group called Ttereo Ddometry based on careful Feature Selection and tracking (SOFT) presented in [83] is currently ranked 1^{st} on publicly available KITTI dataset [84]. For this reason and also for its fast computation speed and low processor usage, SOFT has been used in experiments requiring stereo visual odometry conducted within this thesis.

SOFT has the ability to work as either pure stereo visual odometry or as VI stereo odometry if IMU data is available. Once the images from both cameras have been received and synchronized, features are extracted from each image. Feature extraction is based on corner features and blob masks on the gradient image. Features are then weighted based on their stability and their displacement from the predicted pose change. If IMU is available its measurements are used for prediction, and if its not, prediction is generated using 5-point RANSAC on features extracted from a left camera image. The output after this stage is sparse feature set used for pose change hypothesis generation using 1-point RANSAC algorithm

which results in a set of inliers. The computed inliers are then used in a Gauss-Newton optimization in order to determine final pose change estimate.

Laser odometry. As stated in the previous section (Sec. 3.3.1), LIDAR sensors offer several advantages over the cameras, main being their accuracy, FOV and the fact that they are unaffected by lighting conditions in the environment. However LIDAR sensors are much more complex to produce and their usage in robotics started later than the usage of cameras. Consequently, laser odometry solutions arrived later than visual odometry solutions, especially those capable of working in 3D. Although, many 2D laser odometry solutions exist [85, 86, 87, 88], their usage is highly limited to specific environment and can not be considered as a full replacement for visual odometry solutions. Today there are two main groups of 3D laser odometry solutions. One group consists of laser odometry solutions which convert point clouds into images, while the other group combines all the approaches that use point cloud representation directly obtained from the LIDAR.

Regardless of the approach, main problem with using LIDAR data in comparison to using camera images is in the way data are collected. Cameras, especially those with global shutter technology, record pixel data of the entire image almost instantly. LIDAR sensors on the other hand, either 2D or 3D, use rotating mirror to deflect the laser beam at various angles to acquire depth data, which means in a single scan there is a time shift between the first and the last data recorded. Although in 2D LIDARs this effect is small and in many solutions can be neglected, if one uses pan-tilt unit to rotate the 2D LIDAR, time delay is considerable. This means that if the robot is moving while taking the scan, LIDAR data must be motion compensated before using in the odometry algorithm. The simplest solution is to stop the robot while taking the scan like in [89, 90], however this severely limits the robot's usefulness. A better solution to the problem can be found in using LIDAR data continuously, as they arrive, or motion compensate the entire scan once it is recorded.

Due to the later arrival of 3D laser odometry approaches, logical step was to try and use already well developed algorithms for visual odometry with LIDAR data. This is done by converting LIDAR intesity information into the pixel values of a grayscale image. Several works have proven the effectiveness of this approach [89, 91, 92, 93, 94]. In [92] motion compensation was solved by using continuous data acquisition approach. A frame-to-frame laser odometry is proposed which uses a novel pose interpolation scheme that explicitly accounts for the exact acquisition time of each feature measurement. In [93] LIDAR data is also processed continuously by facilitating Gaussian Process Gauss-Newton (GPGN), an algorithm for non-parametric, continuous-time, nonlinear, batch state estimation. Continuous approach to LIDAR data processing for robot localization is also implemented in [94] where complete SLAM solution is presented. The authors derive the relative formulation of the continuous-time robot trajectory and formulate an estimator for the SLAM problem using temporal basis functions.

When using entire point cloud to compute odometry, motion compensation is often done using data from the IMU like in [95] and [96], sometimes even combined with highly accurate GPS solution. However, with the recent development of dedicated 3D LIDAR sensors, motion compensation is becoming less of a problem, due to their ability to produce 3D point clouds with high frequency. Moreover, modern 3D LIDAR sensors manufactured

by companies like Velodyne have internal IMU sensor which they use to produce already compensated point clouds as an output. However, if not converting them to images, the main question is how to process 3D point clouds.

Many attempts [90, 97, 98] had the philosophy that the environment is its best representation and operate directly on raw 3D point clouds. In almost all of these works, matching 3D point clouds of the environment is performed with algorithms derived from the iterative closest point (ICP) algorithm [99]. The ICP algorithm tries to find point correspondences and then minimize their distance using either point to point metric [99], point to line metric [85] or point to plane metric as in [100]. Although ICP has an advantage of implicitly solving data association problem, it suffers from premature convergence to local minima, especially when the overlap between scene samples decreases [72]. Furthermore the main drawback of every ICP algorithm is their relatively slow speed due to the need of finding correspondences in every iteration and requirement for fairly accurate initial guess of the relative pose. However, because of their simplicity and accuracy they gained a lot of popularity in the research community and became a sort of a standard solution for 3D point cloud registration. Today more than 400 different ICP variants exist. Good overview and detailed comparison and benchmarking between different variants can be found in [101]. Although, state-of-the-art ICP variants are significantly faster and more robust than the initial solutions, their speed is still a limiting factor when considering real-time laser odometry.

Since raw 3D point cloud processing is computationally intensive, a question arises what is the best environment representation that would allow real time processing while preserving precision up to a certain level. Answer to this question is especially important for SLAM systems since they must simultaneously build the map and perform localization using sensors information about the environment. Ability to work in real-time increases SLAM localization performance and enables it to build a meaningful map. One solution to the problem is in segmenting the 3D point cloud into higher level features. During segmentation of raw data into higher level features some precision is always lost but the SLAM system can be designed so that this does not significantly impact its overall performance. The maps consisting of higher level features such as polygons can be almost as good for the navigation tasks as more precise maps consisting of raw data, but also enable execution in real-time which is more important in those tasks. One way to extract features from point clouds is to use special descriptors developed for 3D point clouds like in [102] and [103]. However, although accurate in some environments a descriptor which would work accurately in largely different environments is still missing. Instead of using specifically designed descriptors many methods that use geometric objects, like planes, as features have been developed. These methods will be discussed in detail in the next chapter.

Of all the mentioned laser odometry methods, the most accurate and fastest algorithm for laser odometry, according to KITTI online evaluation protocol, is called Lidar Odometry and Mapping (LOAM) [104]. It can work using continuous data acquisition from the pan-tilt mounted 2D LIDAR as well as with full 3D LIDARs. It is also capable of incorporating IMU data when available and even a version which couples camera and LIDAR data in order to produce even more accurate results exists, presented in [105]. Key idea of LOAM is the division of the complex problem of localization and mapping into two algorithms. One

algorithm, which runs at high frequency, produces low fidelity to estimate velocity of the LIDAR, while second algorithm runs at a frequency of an order of magnitude lower for fine matching and registration of point clouds.

Because LOAM is not an open source algorithm it was not used in this thesis for laser odometry. Instead, two different algorithms were used which both perform point cloud registration based on higher level feature extraction and matching. One is Robot Vision Library (RVL) which will be extensively explained in the next chapter. The second uses Three-Dimensional Normal Distributions Transform (3D-NDT) to represent and match point clouds and is called D2D-3D-NDT [106]. NDT is compact spatial representation of a point cloud which allows fast relative pose calculation. In addition to the usage of 3D-NDT representation, a fast, global-descriptor based on the 3D-NDT is defined in [106] which is used to achieve reliable initial poses for the iterative algorithm. Both D2D-3DNDT and RVL are accurate and fast and their main advantage over LOAM is the ability to calculate covariances of the relative pose estimate which is essential for SLAM.

3.3.3 *Loop closing*

As stated in the Chapter 3.1, SLAM maintains pose and map accuracy by using constraints between matched map landmarks. While the robot is moving and new measurements taken, it is fairly easy to match map landmarks between consecutive measurements. This is because much of the landmarks extracted from measurement in step k will also be extracted from measurement in step $k+1$. However, the constraints between map landmarks effects the accuracy of other landmarks less and less the more away they are. If we imagine a mobile robot moving forward, the constraints made with landmarks from current measurement y_k will effect the accuracy of map landmarks taken from measurement y_i less, the larger the difference $k-i$ is. The question is how to create constraints between map landmarks taken from measurement y_k and the ones extracted from the measurement taken much earlier.

The connection between landmarks from two measurements, y_i and y_j, can be made if they describe the same feature from the environment. This means that connections can be made when two measurements are taken from roughly the same pose in the environment. The problem is that we can not know when the robot is in the same place simply by comparing current and previous poses because of the pose errors accumulated with time. This is why we need an algorithm that will be capable of detecting when the robot returns to the already visited place in the environment by comparing current and previous measurements. Group of algorithms that solve this problem are known as loop closing detection algorithms. As all other algorithms related to SLAM, they depend on the sensors used. Two main groups exist, one for the image based and one for the LIDAR based approaches.

Detecting loop closing using camera images. The image based methods for loop closing detection can be divided into three main categories:

- map to map - builds sub-maps consisting of features from several camera images and then compares those sub-maps by the number of features in common.
- image to map - compares current image to the whole set of previously extracted features in the map.

- image to image - compares features from current image to features from previously extracted images separately.

One of the most well known methods, which sets the benchmark for all other image based loop closing methods, is FAB-MAP presented in [107]. FAB-MAP uses dictionary to compare images, which has to be built prior to running it online. Features extracted from images in the learning dataset are stored in the dictionary in a form of Bag of Words (BoW). Similar features are clustered into the same category, thus reducing the number of different words. When run online, features extracted from current image are compared to the words in the dictionary and number of features belonging to each group in the dictionary is calculated. The match between images is found based on this numbers which drastically increases the matching speed. Besides matching only numbers of similar words, FAB-MAP also takes into account the probability that if one of the detected features from the dictionary is present, some other features are also located in the same image. In order to calculate this probability fast FAB-MAP uses Chow Liu Tree (CLT) [108] which is also built during the learning phase.

Today, several alternatives to FAB-MAP exist. In [109] authors present a method for image based loop closing called HAsh-based LOop Closure (HALOC). It extracts SIFT features from an image and then calculates small hash vector from all the extracted SIFT descriptors. These vectors can then be compared extremely fast which allows it to run online even with thousands of images. The added benefit is that the method is open source and available under the BSD license. Solution presented in [110] uses BoW approach and SUFR features. It achieves real time performance by novel memory handling approach in which frequently visited locations are kept within the working memory (WM), while less frequently visited ones are stored in long term memory (LTM). However, if a frequency changes, locations can efficiently be transferred from LTM to WM and vice-versa. Loop closing solution presented in [111] uses a Local Difference Binary (LDB) [112] descriptor for image matching. Additionally authors add disparity information to it and call it D-LDB, which allows more accurate and robust detection. However, it requires stereo cameras to calculate disparity information. Solution presented in [113] uses BoW and ORB features to produce accurate loop closing detection with very low computation cost. Besides loop closing detection it also offers fast relocalization solution. More detailed analysis and comparison between visual loop detection solutions can be found in [114].

DETECTING LOOP CLOSING USING 3D POINT CLOUDS. Similarly to laser odometry solutions, loop closing solutions that use 3D LIDAR data have appeared later than the ones based on camera images. This was mainly due to the lack of universal descriptors for 3D point clouds and large amount of data required to be processed in real time. Today, similarly to visual based loop closing detection, we can identify three main groups of loop closing algorithms that use 3D point clouds:

- local descriptors based algorithms - identify local features in each point clouds and use them for matching.
- global descriptor based algorithms - describe entire point cloud with a single descriptor and use it for matching.

- object based algorithms - detect whole objects in point clouds and use them for matching.

The work presented in [115] belongs to the first group. It uses key point descriptors called 3D Gestalt descriptor and nearest neighbour search to identify matches. In order to speed this process, matches returned by the nearest neighbour search are accumulated into vote scores for their associated places. This approach results in sub-linear computation time with respect to the number of places. Also a methodology for statistical modelling of vote distribution is developed in order to allow more accurate matching. In [116] authors present a solution which uses three different types of descriptors, the 3D Gestalt descriptor, the neighbour-binary landmark density (NBLD) descriptor [117] and the Boxli descriptor. The main advantage of using these three types of descriptors is the ability to match sparse and dense point clouds, which allows it to work with 3D point clouds generated by both 3D LIDARs and stereo cameras. The solution presented in [118] uses BoW approach to build a dictionary. Then, when working online, BoW is used to get initial matches between received point clouds. For each of those pair matches, transformation between pair is calculated using point features. The transformations are then ranked and the transformation with the highest rank is reported as a match.

Some of the solutions that belong to the second group are [119, 120, 121]. Solution presented in [119] uses NDT to describe point clouds. NDT is used to represent point clouds as feature histograms which can then be used for easy and fast matching. Work described in [121] uses rotation invariant features that describe both geometric and statistical properties of a point cloud. These features are then used for input into the AdaBoost algorithm that calculates non-linear classifier used for matching point clouds. In [120] authors present a solution which describes entire point clouds as histograms. In order to do that they first represent each point with single value equal to its distance from the robot's local coordinate frame. This values are then normalized and discretized into fixed number of intervals. Intervals are then used for histogram generation. Histograms are then compared using Wasserstein metric.

Finally, the algorithms that would belong to the third group and would be suited for largely varying environments are still missing. However, they are intensively researched and the one with promising results which almost belongs to this group was presented in [122] and is called SegMatch. As the authors state, full 3D object segmentation from point cloud is difficult due to the many reasons, one of them is the requirement to see the entire objects in both point clouds that are matched. This is why the authors use more descriptive shapes than keypoint-based features, but do not require this shapes to represent entire objects. SegMatch first filters point clouds into voxels, than ground plane is removed, and finally adjacent voxels are grouped into shapes based on vertical means and variances. The shapes are then converted into features using two descriptors. One descriptor is eigenvalue based and the other is histogram based. Extracted features are used in the matching procedure that relies on a geometric verification step.

Regardless of the used loop closing detection solution once the loop is detected pose constraints between the features have to be estimated and sent to the SLAM back-end. Some loop closing detection algorithms perform relative pose estimation as part of their matching process. If this is the case, pose constraint can simply be obtained from the loop detection

algorithm once the match is reported. If the loop detection algorithm does not calculate relative pose, some of the mentioned laser or visual odometry solutions can be used since now we know which two measurements need to be used for pose constraint generation. In any case, for loop closing detection algorithm to work, robot has to revisit already visited places. SLAM solutions can either passively wait for this to happen, or influence the robot to do it. SLAM solutions with front-ends that have this ability are called active SLAM solutions.

3.3.4 *Active SLAM*

SLAM is mostly treated as a passive system which means that it does not send any commands to the robot - it only acquires sensor data and uses that data to build a map of the surrounding environment and to localize the robot in that map, i.e. it does not decide where the robot must go. However, controlling the robot can prove to be beneficial since it can ensure that the robot returns to the previously visited places and closes the loop. A good example of scenario in which SLAM needs to influence robot's trajectory is when SLAM is coupled with the exploration algorithm. Exploration algorithm needs a map and a location from the SLAM to determine next best place for the robot to go and take measurements. However, returning to the same place is in contrast to the main goal of the exploration algorithm which is to explore the environment in shortest time possible. This is why active SLAM algorithm needs to modify goal planned by the exploration algorithm in order to allow loop closing and ensure that uncertainty of SLAM localization and mapping remains within the desired boundaries.

In [123] *"localization metric"* is introduced which provides the uniform basis for measuring localization quality. The localization quality over a trajectory is combined with the navigation cost of the trajectory and information gained from the environment by following the trajectory in one single criterion. That criterion is used for determining on which trajectory should the robot travel on. Numerical method that uses a non-linear Model Predictive Control (MPC) for estimating the pose and the map error, that will occur by following one trajectory, is introduced in [124]. In [27] relative entropy is used as a measure for the information gain. Environment is discretized into grids and optimal trajectories (according to information gain criterion) are planned on the global scale thus minimizing unnecessary loop closures and noise while ensuring more precise maps. In [28] information gain is also used as a criterion for choosing the trajectory. The difference is that Rao-Blackwellized particle filter is used for SLAM and entropy calculation. In [29] global planning is avoided by using attractors in combination with a local planning strategies. The attractor is placed in the environment according to the current robot goal (explore, improve map or improve localization). The attractor then influences information gain computed by a local planner which uses MPC. In [26], the FastSLAM is used. When localization uncertainty reaches the predefined level the exploration task is stopped and possible previously visited states are considered for loop closing. The state with the highest information gain and the easiest reachability is chosen. When the robot reaches that state it follows its previously traversed path until the uncertainty drops below a desired level and then the exploration is continued. One different approach to minimize localization and mapping errors is used in [125], where reinforcement learning is used to generate a robot's motion in such a way that it

minimizes error generation in the mapping process. This approach enables the usage of a simple exploration strategy while maintaining the location and map accuracy.

With the successful implementation of odometry solution, loop closing detection algorithm, pose constraint estimation and possible integration of the ability too influence the robot movement, SLAM front-end is complete. The other crucial component of every SLAM system is its back-end.

3.4 SLAM BACK-END

As stated in the Sec. 3.2 SLAM back-end is responsible for optimizing landmark and robot poses by taking into account odometry measurements and constraints from the loop closing. Today two main groups based on fundamentally different optimization approaches exist. First group contains approaches that rely on filtering solutions to optimize poses, while the second group contains approaches that use graph-optimization techniques. In the Sec. 3.2 all important solutions from both categories were mentioned. In the present section each group will be covered more extensively together with brief descriptions of current state-of-the art solutions. However, since the main contributions of this thesis which rely on the filtering based approaches, they will be covered in more details than graph-optimization approaches.

3.4.1 *Filtering based SLAM back-end*

First filtering based solutions used EKF, then followed solutions that use EIF and lastly PF based solutions were presented. Through time all filtering solutions evolved and many advanced methods were presented which tried to overcome main disadvantages of the filter they were based on.

In general, all filtering based solutions have two main steps: correction and prediction. Prediction step is used to evaluate current robot pose using motion model (3.4) and odometry inputs, while the correction step is used to optimize pose accuracy based on incoming measurements. Using the motion model (3.4) and the Bayes theorem, we can write general expressions for each of this steps. For prediction step the equation is

$$P(X_k, m|z_{1\dots k-1}, \Omega_{1\dots k}, X_1) = \int P(X_k|X_{k-1}, u_k)P(X_{k-1}, m|z_{1\dots k-1}, \Omega_{1\dots k-1}, X_1)dX_{k-1}\,. \quad (3.7)$$

The first term under the integral represents the distribution of pose X_k calculated using the motion model based on the state X_{k-1} and odometry input Ω_k, while the second term represents the distribution of state X_{k-1} and map landmarks m depending on measurements and odometry inputs received up to step $k-1$. For correction step the general equation is

$$P(X_k, m|z_{1\dots k}, \Omega_{1\dots k}, X_1) = P(X_k, m|z_{1\dots k-1}, u_{1\dots k}, X_1)\frac{P(z_k|X_k, m)}{P(z_k|z_{0\dots k-1}, \Omega_{1\dots k})}\,. \quad (3.8)$$

Here, the first term is equal to the output of the prediction step, while the fraction term takes into account information received by the new measurement z_k. Depending on the selected filter, solutions to equations (3.7) and (3.8) are found using different apporaches.

EKF SLAM BACK-END. The basis of every EKF SLAM algorithm is to describe the motion and measurement models using Gauss distribution. This means that in EKF SLAM both robot and map landmark poses act as Gauss random variables. The expectancy of X_k and m is

$$E\begin{bmatrix} X_k \\ m \end{bmatrix} = \begin{bmatrix} \mu_{X_k} \\ \mu_m \end{bmatrix} = \mu_k\,, \tag{3.9}$$

where μ_k holds the expectancy values of every observed map landmark

$$\mu_k = \begin{bmatrix} \mu_k^1 & \mu_2^k & \dots & \mu_N^k \end{bmatrix}^T\,, \tag{3.10}$$

where N is the number of observed landmarks. The covariance matrix is equal to

$$E\left[\begin{pmatrix} X_k - \mu_{X_k} \\ m - \mu_m \end{pmatrix}\begin{pmatrix} X_k - \mu_{X_k} \\ m - \mu_m \end{pmatrix}^T\right] = \begin{bmatrix} \Sigma_{X_k} & \Sigma_{X_k,m} \\ \Sigma_{m,X_k} & \Sigma_m \end{bmatrix} = \Sigma_k\,, \tag{3.11}$$

where Σ_m represents covariance matrix of every observed landmark

$$\Sigma_m = \begin{bmatrix} \Sigma_m^{1,1} & \Sigma_m^{1,2} & \cdots & \Sigma_m^{1,N} \\ \Sigma_m^{2,1} & \Sigma_m^{2,2} & \cdots & \Sigma_m^{2,N} \\ & & \vdots & \\ \Sigma_m^{N,1} & \Sigma_m^{N,2} & \cdots & \Sigma_m^{N,N} \end{bmatrix} \tag{3.12}$$

where $\Sigma_k^{i,i}$ represents the covariance of map landmark m^i and $\Sigma_k^{i,j}$ represents cross-covariance between map landmarks m^i and m^j.

Using Gauss distribution assumption, the motion model (3.4) in EKF SLAM becomes

$$P(X_k|X_{k-1}, \Omega_k) \Longleftrightarrow X_k = f(X_{k-1}, \Omega_k) + w_k \tag{3.13}$$

where f is a function that calculates expectancy of robot pose X_k based on odometry measurements Ω_k and previous pose X_{k-1} and w_k represent white Gaussian noise with covariance Q_k. Similarly the measurement model becomes

$$P(z_k|X_k, m) \Longleftrightarrow z_k = h(\mu_{X_k}, \mu m) + v_k \tag{3.14}$$

where h represents sensor measurement model, and v_k represents Gauss white noise with covariance R_k. Using these equations, standard EKF equations can be applied to get the expectancy and covariance of X_k and m after the prediction and correction steps. For the prediction step the EKF SLAM equations are

$$\bar{\mu}_k = f(\mu_{k-1}, \Omega_k)\,, \tag{3.15}$$

$$\bar{\Sigma}_k = F_k \Sigma_{-1} F_k^T + Q_k \tag{3.16}$$

where $F_k = \nabla f(\mu_{k-1})$ represents Jacobian of f calculated at μ_{k-1}. The correction equations are

$$\begin{bmatrix} \mu_{X_k} \\ \mu_{m_k} \end{bmatrix} = \begin{bmatrix} \bar{\mu}_{X_{k-1}} \\ \bar{\mu}_{m_{k-1}} \end{bmatrix} + K_k\left[z_k - h(\bar{\mu}_{X_k}, \bar{\mu}_{m_{k-1}})\right] \tag{3.17}$$

$$\Sigma_k = \Sigma_{k-1} - K_k H_k \Sigma_{k-1} \tag{3.18}$$

Algorithm 1: EKF-SLAM pseudocode

1: Set initial value of X_1 and extract initial landmarks
2: *loop*:
3: Complete prediction step by calculating $\bar{\mu}_k$ and $\bar{\Sigma}_k$
4: Retrieve new measurement z_k
5: Extract new features and estimate their pose m_k^i
6: **for** Every extracted feature **do**
7: **if** Observed for the first time **then**
8: Extend μ_k by adding m_k^i
9: Set covariance of new feature $\Sigma_k^{i,i}$ to high value
10: Extend Σ_k and add $\Sigma_k^{i,i}$, set all $\Sigma_k^{i,j}$, $j \neq i$ to zero
11: **end if**
12: Calculate difference between the stored expected value of landmark pose μ_k^i and the newly measured one z_k^i
13: Calculate new Σ_k and μ_k
14: **end for**

where

$$K_k = \Sigma_{k-1} H_k^T S_k^{-1}$$
$$S_k = H_k \Sigma_{k-1} H_k^T + R_k$$

and $H = \nabla h$ is Jacobian of h calculated at $(\bar{\mu}_{X_{k-1}}, \bar{\mu}_{m_{k-1}})$. The full algorithm of EKF-SLAM is depicted in Algorithm 1.

Two things are worth pointing out. First, since sensors are mounted on the robot only landmark poses relative to the robot local frame can be measured. In order to get difference in step 11, which is referred to as innovation, landmark absolute pose μ_k^i has to be transferred into the same local frame. This is done using predicted robot pose $\mu_{\bar{X}_k}$ and is the reason why measurement model h is dependant on $\bar{\mu}_{X_k}$. Second, when calculating new Σ_k and μ_k in step 12, if landmark was observed for the first time, the innovation in step 11 will be 0. This means that second term in update equation (3.17) will also be zero and hence $\mu_k = \bar{\mu}_k$. However, the covariance of $\Sigma_k^{i,i}$ will change according to the equation (3.18). Cross-covariance Σ_{X_k, m_k^i} will also change because of the Jacobian H.

One iteration of algorithm 1 is shown in Fig. 3.3. First step 1 is completed by setting initial pose X_1 and extracting landmarks m_1^i from measurement z_1. Then new pose X_2 is predicted and new landmarks m_2^i are extracted. This completes steps 3-5. Now landmark m_1^2 is matched with landmark m_2^1 and landmark m_1^3 is matched with landmark m_2^2. Landmarks m_2^3 and m_2^4 are added as new landmarks, in steps 8-10, since they were not matched with any landmarks from z_1. Finally update is performed and new robot pose and landmark poses are estimated in steps 12-13.

Every EKF SLAM algorithm has the same problems as any other algorithm which relies on the EKF. The main three are:

- Convergence: after some time, covariances describing the uncertainty of map landmarks will converge to the same value and after that they will not change according

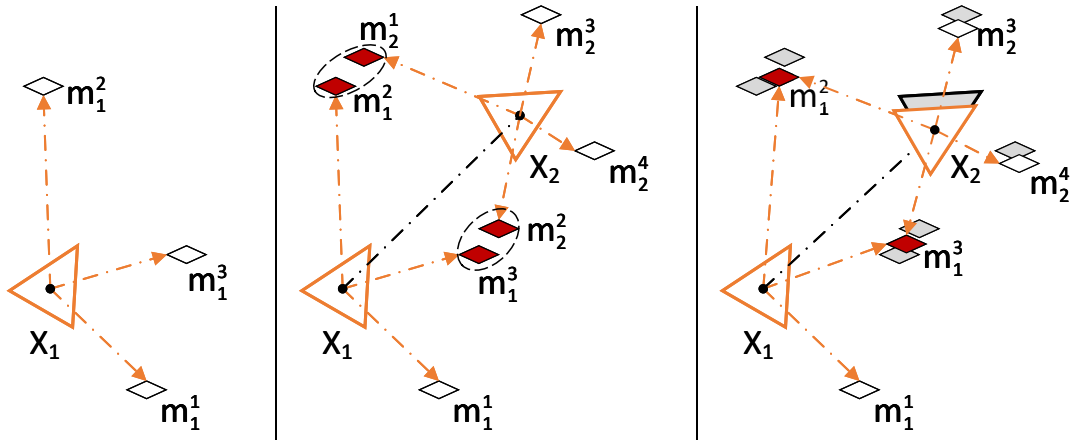

Figure 3.3: One iteration of the EKF-SLAM algorithm depicted in Algorithm 1. Red landmarks represent landmarks that were observed from two measurements, while grey objects represent robot and landmark poses before the update occurred.

to the measurement and motion uncertainties. Due to this, a correction will have less impact to the overall accuracy than it should if the covariances were updated accordingly.

- Mathematical complexity: Whenever new landmark is introduced, the dimension of covariance matrix Σ_k is increased by 1. Since calculations of K_k requires the inversion of S_k which has the same dimensions as Σ_k the inversion complexity rises quadratically with the number of added map landmarks.

- Linearity: Both measurement and motion model have to be linearized in order to calculate Jacobians F and H. Depending on the amount of non-linearities in the models the errors introduced this way can severely effect the SLAM accuracy.

One of the first truly successful implementations of the EKF-SLAM algorithm was presented in [32]. It proved that indeed it is possible for an autonomous robot to start in an unknown environment and, using relative observations only, incrementally build a perfect map of the world and to compute simultaneously a bounded estimate of the vehicle location. This was proved using real world experiments with a robot equipped only with a millimetre-wave radar. In [35] a similar result was achieved for the indoor environment and the paper focused on the SLAM front-end which is based on sonar. Main contributions are concerned with smart handling noisy and sparse sonar data in order to convert them to reliable map features. The work presented in [36] minimizes the number of map features by using highly reduced surface models segmented from point clouds obtained by rotating 2D LIDAR. Also features are organized into an elastic graph structure with bounded number of sub-maps which results in a faster correction step and can be calculated online. The EKF-SLAM solution based on planar features extracted from measurements obtained by 2D rotating LIDAR is also presented in [37]. It uses heuristic represented residuals to model systematic errors which could not be compensated during calibration and in-tern improves overall accuracy due to the more accurate error description. One truly revolutionary approach to the EKF-SLAM based on mono camera was presented in [38] called MonoSLAM. Although

MonoSLAM was already discussed in the section related to the visual odometry, it also holds important novelties regarding EKF-SLAM. In order to reduce the number of features authors developed a method which builds a probabilistic framework of carefully selected features which create sparse, but persistent map. They also introduced, for the first time, an active approach to mapping and also use general motion model to describe camera movements in a smooth way which increases prediction accuracy. In [39] authors combined EKF with RANSAC to increase the accuracy and speed by using EKF estimate to reduce the number of samples used as input for RANSAC.

Although many new EKF-SLAM approaches do minimize some of the mentioned EKF problems, none of them completely solves them. Depending on the implementation some methods are more robust than the others, but still, due to the EKF core, in some situations and/or after a certain time the method has been working online, these problems will start to effect the accuracy.

PF SLAM BACK-END. Particle filter based SLAM approaches are fundamentally different from the EKF-SLAM solutions. The two key differences are: (i) that the PF does not linearize the motion model and (ii) that it does note require that the state distributions act as Gauss distribution. The main problem with PF is that it is computationally far too complex to be used for estimating the pose of every map feature extracted from the environment. Because of this it is impossible to implement PF directly on the SLAM problem defined in (3.2). However, it is possible to reduce the state space by reformulating the SLAM problem using Rao-Blackwellization (R-B). It is based on partitioning the joint state according to the product rule:

$$P(X_1, X_2) = P(X_2|X_1)P(X_1)\,. \tag{3.19}$$

This means that now, if $P(X_2|X_1)$ can be analytically computed, instead of using PF to estimate joint distribution $P(X_1, X_2)$ we can use PF only to estimate $P(X_1)$ which is much less computationally complex. In order to do this with the SLAM problem (3.2), it is first extended so that it includes not only current robot pose X_k, but also a set of past robot poses $X_{1:k-1}$. Now it can be partitioned as:

$$P(X_{1\ldots k}, m|z_{1\ldots k}, \Omega_{1\ldots k}, X_1) = P(m|X_{1\ldots k}, z_{1\ldots k})P(X_{1\ldots k}|z_{1\ldots k}, \Omega_{1\ldots k}, X_1)\,. \tag{3.20}$$

Partitioning the distribution this way, means that now the distribution is on the entire discrete robot trajectory, rather than on the single pose X_k. Although this may seem as making the problem even more complex, this is not the case due to one key fact. Map landmarks are now conditioned only on the trajectory states and not on each other. This means that, since measurements are also mutually independent, one map landmark is independent on the others. As a result, introducing new map landmarks now increase the complexity linearly as opposed to the quadratic complexity increase in EKF SLAM. SLAM back-ends that estimate discrete robot trajectory, instead of the current robot pose and the poses of map landmarks, are also referred to as pose graph filtering SLAM back-ends.

Using distribution partitioning (3.20) in PF-SLAM means that each state (particle) consists of trajectory which is represented as weighted samples and map which is computed analytically. The joint distribution at time step k thus consists of $i = 1 \ldots N$ particles, where

each particle is defined as:

$$p_k^i = \{w_k^i, X_{1\dots k}^i, P(m|X_{1\dots k}^i, z_{1\dots k})\} \qquad i = 1\dots N \tag{3.21}$$

where w_k represents particle weight. The map m of each particle consists of independent Gaussian distributions each representing one map landmark:

$$P(m|X_{1\dots k}^i, z_{1\dots k}) = \prod_j^M P(m^j|X_{1\dots k}^i, z_{1\dots k})\,. \tag{3.22}$$

This means that each of the landmarks m^j is represented by its expectancy μ_{m^j} and covariance $\Sigma_m^{j,j}$. However, instead of having one large Σ containing all covariances of map landmarks which increases every time new landmark is added, each landmark's covariance $\Sigma_m^{j,j}$ is constant in size. Moreover, this size is small and determined by the exact pose representation. For example covariance of map landmark whose orientation is represented by quaternion would have size of 7×7. This is possible because map landmarks are now independent of each other which means that cross-covariances $\Sigma_m^{i,j}$, $i \neq j$ are equal to zero.

When new discrete state X_k is added into the trajectory, new map landmarks are extracted from the measurement z_k. Those observed for the first time are added into the map and those that already exist and are observed again are processed individually using EKF, which renders map estimation trivial. However, PF implementation for estimating trajectory $X_{1\dots k}$ distribution is more complex.

Basically, PF-SLAM algorithm consists of four main steps:

1. For every particle p_i calculate the distribution of new discrete pose X_k based on trajectory history distribution $X_{1\dots k-1}^i$, measurements $z_{1:k}$ and odometry input Ω_k

$$X_k^i \sim \pi(X_k|X_{1\dots k-1}^i, z_{1\dots k}, u_k)\,, \tag{3.23}$$

   where π stands for selected distribution which can vary depending on the algorithm. Then add X_k^i to the particle trajectory $X_{1\dots k}^i = \{X_k^i, X_{1\dots k-1}^i\}$.

2. Calculate new particle weight w_k^i using the importance function:

$$w_k^i = w_{k-1}^i \frac{P(z_k|X_{1\dots k}^i, z_{1\dots k-1})P(X_k^i|X_{k-1}^i, \Omega_k)}{\pi(X_k|X_{0:k-1}^i, Z_{0:k}, \Omega_k)}\,. \tag{3.24}$$

   The first term in the numerator represents the measurement model and the second term stands for the motion model. The measurement model is different than (3.5) and is obtained by marginalizing map components:

$$P(z_k|X_{1\dots k}, z_{1\dots k-1}) = \int P(z_k|X_k, m)P(m|X_{1\dots k-1}, z_{1\dots k-1})\mathrm{d}m \tag{3.25}$$

3. Do resampling if necessary. Resampling selects number of particles and discards the rest. The selected particles are given new uniform weight coefficients. The exact method of selecting which particles to keep and when to do it differs among the PF algorithms.

4. Lastly, extract landmarks from new measurement z_k. Landmarks observed for the first time are added into the filter and landmarks that were not observed are left unchanged. The landmarks that have already existed, and were observed again, are updated using EKF and information from the new pose distribution.

Two main implementations of R-B PF-SLAM exist, FastSLAM 1.0 [40] and FastSLAM 2.0 [41]. They differ in the selection of the distribution π used in steps 1 and 2. FastSLAM 1.0 uses motion model (3.4) for π:

$$X_k^i \sim P(X_k|X_{k-1}^i, \Omega_k)\,, \tag{3.26}$$

which means that weights in step 2 are calculated based only on the marginalized measurement model:

$$w_k^i = w_k^{i-1} P(z_k|X_k^i)\,. \tag{3.27}$$

FastSLAM 2.0 includes the observation in the distribution π:

$$X_k^i \sim P(X_k|X_{1\ldots k}^i, z_{1\ldots k}, \Omega_k) \tag{3.28}$$

and π can then be expressed as:

$$P(X_k|X_{1\ldots k}^i, z_{1\ldots k}, \Omega_k) = \frac{1}{C} P(z_k|X_{1\ldots k}^i, z_{1\ldots k-1}) P(X_k|X_{k-1}^i, \Omega_k) \tag{3.29}$$

where C is normalizing constant. This results in the particle weight being calculated as:

$$w_k^i = w_{k-1}^i C \tag{3.30}$$

Main advantage of FastSLAM 2.0 over FastSLAM 1.0 is that its distribution π gives the smallest possible variance in importance weight calculation, i.e. it is locally optimal. The main problem of both FastSLAM 1.0 and 2.0 is that they suffer degeneration since they cannot *forget the past*. The main reason for this is that marginalizing the map in order to calculate the weight in step 2 means that the result is dependant on the pose and measurement history. When using resampling, some of this history is lost, and so is the statistical accuracy of the algorithm.

EIF SLAM BACK-END. EIF is essentially EKF and as such retains all the advantages and disadvantages together with the requirement that both process and measurement model act as Gauss distributions. The main difference between EKF and EIF is that instead of estimating expectancy μ and covariance Σ, EIF estimates information vector η and information matrix Λ. The relations between (μ, Σ) and (η, Λ) are:

$$\Sigma = \Lambda^{-1} \qquad \mu = \Sigma\eta \tag{3.31}$$

The main advantage of EIF SLAM over EKF SLAM was presented in the first viable solution to the EIF SLAM [126]. In [126] authors used the EIF ability to be distributed and decentralised and applied it to the SLAM solution for multiple robots. Authors showed that there are several key advantages of using information form in distributed multiple agents SLAM, however its usefulness in SLAM for a single agent over the EKF solution was proven later in

[1]. There, authors observed that information matrix of EIF SLAM exhibits one key property which would enable much faster computation of EIF SLAM equations. This property is shown in Fig. 3.4. The left image shows the estimated map and robot pose, the centre image shows the normalized covariance matrix estimated using EKF approach and the right image shows the information matrix. Pixels in image are darker the closer the value of matrix is to 1 and are lighter the closer the matrix value is to 0. As can be seen, in information matrix, much of the elements are close to zero values (large white areas).

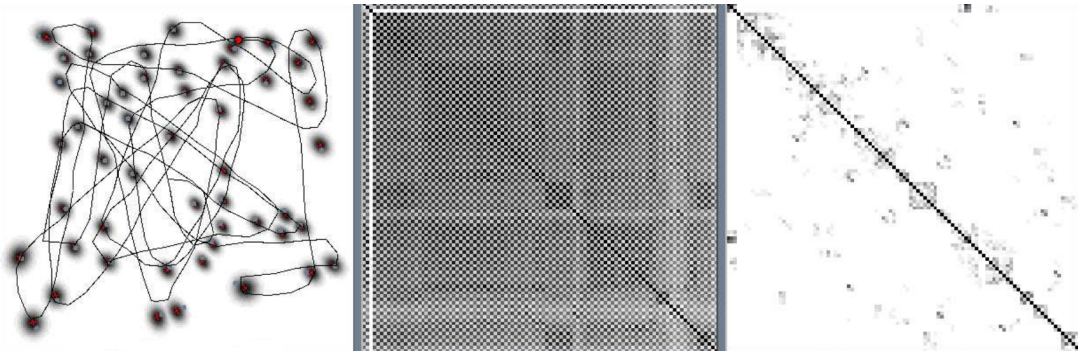

Figure 3.4: Nearly sparse structure of the information matrix in EIF-SLAM. Left image represents poses of landmarks and robot poses, center image represents covariance matrix in EKF-SLAM and right image represents information matrix in EIF-SLAM. Pixels in images representing covariance and information matrices are darker the closer the value of matrix is to 1 and are lighter the closer the matrix value is to 0. The image was taken from [1].

Many near zero values in the information matrix represent weak constraints between the poses. This means that many of the constraints within the information matrix have almost no effect on the state distribution. This result motivated the authors to try and utilize this sparse structure in order to make EIF SLAM much faster than its EKF counterpart. Authors called this new version of EIF, which uses sparsity of the information matrix, Sparse Extended Information Filter (SEIF). The main difference between EIF and SEIF is in the sparsification step, performed after every measurement update. Sparsification step detects which constraints in the information matrix hold little information and removes them. This ensures that information matrix always remains sparse with all the week constraints replaced by zero blocks. Consequently, equations of all other steps are modified in order to effectively use the sparsity of the information matrix. The usefulness of sparsity in SLAM as well as EIF back-end will be discussed in more detail in the next chapter, while for the exact SEIF implementation the reader is referred to [1].

Main advantage of SEIF over EIF and EKF is the fact that the computation and memory complexity rises almost linearly with the addition of new map landmarks instead of quadratically thanks to the sparse information matrix. However, sparsification of the information matrix introduces errors and makes SEIF SLAM less accurate than EKF and EIF SLAM solutions.

The three filters, EKF, PF and EIF serve as a backbone of all existing filtering SLAM back-ends. Many SLAM solutions based on these back-ends have applied modifications to the filtering equations in order to make them more robust, faster and more accurate.

However, the core has always remained the same. As stated in the Sec. 3.2 fundamentally different approaches to the filtering based back-ends are, SLAM back-ends based on graph optimization. Although their development began almost at the same time as that of filtering based ones, due to the lack of theoretical solutions and fast computers they could not compete with the filtering approaches. However, in the last decade this has begun to change and today graph optimization approaches have taken the leading role in modern SLAM solutions.

3.4.2 *Graph optimization SLAM back-end*

All SLAM back-ends that use graph optimization to estimate robot pose and map formulate SLAM as nonlinear least squares problem instead of relying on prediction and correction steps. In general, non linear least squares problem is given in a form:

$$\min_{x \in R^n} F(x) = \min_{x \in R^n} \sum_{i=1}^{m} r_i(x)^2 \tag{3.32}$$

where n is the number of variables and r is residual representing the error between the measured value y and predicted value $f(x)$. The goal is to find x which minimizes F. For this, different methods can be used; however, today most commonly used is the Gauss-Newton (GN) method or its modified version Levenberg–Marquardt (LM) method. The basis of both methods is that they start from some initial guess x_0 and then iteratively change x by some value Δx:

$$x_{k+1} = x_k + \Delta x \tag{3.33}$$

until the minimization criterion has been satisfied. At each iteration the model is linearized by approximation to a first-order Taylor polynomial expansion about x_k and the final result of each iteration is linear least squares problem:

$$(J^T J)\Delta x = J^T \Delta r' \tag{3.34}$$

where J is the Jacobian of f and $\Delta r'$ is the residual between y and $f(x_k)$. This linear problem can be solved using some of the well established methods, like Q-R decomposition. Once Δx is calculated, it is used to get x_{k+1} which serves as an initial estimate for the next iteration. The modified version of least squares problem is the weighted non linear least squares, which is used when observations are not equally reliable:

$$S = \sum_{i=1}^{m} W_i r_i^2 \tag{3.35}$$

The final linear problem obtained for weighted least squares is similar to the normal least squares:

$$(J^T W J)\Delta x = J^T W \Delta r' \tag{3.36}$$

The weighted least squares is more suitable to SLAM since in SLAM measurement reliability is defined by the covariance and information matrices.

Today many different solutions to SLAM defined as non linear least squares problem exist. However, the two most accurate, fastest and robust solutions are iSAM [54] with its improved version iSAM2 [56] and g^{2}o [55].

The most general solution of all f them is g^{2}o which is not limited to the SLAM solution, but can work for any problem that can be formulated as a graph structure. It uses LM algorithm to minimize non linear least squares problem defined as:

$$\min_x F(x) = \min_x \sum_{i,j \in C} e(x_i, x_j, z_{i,j}^T) \Lambda_{i,j} e(x_i, x_j, z_{i,j}) \,, \tag{3.37}$$

where $x = (x_1^T, \ldots, x_n^T)$ is a vector of parameters, $z_{i,j}$ represents the mean of the constraint between x_i and x_j, $\Lambda_{i,j}$ represents its information matrix and e represents error function which measures how well the real values of x_i and x_j satisfy the constraint z_{ij}. In context of graph structure each vector x_i can be viewed as a node and each constraint as an edge between these nodes. The information matrix serves as a weight distribution information, where constraints that hold more information have higher weights.

When g^{2}o is used for SLAM each node represents pose of a robot or landmark and edges are added based on the measurement and odometry information. The main benefit of defining the SLAM problem this way is that the Jacobian of the error function used to solve linearized problem at each iteration of LM algorithm is sparse since the error function of each constraint depends only on the values of two nodes. This is utilized in g^{2}o by using sparse matrix solvers to drastically increase the speed of solving the linear least squares problem. One more advantage of g^{2}o is the fact that it, due to its multi-purpose design, allows for many different parametrizations of x. However, although its multi-purpose design is one of its main advantages it is also one of its main weaknesses in the context of SLAM since it does not accommodate all the specifics of the SLAM designed solutions such as iSAM. Nevertheless, g^{2}o is currently the most widespread algorithm used in modern graph optimization SLAM back-ends.

The first version of iSAM was presented in [54]. It explicitly defined the least square problem to accommodate for SLAM motion and measurement models:

$$\min_{X,L} F(X, L) = \min_{X,L} \left(\sum_{i=1}^{M} \| f_i(X_{i-1}, \Omega_i) - X_i) \|^2 + \sum_{k=1}^{K} \| h_k(X_{i_k}, l_{j_k}) - z_k \|^2 \right) , \tag{3.38}$$

where $X = \{X_1 \ldots X_M\}$ represents robot poses, Ω_i is odometry information, $L = \{l_1 \ldots l_N\}$ are map landmarks, $z = \{z_1 \ldots z_K\}$ represents landmark measurements, and f and h are motion and measurement model, respectively. iSAM uses Q-R factorization to solve the linear least squares problem. Its main contribution is that it does not perform full Q-R factorization each time new measurement or odometry information is received. Instead it exploits the fact that many measurements affect only local landmark and robot poses, and leave most of the other poses intact. When new measurement arrives it only partially updates the information matrix using Givens rotations [127]. However, this approach requires optimal variable ordering to affect the smallest possible area of the information matrix. As new data is added, ordering of the information matrix diverges more and more from the optimal, and thus reordering is required in order to avoid unnecessary changes in the information matrix. iSAM performs this reordering periodically, however during this step linearization is required and accuracy is lost.

In order to solve this problem authors presented an improved version iSAM2 [56] in which they used novel data structure called Bayes tree, which they have first presented in [128]. The main advantage of Bayes tree is removing the need to repeatedly solve linear

least square problem to update linearization point of the next iteration in non linear least squares solvers. Using Bayes tree in non linear least squares allows the replacement of this step by fluid relinearization of a reduced set of variables which results in higher efficiency, while completely retaining accuracy and sparsity of the information matrix. Bayes tree implementation in iSAM2 allows even more effective detection and changing only those parameters of the information matrix that are affected by the new measurement or odometry information. This makes it faster and more accurate than the original iSAM.

3.5 SUMMARY

In this chapter introduction to the SLAM problem has been given. Main aspects of SLAM have been discussed and its two main parts, front-end and back-end have been explained in more details. SLAM front-end deals interpreting sensor data and forming pose constraints used in the SLAM back-end. It consists of odometry algorithm, loop closing detection algorithm, pose constraint calculation algorithm and sometimes the active component which influences the robot's movement in order to allow more loop closings. SLAM back-end uses constraints from the SLAM front-end to optimize poses of map landmarks and robot trajectory. Two main groups of SLAM back-ends exist, one uses the filter approach based on prediction and measurement update steps, while the other uses least squares formulation of the SLAM problem.

Today graph optimization SLAM back-ends prevail in the state-of-the-art SLAM solutions. However, in the following three chapters three scientific contributions will be presented which prove that by using new theoretical advancements in the filter theory, filtering based SLAM back-ends can again reclaim their role as best option for modern SLAM solutions. The next chapter will introduce a filter that served as a core for SLAM solution developed within this thesis. It is called the Exactly Sparse Delayed State Filter (ESDSF) and it uses best characteristics of both PF and SEIF SLAM back-ends. It will be coupled with a SLAM front-end which represents environment using planar features, thus ensuring low memory complexity and fast map rebuilding in case of a trajectory update.

4

Fast active planar 3D SLAM based on exactly sparse delayed state filter

In this chapter a complete active SLAM solution is presented. The SLAM back-end algorithm responsible for trajectory estimation is based upon Exactly Sparse Delayed State Filter (ESDSF) derived in [44]. It is a pose graph SLAM back-end which, as explained in Sec. 3.4.1, allows trajectory estimation independent of the environment map. This drastically reduces the state space of the filter and allows for faster computation of the update step. The derived SLAM front-end represents the first scientific contribution of this thesis and is published in [129]. It is based on segmenting planar surfaces from the point clouds obtained with 3D LIDAR. The extension to the SLAM front-end which allows it to work as an active SLAM algorithm is published in [130].

As explained in the Sec. 3.3.2, when attempting to calculate pose constraints between point clouds obtained from the 3D LIDAR, there is a problem of large amount of data required to be processed. Some solutions deal with this problem by segmenting point clouds into higher level features. The SLAM front-end presented in this chapter is developed on this idea. It is based on segmenting the 3D point clouds into planar surface segments. Planar surface segments have been chosen to represent the environment because they are prevalent in indoor and outdoor urban spaces. All planar surface segments extracted from one point cloud form a planar local map. These local maps are then used to build global planar map of the environment and by the Local Map Registration (LMR) algorithm to calculate the pose constraints. The global map of the environment is built by merging coplanar surface segments from the local maps into the global planar surfaces. Parameters of the global planar surfaces are estimated based on the probabilistic parameters of each planar surface segment they consist of. The derived active SLAM component works as an extension to the SLAM front-end without interfering with any of its components and allows the presented SLAM solution to work in combination with the exploration algorithm. It continuously checks the current uncertainty in the robot pose and when that uncertainty becomes too high it cancels the exploration and sends the robot to close the loop. Once the loop is closed it sends the robot back to the previously set exploration goal.

The most similar planar front-end SLAM solution to the one presented in [129] is presented in [72] where surface segments are extracted from point clouds acquired from a 2D rotating LIDAR. However, there are three main differences. First, in [72] planar surface segments are extracted from raw point clouds, while method presented in [129]

uses projection of the point clouds into lower dimensional space before extracting planar surface segments. This makes it by the order of the magnitude faster. Second, the method used for generating the pose constraint presented in [72] uses a global approach and does not benefit from initial pose estimate which makes it by the order of magnitude slower. Also it assumes that the orientation error obtained by plane registration is neglectable and minimizes only the translational errors to produce constant time updates. The solution presented in [129] minimizes both translation and orientation error which makes it more accurate. Third, planar surface segments in [72] are not merged in any way and are all added to the global map, while method presented in [129] performs merging of extracted planar surface segments that lie on the same plane which significantly reduces complexity of the global map making it suitable to represent large-scale environments.

The active SLAM solution presented in [130] is most similar to the one presented in [26] which uses particle filter SLAM back-end and also checks current robot's pose uncertainty to initiate the loop closing. However, there are two main differences between active SLAM solution presented in [130] and the one presented in [26]. First, SLAM solution presented in [130] is based on ESDSF which makes state choosing for loop closing much easier because of a more suitable state representation. Second, method presented in [130] does not use uncertainty as a fixed measure for cancelling the exploration since it can lead to a longer and repetitive loop closures.

The rest of the chapter is organized as follows. First, the main building blocks of the complete SLAM solution are described. Then ESDSF SLAM back-end and its implementation is explained in detail. After that SLAM front-end is presented and planar model and merging algorithm is explained. Afterwards, an active SLAM solution that works as an extension to the presented planar front-end is presented. Finally the experimental results using real world datasets are given.

4.1 THE OVERALL CONCEPT OF THE PROPOSED SLAM SYSTEM

The layout of the proposed active planar SLAM system with its key components is shown in Fig. 4.1. Hereafter, we briefly describe the functions of each component, while detailed descriptions are given in sections marked in the Fig. 4.1. *SLAM back-end* is responsible

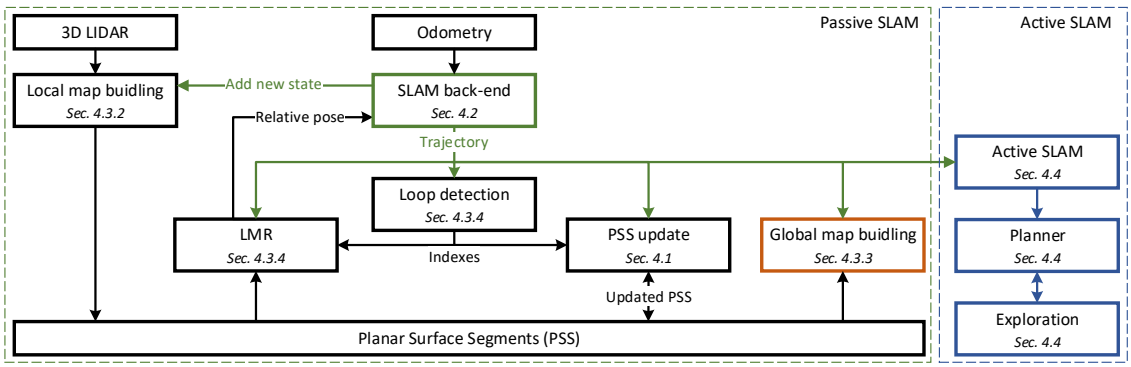

Figure 4.1: The overall concept of the proposed SLAM system.

for vehicle localization and trajectory building. Trajectory consists of discrete states X_i,

$i = 0, \ldots, n-1$, where each state is represented by the robot pose (3D position and orientation quaternion) expressed in the coordinate frame assigned to the first state. A new state X_k is added to the trajectory when the pose difference between the last added state in the trajectory and the current pose exceeds predefined thresholds. Once the new state is added two things occur: i) point cloud acquired in this state is sent to the *Local map building* module and ii) *Loop detection* algorithm begins to search for possible loop closings between the state X_k and other states in the trajectory. *Local map building* module then segments received point cloud into planar surface segments (PSS) and builds the local map. If the loop closing is detected between states X_k and X_i indexes (k, i) are sent to the *Local Map Registration (LMR)* module alongside with the current trajectory for predicting relative pose between the states X_k and X_i. The LMR uses SLAM trajectory and the segmented planar surface segments for the initial guess and precise estimation of the relative pose between the states X_k and X_i. Once the relative pose is calculated, it is reported back to the *SLAM back-end* which then incorporates it as a pose constraint into the pose graph and performs trajectory update. Updated trajectory and indexes (k, i) are sent to the *Planar surface segments update* module which uses new trajectory to update the existing planar surface segments in the local maps. After the local maps are updated, that trajectory is also sent to the *Global map building* module which then incorporates the updated planar surface segments together with the newly segmented planar surface segments into the global map. If no loop closing was detected the trajectory is sent to the *Global map building* module immediately after the trajectory augmentation which then adds the newly segmented planar surface segments to the global map. In the background, *SLAM back-end* continuously predicts the current robot pose using odometry information and sends it to the *Active SLAM* alongside with the current trajectory. If current pose uncertainty becomes high enough *Active SLAM* module starts to search for possible nearby states that are suitable for loop closing. Once such state is identified, pose of that state is sent to the *Planner* which stops the exploration end sends the robot to that state. After the loop closing is completed and trajectory updated the *Active SLAM* module signals the *Planner* to resume the goal previously set by the *Exploration* module.

4.2 ESDSF SLAM BACK-END

ESDSF based SLAM back-end combines best characteristics of the PF-SLAM and SEIF-SLAM back-ends. In general, ESDSF is a special form of EIF with the main advantage of having an exactly sparse information matrix. This means that it does not require sparsification step like the SEIF back-end. The exact sparsity results from the used motion model and from the fact that the ESDSF SLAM back-end estimates discrete robot trajectory, similarly to the PF-SLAM back-end. As explained in the Sec. 3.4.1, this allows map landmarks to be independent on each other, which in turn allows trajectory estimation independent of the environment map. However, ESDSF back-end in contrast to the PF-SLAM back-end and similarly to the EIF back-end estimates only one trajectory, making it much faster.

4.2.1 *State space construction*

Trajectory T_n in an ESDSF SLAM back-end consists of n discrete states X_i, $i = 1 \dots n$, and is represented by a single Gaussian random variable

$$T_n = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix}, \quad \begin{matrix} X_i \sim \mathcal{N}(\mu_{X_i}, \Sigma_{X_{i,i}}) = \mathcal{N}(\eta_{X_i}, \Lambda_{X_{i,i}}) \\ T_n \sim \mathcal{N}(\mu_n, \Sigma_n) = \mathcal{N}(\eta_n, \Lambda_n) \end{matrix}, \tag{4.1}$$

where μ_{X_i} and $\Sigma_{X_{i,i}}$ are mean and covariance of the state X_i, while μ_n and Σ_n are mean and covariance of the trajectory T_n, respectively. As in EIF, the equivalent representation of the Gaussian distribution in the information form is given by the relation $\eta = \Sigma^{-1}\mu$ and $\Lambda = \Sigma^{-1}$, thus, for example, the information vector and information matrix of the state X_i are given by $\eta_{X_i} = \Sigma^{-1}_{X_{i,i}}\mu_{X_i}$ and $\Lambda_{X_{i,i}} = \Sigma^{-1}_{X_{i,i}}$, respectively. As shown in [44] the information matrix Λ_n of the ESDSF trajectory T_n has a sparse tridiagonal structure

$$\Lambda_n = \begin{bmatrix} \Lambda_{X_{n,n}} & \Lambda_{X_{n,n-1}} & \\ \Lambda_{X_{n-1,n}} & \Lambda_{X_{n-1,n-1}} & \Lambda_{X_{n-1,n-2}} \\ & \Lambda_{X_{n-2,n-1}} & \Lambda_{X_{n-2,n-2}} \\ & \vdots & \vdots \end{bmatrix}, \tag{4.2}$$

which is the result of using the motion model and trajectory augmentation equations described in the sequel. Sparsity of the information matrix is the key advantage of ESDSF, since it enables fast computation of the matrix inverse using specially designed sparse-matrix solvers. The main difference between ESDSF information matrix and SEIF information matrix is that in ESDSF most of the values are exactly zero and hence no sparsification step is necessary.

Each state X_i consists of a position and orientation that the robot had at the time when the state was added to the trajectory. Whenever a new state X_i is added to the trajectory, measurement z_i is taken from the sensor used to map the environment. Although several different rotation representations exist, as detailed in Sec. 2.2, for the SLAM solution presented in this chapter orientation of the robot is represented using quaternions. This means that μ_{X_i} is a 7×1 vector

$$\mu_{X_i} = \begin{bmatrix} x & y & z & q_w & q_x & q_y & q_z \end{bmatrix}^T, \tag{4.3}$$

where (x, y, z) represents robot's position and quaternion $q = (q_w, q_x, q_y, q_z)$ its orientation in coordinate frame assigned to X_1.

4.2.2 *Motion model*

While the robot moves, its current pose is estimated using the same motion model as in EKF SLAM back-end. The motion model is described as the first order non-linear Markov process

$$X_{n+1} = f(X_n, \Omega_n, w_n), \tag{4.4}$$

where X_n represents the last state in the trajectory T_n, X_{n+1} represents the current robot pose, w_n represents zero-mean white Gaussian noise with covariance Q_n, while Ω_n stands

for robot displacement between X_n and X_{n+1} obtained from odometry. Although, odometry is part of the SLAM front-end, it its implementation is explained here since in the present SLAM solution no visual or laser odometry solution was used. Instead, fusion of information from encoders placed on robot's wheels and IMU were used to estimate robot's translational velocities

$$v = \begin{pmatrix} v_x & v_y & v_z \end{pmatrix} \tag{4.5}$$

and rotational velocities

$$\omega = \begin{pmatrix} \omega_x & \omega_y & \omega_z \end{pmatrix} \tag{4.6}$$

relative to the robot's body coordinate system. Both v and ω are represented with quaternions where the vector part of a quaternion corresponds to a direction of motion or axis of rotation, respectively, and its length to a speed amplitude. Kinematics of such system can be described by

$$\begin{pmatrix} 0 \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = q \begin{pmatrix} 0 \\ v_x \\ v_y \\ v_z \end{pmatrix} q^{-1} \tag{4.7}$$

$$\dot{q} = \frac{1}{2} q * \omega \tag{4.8}$$

$$\dot{q} = M \cdot q \tag{4.9}$$

$$\begin{pmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{pmatrix} = \begin{pmatrix} 0 & -\frac{\omega_0}{2} & -\frac{\omega_1}{2} & -\frac{\omega_2}{2} \\ \frac{\omega_0}{2} & 0 & \frac{\omega_2}{2} & -\frac{\omega_1}{2} \\ \frac{\omega_1}{2} & -\frac{\omega_2}{2} & 0 & \frac{\omega_0}{2} \\ \frac{\omega_2}{2} & \frac{\omega_1}{2} & -\frac{\omega_0}{2} & 0 \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix}, \tag{4.10}$$

where $[x\ y\ z]$ is the vehicle's position and $[q_0\ q_1\ q_2\ q_3]$ its orientation quaternion in a global frame. In (4.8) "$*$" denotes quaternion multiplication while "$\cdot$" in (4.9) denotes matrix multiplication which is then expanded in (4.10). Now we can discretize the vehicle's kinematics model (4.7)-(4.8) using Euler method and describe its uncertainty with additive Gaussian white process noise w_n with mean value 0 and covariance Q_n. In this way, the non-linear first order Markov process is obtained and used as ESDSF motion model (4.4).

4.2.3 *Prediction step*

Whenever new odometry data becomes available and the current robot pose is estimated using (4.4), the prediction step of ESDSF is triggered. ESDSF prediction consists of two sub steps: (i) augmentation of the trajectory T_n with X_{n+1}, and, conditionally, (ii) marginalization of X_n that is subject to the pose difference between X_n and X_{n-1}.

AUGMENTATION. The augmentation step always happens immediately after the current robot pose X_{n+1} is estimated using the motion model (4.4). During augmentation, trajectory

T_n is augmented with X_{n+1} and thus becomes T_{n+1}. Before the augmentation of T_n, we can write the distribution of all the states in the information form as

$$p(X_n, m|z_{1\dots n}, u_{1\dots k}) = \mathcal{N}^{-1}\left[\begin{pmatrix} \eta_{X_n} \\ \eta_m \end{pmatrix}, \begin{pmatrix} \Lambda_{X_nX_n} & \Lambda_{X_nm} \\ \Lambda_{mX_n} & \Lambda_m \end{pmatrix}\right], \tag{4.11}$$

where $z_{1\dots n}$ denotes the history of all measurements, $u_{1\dots k}$ stands for history of all odometry data, and m represents all trajectory states except the last one, i.e., $m = \{X_1 \dots X_{n-1}\}$. Note that usually there are more odometry data than the states in the trajectory, hence we use a different index. After the augmentation with the state X_{n+1}, using Markov first order assumption and the fact that poses and measurements are not correlated, we have

$$\begin{aligned} &p(X_{n+1}, X_n, m|z_{1\dots n}, u_{1\dots k+1}) = \\ &= p(X_{n+1}|X_n, m, z_{1\dots n}, u_{1\dots k+1})\; p(X_n, m|z_{1\dots n}, u_{1\dots k+1}) \\ &= p(X_{n+1}|X_n, u_{1\dots k+1})\; p(X_n, m|z_{1\dots n}, u_{1\dots k}) \\ &= \mathcal{N}^{-1}(\eta_{n+1}, \Lambda_{n+1})\,. \end{aligned} \tag{4.12}$$

Final expressions for η_{n+1} and Λ_{n+1} are

$$\eta_{n+1} = \begin{bmatrix} Q_n^{-1}(\mu_{X_{n+1}} - F_n\mu_{X_n}) \\ \eta_{X_n} - F_n^T Q^{-1}(\mu_{X_{n+1}} - \mu_{X_n}) \\ \eta_m \end{bmatrix} \tag{4.13}$$

$$\Lambda_{n+1} = \begin{bmatrix} \boxed{Q_n^{-1}} & \boxed{-Q_n^{-1}F_n} & 0 \\ \boxed{-F_n^T Q_n^{-1}} & \Lambda_{X_{n,n}} + F_n^T Q_n^{-1} F & \Lambda_{X_nm} \\ 0 & \Lambda_{mX_n} & \Lambda_{mm} \end{bmatrix}, \tag{4.14}$$

where F_n stands for the Jacobian of the motion model (4.4). From (4.14) we can see that augmenting the trajectory T_n with the new state X_{n+1} requires only the addition of three new blocks, marked with rectangles, to the information matrix. All the other elements remain unchanged and sparsity is preserved.

When augmentation of the trajectory is complete, we check the pose difference between X_n and X_{n-1}. If the difference is larger than the predefined threshold, we conclude that the measurement associated with X_n provides new information and we keep both X_n and X_{n+1} within the state space and the prediction step ends. When new odometry data becomes available, the new robot pose is estimated based on X_{n+1} and the pose difference is then checked between X_{n+1} and X_n. Otherwise, we proceed with the marginalization step described in the sequel.

Marginalization. If the difference between X_n and X_{n-1} is smaller than the predefined threshold, state X_n is marginalized from the trajectory and replaced by X_{n+1}, i.e., $X_n \leftarrow X_{n+1}$. Marginalization of the state from the trajectory equals marginalizing a multivariate Gauss distribution. In general, we can write the multivariate Gauss distribution as

$$p\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}; \mu, \Sigma\right), \quad \begin{matrix} x \sim \mathcal{N}(\mu_x,\ \Sigma_{xx}) \\ y \sim \mathcal{N}(\mu_y,\ \Sigma_{yy}) \\ z \sim \mathcal{N}(\mu_z,\ \Sigma_{zz}) \end{matrix}, \tag{4.15}$$

where

$$\Sigma = \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} & \Sigma_{xz} \\ \Sigma_{yx} & \Sigma_{yy} & \Sigma_{yz} \\ \Sigma_{zx} & \Sigma_{zy} & \Sigma_{zz} \end{bmatrix} = \Lambda^{-1} = \begin{bmatrix} \Lambda_{xx} & \Lambda_{xy} & \Lambda_{xz} \\ \Lambda_{yx} & \Lambda_{yy} & \Lambda_{yz} \\ \Lambda_{zx} & \Lambda_{zy} & \Lambda_{zz} \end{bmatrix}^{-1}. \tag{4.16}$$

For example, if we want to marginalize y, we need to solve the following integral

$$p(x,z) = \int p\left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}; \mu, \Sigma\right) dy = \mathcal{N}^{-1}(\bar{\eta}, \bar{\Lambda}), \tag{4.17}$$

where $\bar{\eta}$ and $\bar{\Lambda}$ are equal to

$$\bar{\eta} = \eta_\alpha - \Lambda_{\alpha\beta}\Lambda_\beta^{-1}\eta_\beta \tag{4.18}$$

$$\bar{\Lambda} = \Lambda_\alpha - \Lambda_{\alpha\beta}\Lambda_\beta^{-1}\Lambda_{\beta\alpha} \tag{4.19}$$

with

$$\begin{aligned} \eta_\alpha &= \begin{bmatrix} \eta_x \\ \eta_z \end{bmatrix}, \ \Lambda_\alpha = \begin{bmatrix} \Lambda_{xx} & \Lambda_{xz} \\ \Lambda_{zx} & \Lambda_{zz} \end{bmatrix} \\ \Lambda_{\alpha\beta} &= \begin{bmatrix} \Lambda_{xy} \\ \Lambda_{zy} \end{bmatrix}, \ \Lambda_\beta = \Lambda_{yy}. \end{aligned} \tag{4.20}$$

Concretely, for the case of ESDSF we have to marginalize X_n from T_{n+1}

$$\begin{aligned} &p(X_{n+1}, m | z_{1\ldots n+1}, u_{1\ldots k+1}) \\ &\quad = \int p(X_{n+1}, X_n, m | z_{1\ldots n+1}, u_{1\ldots k+1}) dX_n \\ &\quad = \mathcal{N}^{-1}(\bar{\eta}_n, \bar{\Lambda}_n). \end{aligned} \tag{4.21}$$

To find $\bar{\eta}_n$ and $\bar{\Lambda}_n$, we use (4.18) and (4.19), which yield for elements of (4.20) the following formulae

$$\eta_\alpha = \begin{bmatrix} \eta_{X_{n+1}} \\ \eta_m \end{bmatrix}, \ \Lambda_\alpha = \begin{bmatrix} \Lambda_{X_{n+1},n+1} & \Lambda_{X_{n+1}m} \\ \Lambda_{mX_{n+1}} & \Lambda_{m,m} \end{bmatrix}, \ \Lambda_{\alpha\beta} = \begin{bmatrix} \Lambda_{X_{n+1},n} \\ \Lambda_{mX_n} \end{bmatrix}, \ \Lambda_\beta = \Lambda_{X_{n,n}}. \tag{4.22}$$

These equations can be further simplified, because after the trajectory augmentation with X_{n+1}, states X_n and X_{n+1} are only connected via neighboring states, which means that matrices $\Lambda_{X_{n+1}m}$ and $\Lambda_{mX_{n+1}}$ are zero matrices and only a single block in Λ_{mX_n} is different from zero. Taking this into account and using final expressions for η_{n+1} and Λ_{n+1} given by equations (4.13) and (4.14) expressions for $\bar{\eta}_n$ and $\bar{\Lambda}_n$ can be obtained

$$\begin{aligned} \bar{\eta}_n &= \begin{bmatrix} Q_n^{-1}(\mu_{X_{n+1}} - F_n\mu_{X_n}) \\ \eta_m \end{bmatrix} - \begin{bmatrix} -Q_n^{-1}F_n \\ \Lambda_{mX_n} \end{bmatrix} \alpha_n^{-1}(\eta_{X_n} - F_n^T Q_n^{-1}(\mu_{X_{n+1}} - F_n\mu_{X_n})) \\ &= \begin{bmatrix} Q_n^{-1}F_n\alpha_n^{-1}\eta_{X_n} + \beta_n(\mu_{X_{n+1}} - F_n\mu_{X_n}) \\ \eta_m - \Lambda_{mX_n}(\eta_{X_n} - F_n^T Q_n^{-1}(\mu_{X_{n+1}} - F_n\mu_{X_n})) \end{bmatrix} \end{aligned} \tag{4.23}$$

$$\begin{aligned} \bar{\Lambda}_n &= \begin{bmatrix} Q_n^{-1} & 0 \\ 0 & \Lambda_{mm} \end{bmatrix} - \begin{bmatrix} -Q_n^{-1}F_n \\ \Lambda_{mX_n} \end{bmatrix} \alpha_n^{-1} \begin{bmatrix} -F_n^T Q_n^{-1} & \Lambda_{X_n m} \end{bmatrix} \\ &= \begin{bmatrix} \beta_n & Q_n^{-1}F_n\alpha_n^{-1}\Lambda_{X_n m} \\ \Lambda_{mX_n}\alpha_n^{-1}F_n^T Q_n^{-1} & \Lambda_{mm} - \Lambda_{mX_n}\alpha_n^{-1}\Lambda_{X_n m} \end{bmatrix}, \end{aligned} \tag{4.24}$$

where

$$\begin{aligned}\alpha_n =& \Lambda_{X_{n,n}} + F_n^T Q_n^{-1} F_n \\ \beta_n =& Q_n^{-1} - Q_n^{-1} F_n (\Lambda_{X_{n,n}} + F_n^T Q_n^{-1} F_n)^{-1} F_n^T Q_n^{-1} \\ =& (Q_n + F_n \Lambda_{X_{n,n}}^{-1} F_n^T)^{-1}.\end{aligned}$$

Taking into account that Λ_{mX_n} has only a single non-zero block, we can see from (4.24) that, similarly to augmentation, only four blocks of the information matrix Λ_n need to be changed during the marginalization. Once marginalization is complete, $\overline{\Lambda}_n$ and $\overline{\eta}_n$ become the new Λ_n and η_n, respectively, and $X_n \leftarrow X_{n+1}$, which then concludes the prediction step.

4.2.4 *Measurement model and update*

Update in the ESDSF SLAM is triggered every time the loop closing is detected between two trajectory states X_i and X_j. The measurement model in the ESDSF SLAM system is given in the form of a relative pose between states X_i and X_j

$$Z = h(X_i, X_j) + v,\ v \sim \mathcal{N}(0, R_{ij}), \tag{4.25}$$

where v represents measurement noise and is assumed to be a white zero-mean Gaussian with covariance matrix R_{ij}. The relative pose describes rotation q_{ij}^L and translation t_{ij}^L of the sensor, which records measurements between poses that sensor had when it recorded data in states X_i and X_j.

$$q_{ij} = q_i^{-1} q_j \tag{4.26}$$

$$t_{ij} = q_i^{-1}(t_j - t_i) q_i \tag{4.27}$$

$$h \left\{ \begin{array}{c} q_{ij}^L = q_{RL}^{-1} q_{ij} q_{RL} \\ t_{ij}^L = q_{RL}^{-1}(q_{ij} t_{RL} q_{ij}^{-1} + t_{ij} - t_{RL}) q_{RL} \end{array} \right\}, \tag{4.28}$$

where q_{RL} and t_{RL} describe the relative pose between coordinate frames of the LIDAR sensor and the vehicle. The measurement is obtained from a Local Map Registration (LMR) algorithm based on the saved measurements z_i and z_j.

The update equations of ESDSF are the same as EIF update equations

$$\eta_n' = \eta_n + H^T R^{-1}(z_{i,j} - h(X_i, X_j)) \tag{4.29}$$

$$\Lambda_n' = \Lambda_n + H^T R^{-1} H \tag{4.30}$$

where H is the measurement Jacobian and $z_{i,j}$ is measurement sent by the LMR. However, since h depends only on X_i and X_j, the measurement Jacobian H has a sparse structure

$$H_{n+1} = \left[\cdots \quad 0 \quad \cdots \quad \frac{\partial h}{\partial X_i} \quad \cdots \quad 0 \quad \cdots \quad \frac{\partial h}{\partial X_j} \quad \cdots \quad 0 \right].$$

This means that the update always affects only four blocks in the information matrix that share information associated to X_i and X_j, thus making it constant time. The problem is that after the update, vector μ_n has to be calculated as $\mu_n = \Lambda_n^{-1} \eta_n$, which is not constant time. However, this is drastically speeded up due to the sparsity of the information matrix.

4.2.5 *Covariance estimation and computational complexity analysis*

In order to perform both the prediction and the update, the ESDSF uses relative pose estimated between different time instances. The prediction step relies on the relative pose between two consecutive states determined by the odometry module, while the update step uses the relative pose evaluated by the LMR module between the two states for which the loop closing is detected. To incorporate these measurements, ESDSF needs information about their uncertainties, where Q_n represents the odometry uncertainty used in the prediction, while R_n represents uncertainty of the LMR algorithm used in the update.

Since majority of sensors and relative pose estimation algorithms rely on Euler angle representation in order to associate the uncertainties in quaternion representation, the unscented transform (UT) presented in [131] is used. The UT algorithm first takes measurement and the associated covariance matrix in Euler angles as input, and then applies the nonlinear transformation function resulting with the quaternion representation and the new covariance matrix.

Considering the performance of ESDSF as a SLAM back-end, computation complexity of each step has to be taken into account. The prediction step of ESDSF is constant-time similar to ESDSF, since it always changes the same number of blocks in the information matrix. In the case when the augmentation is not followed by marginalization, three new blocks are added, while in the case when marginalization is performed, only three existing blocks are changed. However, μ_{X_n} is necessary to complete the calculation of the new information vector η_{n+1}. The straightforward way to get state μX_n would be to extract it from the trajectory T_n, which requires computing $\mu_n = \Lambda_n^{-1}\eta_n$. This is a computationally demanding operation, due to the inversion of the information matrix, and should be avoided. Since after the update, states permanently added to the trajectory do not change until the next update, an auxiliary vector μ_n^{aux} can be constructed which stores states permanently added between the updates. After the update is performed, vector μ'_n is calculated and copied into μ_n^{aux} $\mu_n^{\text{aux}} = \mu'_n$ and new states are continuously added in the μ_n^{aux} until the next update. This way the history of the expectancy of all the states is kept, which can then be exploited in the prediction step with no need for inversion. Although in the prediction only μ_{X_n} is needed, the update step requires the knowledge of the entire μ_n. After the loop closing between states X_i and X_j is detected, their respective poses are required in order to calculate measurement prediction used in the update step and in the LMR algorithm as the initial guess. This means that before every update $\mu_n = \Lambda_n^{-1}\eta_n$ would again need to be calculated. To avoid the calculation of μ_n, μ_{X_i} and μ_{X_j} are simply obtained from the auxiliary vector μ_n^{aux}. Since it cannot be known upfront which states will be included in the update, history of all the states needs to be kept.

By using the auxiliary vector μ_n^{aux} prediction and update steps are kept constant time regardless of the number of states in the trajectory. However, after the update is done, updated state estimate μ'_n has to be calculated using new η'_n and Λ'_n, in this step calculation of $\Sigma'_n = \Lambda_n'^{-1}$ cannot be avoided. Nevertheless it can be speeded up drastically by the use of algorithms that exploits characteristics of the sparse matrices.

The last computational problem arises from the usage of unit quaternions. When using quaternions for orientation instead of Euler angles, each quaternion in T_n must remain

normalized

$$\sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} = 1 \tag{4.31}$$

After the prediction step quaternions will remain normalized. However, using the equations (4.29) and (4.30) during the update step, will result in information vector η'_n and information matrix Λ'_n. When μ'_n is calculated using the equation $\mu'_n = \Lambda'^{-1}_n \eta'_n$ quaternions representing orientations in μ'_n will no longer remain unit quaternions. This requires normalizing each one. After the normalization the information vector η'_n must also be recalculated using equation $\eta'_n = \Lambda'_n \mu_n^{\text{norm}}$. Besides time required to do this, the main problem is the information matrix which remains the same, although the information vector has changed.

Nevertheless, due to their numerical stability, ability to avoid the gimbal lock problem and easier composition (i.e. there is no requirement to ensure values remain between $-\pi$ and π) quaternions were chosen for the orientation representation. This analysis concludes the description of the implemented ESDSF SLAM back-end. In the next section SLAM front-end and its key components will be discussed in detail.

4.3 SLAM FRONT-END BASED ON PLANAR SURFACES

Algorithms that segment point clouds into planar surface segments can be divided into organized and unorganized, based on the type of used point clouds, and structured and unstructured, based on the environments they can work on. In [132] two different algorithms are presented i) a subwindow based region growing (SBRG) algorithm for structured environments and ii) a hybrid region growing (HRG) algorithm for unstructured environments. Both SBRG and HRG work only on organized point clouds which are first divided into subwindows and then classified as planar or non planar based on their shape. Only planar subwindows are used in SBRG while both planar and non planar subwindows are used in HRG. In [133] Cached Octree Region Growing (CORG) algorithm is presented which computes planar segments from unorganized point clouds in both structured and unstructured environments. The main idea of CORG is to accelerate the neighbour searching algorithm in the octree by requiring a single nearest neighbour search trial for each point in the octree. As a result, a compromise is made between memory and speed by caching the indices of the nearest neighbours searched for each point.

In context of SLAM, some of the earliest 3D SLAM solutions based on planar segments used a mobile robot equipped with a rotating 2D LIDAR producing three dimensional point clouds. The paper [37] presents a feature-based SLAM approach based on the Extended Kalman Filter (EKF), where the scans are directly converted into a planar representation composed of polygons and plane parameters with associated uncertainty in the framework of Symmetries and Perturbation model (SPmodel). The resulting maps are very detailed and compact but the approach is computationally very demanding and therefore not suitable for real-time applications. Work described in [36] is interested only in the local aspect of mapping, doing sequential surface capture of the workspace, while global pose corrections are propagated on-line in an elastic graph with a bounded number of elastic sub-maps. A feature based graph SLAM that uses rectangles in order to build a global map of indoor environments is presented in [71]. Algorithm allows extraction of rectangles from LIDAR measurements even in conditions of only partial visibility.

Besides LIDAR based SLAM solutions that use planar features, there are several SLAM solutions based on dense point clouds obtained from RGB-D sensors [62, 65, 66, 67, 68] and by dense multi-view stereo reconstruction [62, 63]. However, these methods are different from LIDAR based approaches due to significantly smaller operating range and field of view.

The planar SLAM front-end presented in [129] uses the modified version of point cloud segmentation algorithm presented in [134] and modified version of point cloud registration algorithm presented in [135]. The main reason for choosing these two algorithms is because they work on unorganized point clouds as input and achieves comparable accuracy to the techniques based on organized point clouds which makes them by the order of magnitude faster. The point cloud segmentation algorithm from [134], was originally developed for RGB-D sensors and was now adapted to operate on full field of view 3D LIDAR point clouds. This was done by dividing and projecting full FOV 3D point clouds onto three image planes which allows fast 2.5D point cloud segmentation based on recursive 2D Delaunay triangulation and region merging. Point cloud registration algorithm presented in [135] has been changed to include initial guess of the relative pose from the SLAM trajectory. Also, its planar segment model and registration algorithm has been adapted to take into account both the uncertainty model and 360° FOV of 3D LIDAR. By doing this, the number of outliers in pose constraint calculation has been significantly reduced and its process speed has been increased at the same time.

4.3.1 *Notation used in this chapter*

Before proceeding with the detailed explanation of the developed planar SLAM front-end, we give and overview of the commonly used notation for easier reference.

- P_l - Point cloud associated with l-th trajectory state
- M_j - Local map that consists of planar surface segments segmented from j-th point cloud
- S_j - Coordinate frame of j-th local map (the same as coordinate frame of j-th point cloud)
- $(R_{i,j}, t_{i,j})$ - Rotation matrix and translation vector which transform S_j into S_i
- ${}^{g}F_{i,j}$ - j-th planar surface segment in i-th local map which belongs to g-th global planar surface
- $S_{F_{i,j}}$ - Local coordinate frame of j-th planar surface segment in i-th local map
- $({}^{F}R_{i,j}, {}^{F}t_{i,j})$ - Rotation matrix and translation vector which transform $S_{F_{i,j}}$ into S_i
- ${}^{F}n_{i,j}$ - Unit normal of $F_{i,j}$ expressed in $S_{F_{i,j}}$ (expected value is $[0\ 0\ 1]$)
- ${}^{F}\rho_{i,j}$ - Distance of $F_{i,j}$ from $S_{F_{i,j}}$ (expected value is 0)
- $\Sigma_{q_{i,j}}$ - Covariance matrix of perturbation vector q representing uncertainty of ${}^{F}n_{i,j}$ and ${}^{F}\rho_{i,j}$ in $S_{F_{i,j}}$

- ${}^{n}G_l$ - l-th global surface with assigned n-th local map
- S_{G_l} - Local coordinate frame of l-th global surface
- $({}^{G}R_l, {}^{G}t_l)$ - Rotation matrix and translation vector which transform S_{G_l} into coordinate frame of local map M_n assigned to ${}^{n}G_l$
- $({}^{G}R_l^0, {}^{G}t_l^0)$ - Rotation matrix and translation vector which transform S_{G_l} into coordinate frame S_0
- ${}^{G}n_l$ - Unit normal of G_l expressed in S_{G_l} (expected value is $[0\ 0\ 1]$)
- ${}^{G}\rho_l$ - Distance of G_l from S_{G_l} (expected value is 0)
- Σ_{G_l} - Covariance matrix of perturbation vector representing uncertainty of ${}^{G}n_l$ and ${}^{G}\rho_l$ in S_{G_l}

4.3.2 *Local map building*

Local maps consist of planar surface segments extracted from one point cloud. In this section, we first describe a method for detection of planar surface segments from 3D LIDAR point clouds and then give their mathematical description and uncertainty model. Local maps are used for creation of the global environment map and for calculating relative poses used as the pose constraints in the SLAM.

Detection of planar surface segments.

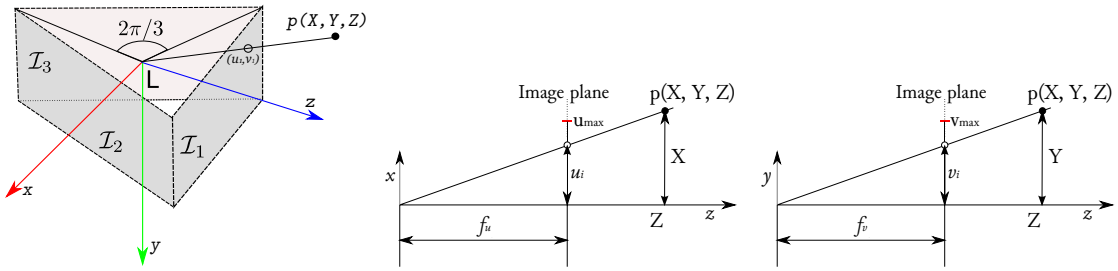

Figure 4.2: Point cloud division and projection onto three image planes: a) three image planes with their coordinate frame, b) determination of the horizontal pixel coordinate u_i of the point's p projection onto the image plane, c) determination of the vertical pixel coordinate v_i of the point's p projection onto the image plane.

The approach used for extracting planar surface segments is based on 2D Delaunay triangulation, which requires an appropriate 2D projection of the input point cloud. One possibility is to project the point cloud onto a cylindrical surface. However, projections of straight lines in 3D space onto a cylindrical surface are not straight lines. Therefore, triangles obtained by Dealunay triangulation applied to such projection don't represent triangular surfaces in reality. This is the reason why method presented here projects point clouds onto three image planes I_1, I_2 and I_3, each covering a field of view of 120° in horizontal direction, as shown in Fig. 4.2a. Point cloud projection is performed using the pinhole model. First,

two sets are defined: $s_n = \{0, \sin(2\pi/3), -\sin(2\pi/3)\}$ and $c_n = \{1, \cos(2\pi/3), \cos(2\pi/3)\}$. Then for each point $p(X, Y, Z)$ in the point cloud P_l the following equations are evaluated for $i = 1, 2, 3$:

$$x_i = -s_n(i)Z + c_n(i)X \tag{4.32}$$

$$y_i = Y \tag{4.33}$$

$$z_i = c_n(i)Z + s_n(i)X \tag{4.34}$$

$$u_i = f_u \frac{x_i}{z_i} + u_c \tag{4.35}$$

$$v_i = f_v \frac{y_i}{z_i} + v_c\,. \tag{4.36}$$

Equations (4.32) - (4.34) represent the transformation of the point $p(X, Y, Z)$ from the LIDAR coordinate frame into the coordinate frame of the image plane I_i, while the equations (4.35) and (4.36) represent the projection of the transformed point into the pixel (u_i, v_i) of the I_i. Values of u_{max} and v_{max} define the image resolution which affects the precision of the projection. Higher image resolution corresponds to greater projection precision. However, the processing speed drops with increase of resolution. Pixel (u_c,v_c) represents the center of the projection while (f_u, f_v) represent vertical and horizontal focal lengths of the pinhole model, respectively. Their values are determined based on the LIDAR FOV (Fig. 4.2b and 4.2c) in order to capture the whole scan. Value of u_c is always set at the horizontal middle of the image plane $(u_{max}/2)$ with horizontal focal length set to $f_u = u_c/\sqrt{3}$. Values of v_c and f_v are set according to the vertical FOV of the used LIDAR.

Let the point's p projection to the image plane I_i at the image pixel (u_i, v_i) be such that conditions $u_i >= 0$, $u_i <= u_{max}$, $v_i >= 0$ and $v_i <= v_{max}$ are satisfied. Then the point p belongs to I_i and the distance $r = \sqrt{X^2 + Y^2 + Z^2}$ of the point p from the projection center L is assigned to its image pixel. For points belonging to the same image plane whose projections fall at the same image pixel, r is set to the range of the closest point from L. In this way triplet (u_i, v_i, r) is formed for every pixel in the image plane I_i corresponding to a point in the point cloud, thus obtaining 2.5D input for segmentation method. The projected point clouds are then segmented into connected approximately planar subsets using a split-and-merge algorithm based on the approach proposed by [136], which consists of an iterative Delaunay triangulation method followed by region merging. An example of the point cloud segmentation to planar 3D surface segments is shown in Fig. 4.3. The obtained planar surface segments represent features which are used in the presented SLAM system for global map representation and trajectory estimation.

The parameters of the plane supporting a planar surface segment are determined by least-square fitting of the plane to supporting points of the segment. Each surface segment is assigned a reference frame S_F with an origin in the centroid of the supporting point set and z-axis parallel to the supporting plane normal. The orientation of the x-axis and y-axis in the supporting plane are defined by the eigenvectors of the covariance matrix Σ_p representing the distribution of the supporting points within this plane. The purpose of assigning reference frames to surface segments is to provide a framework for global map building explained in Sec. 4.3.3 and pose constraint estimation explained in Sec. 4.3.5.

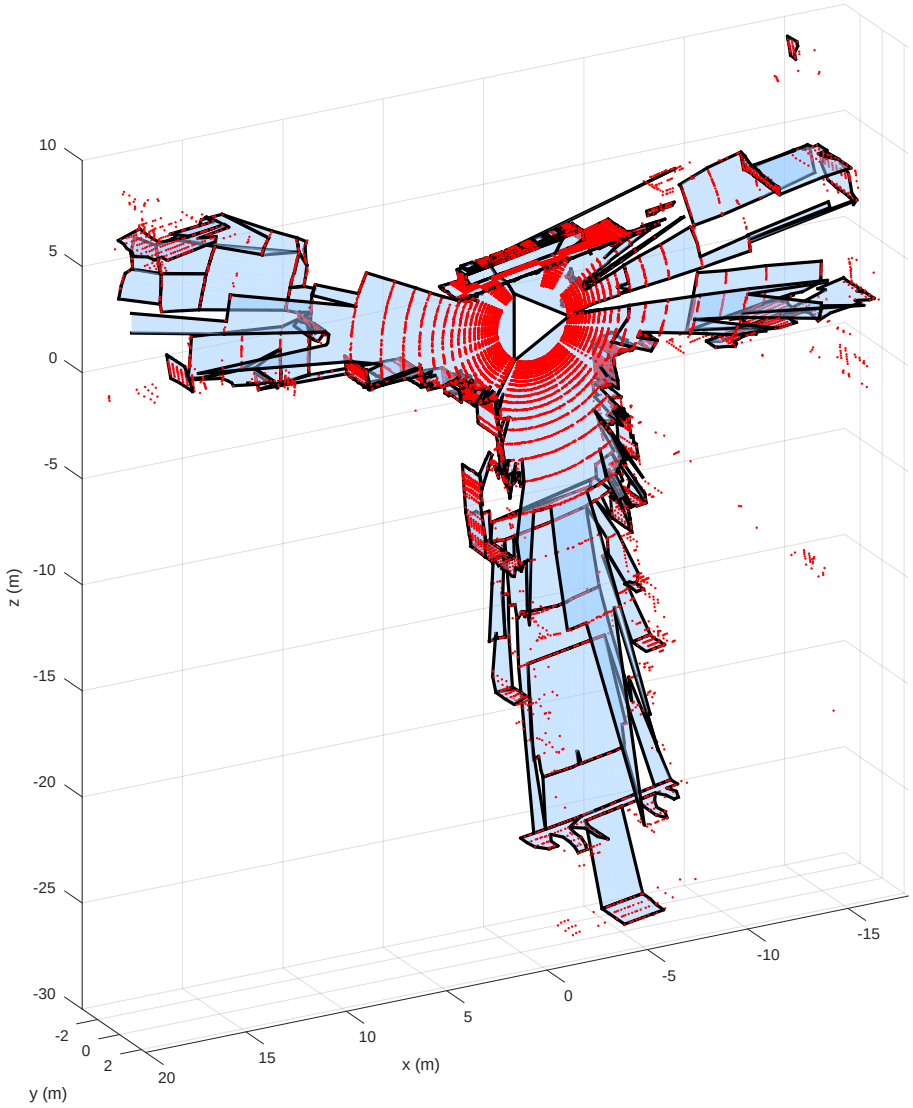

Figure 4.3: Local planar 3D map. Red dots represent points from the point cloud acquired by the LIDAR.

Representation of planar surface segments. Planar surface segments are represented by sets of 2D polygons in the same way as described in [135]. Each 2D polygon is defined with its outer and inner contours and with supporting 3D plane which is defined by the equation

$$({}^{F}n)^{\mathrm{T}} \cdot {}^{F}p = {}^{F}\rho, \tag{4.37}$$

where ${}^{F}n$ is the unit normal of the plane represented in the planar surface segment reference frame S_F, ${}^{F}\rho$ is the distance of the plane from the origin of S_F and ${}^{F}p \in \Re^3$ is an arbitrary point of the plain represented in S_F. The uncertainty of the supporting plane parameters is described by three random variables that form the disturbance vector $q = [s_x, s_y, r]^{\mathrm{T}}$. These three variables describe the deviation of the true plane parameters from the measured plane parameters. In the ideal case, where the measured plane is identical to the true plane, the true plane normal is identical to the z-axis of S_F, which means that ${}^{F}n = [0, 0, 1]^{\mathrm{T}}$, while ${}^{F}\rho = 0$. In a general case, however, the true plane normal deviates from the z-axis of S_F and this deviation is described by the random variables s_x and s_y, representing the deviation

in directions of the x-axis and y-axis of S_F respectively, as illustrated in Fig. 4.4 for the x direction.

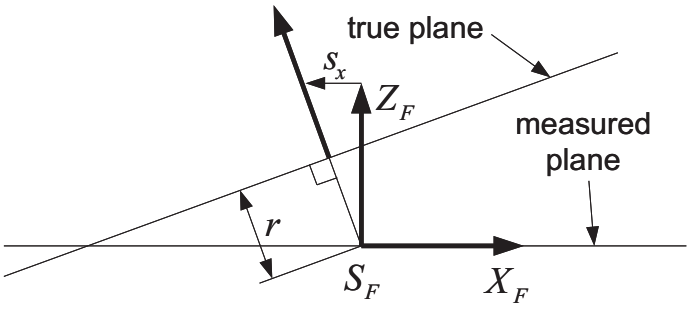

Figure 4.4: Deviation of the true plane from the measured plane.

The unit normal vector of the true plane can then be written as

$$^F n = \frac{1}{\sqrt{s_x^2 + s_y^2 + 1}} \begin{bmatrix} s_x & s_y & 1 \end{bmatrix}^{\mathrm{T}}. \tag{4.38}$$

The random variable r represents the distance of the true plane from the origin of S_F, i.e.

$$^F \rho_{i,j} = r. \tag{4.39}$$

Gaussian uncertainty model is used, where the disturbance vector q is assumed to be normally distributed with 0 mean and covariance matrix Σ_q. Covariance matrix Σ_q is a diagonal matrix with variances σ_{sx}^2, σ_{sy}^2 and σ_r^2 on its diagonal. Computation of covariance matrices Σ_q is explained in [137]. Finally, a planar surface segment denoted by the symbol F segmented from the point cloud P_l is associated with the quadruplet

$$F = \left(^F R,\ ^F t,\ \Sigma_q,\ \Sigma_p \right), \tag{4.40}$$

where $^F R$ and $^F t$ are respectively the rotation matrix and translation vector defining the pose of S_F relative to the coordinate system S_l of the point cloud P_l.

From this point on, it is important to distinguish between different planar surface segments, so notations shall be briefly explained for easier understanding. Whenever the point cloud P_l is segmented, a local map M_l is created. The local map has the coordinate frame S_l equal to the coordinate frame of P_l and consists of planar surface segments segmented from P_l. Every planar surface segment $F_{i,j}$ in M_l is identified by two indexes (i and j), where the value of index i represents the ID number of the planar surface segment in M_l and the value of index j identifies the local map which the planar surface segment belongs to (i.e. all planar surface segments contained within M_l have value of index $j = l$). According to this notation, planar surface segment $F_{i,j}$ is represented by quadruplet

$$F_{i,j} = \left(^F R_{i,j}, ^F t_{i,j}, \Sigma_{q_{i,j}}, \Sigma_{p_{i,j}} \right) \tag{4.41}$$

4.3.3 *Global map building*

In this section approach to building the environment global map which consists of global planar surfaces is presented. The reference coordinate frame of the global map is the same as

the coordinate frame of the local map M_1 associated with the first state X_1 in the trajectory. Each global planar surface joins all planar surface segments from the local maps that approximately lie on the same plane in the environment. That is why the global map has much less planar surface segments than the total number of segments in all local maps, which makes it faster to process and requires less memory to store. For example, entire floor, roof or wall is represented by only one global planar surface in the global map.

The global map is represented with a hierarchical graph. Nodes in that graph are: i) global planar surfaces ${}^{j}G_{g}$, at the highest level, where values of indexes g and j represent respectively the ID number of the global planar surface and the local map assigned to the g-th global planar surface, ii) local maps M_j, at the middle level, where value of index j represents the ID number of the local map, and iii) planar surface segments, at the lowest level ${}^{g}F_{i,j}$, where the value of index i represents the ID number of the planar surface segment in the local map M_j. The triplet (i, j, g) uniquely identifies the planar surface segment and its belonging to a certain local map and a certain global planar surface. For example ${}^{1}F_{3,4}$ means that ID of the planar surface segment is 3, and that it is part of the global planar surface 1 and the local map 4. If a planar surface segment does not belong to any global planar surface index g is omitted from the notation. Connections between planar surface segments and local maps are formed during the segmentation process and are kept unchanged afterwards.

Every global planar surface ${}^{n}G_l$, $l \in \{1, \ldots, N_G\}$, is defined by

$$ {}^{n}G_l = \left({}^{G}R_l,\ {}^{G}t_l,\ {}^{G}\Sigma_l,\ {}^{G}R_l^0,\ {}^{G}t_l^0 \right) \tag{4.42} $$

where N_G is the total number of global planar surfaces in the global map, ${}^{G}R_l$ and ${}^{G}t_l$ are respectively the rotation matrix and translation vector defining the transformation between local coordinate frame S_{G_l} of the global surface ${}^{n}G_l$ and the coordinate frame S_n of the local map M_n which is assigned to ${}^{n}G_l$, ${}^{G}\Sigma_l$ is the covariance matrix defining uncertainties of the perturbation vector of the unit-normal ${}^{G}n_l$ and the plane distance ${}^{G}\rho_l$ from the origin of S_{G_l}, and ${}^{G}R_l^0$ and ${}^{G}t_l^0$ are rotation and translation vector defining the transformation between S_{G_l} and S_0.

Updating of the global map occurs every time a new state is permanently added to the trajectory. The update process differs depending on whether or not loop closing has been detected with the newly added state. First, it shall be explained how the global map updating is performed if a new state is added to the trajectory and loop closing is not detected, and than additional steps that are performed if the loop closing is detected shall be described.

Global map update after trajectory augmentation without loop closing. Let's assume that there are $k-1$ states in the trajectory and that at time step k the state X_k was added. After point cloud P_k has been segmented, the local map M_k is created. At this point none of the planar surface segments from M_k is present in the global map. The next step is to determine pairs between the planar surface segments already included in the global map and the newly extracted segments from M_k. The exact way would be to try to match every planar surface segment from M_k with all previously extracted segments. However, this approach would be far too slow for real time application. Solution is to try to match the planar surface segments from M_k only with the planar surface segments from M_{k-1} and to update the global map accordingly. With this approach the accuracy

deterioration is neglectable if no loop closing is detected because it is reasonable to assume that the newly extracted planar surface segments in M_k mostly originate from the same planes as the surface segments extracted from M_{k-1}. So the first step in the global map update is to perform planar surface segments matching between M_k and M_{k-1}. Two planar surface segments are matched if they lie on approximately the same plane. This matching process is different than the segments matching when estimating pose constraints between local maps described in 4.3.5. While for generating pose constraint it is important to find pairs of planar surface segments between two local maps that are coplanar and overlap, in the case of building the global map, it is not necessary for planar surface segments to overlap. For example, if a vehicle is moving through a corridor, every new state will have one additional wall segment that needs to be connected with the previous segments. These segments do not overlap in the environment, but they do lie on the same plane and should represent one global planar surface. This is why only coplanarity condition is checked. The entire algorithm for checking if surfaces are coplanar is described in [135] and only final expressions and brief description are given here. In order to check if $(F_{i,k-1},F_{j,k})$ are coplanar, the Mahalanobis distance $d(e)$ is used

$$d(e) = e^{\mathrm{T}}(E\Sigma_{q_{j,k}}E^{\mathrm{T}} + CP_{k-1,k}C^{\mathrm{T}} + \Sigma_{q_{k-1,i}})^{-1}e, \tag{4.43}$$

where E represents the Jacobian matrix propagating the uncertainty of planar surface segment parameters and C represents the Jacobian matrix propagating the uncertainty of the transformation between two local maps (the final expressions for E and C are given in Appendix A.1), $P_{k-1,k}$ is the uncertainty of the transformation between local maps (calculated from the SLAM trajectory using unscented transform the same way as ${}^{I}P$ is calculated in Sec. 4.3.5) and e is the random variable given by

$${}^{F}\tilde{n}_{j,k} = \begin{bmatrix} {}^{F}\tilde{n}^{x}_{j,k} & {}^{F}\tilde{n}^{y}_{j,k} & {}^{F}\tilde{n}^{z}_{j,k} \end{bmatrix}^{T} = {}^{F}R^{\mathrm{T}}_{i,k-1}R_{k-1,k}{}^{F}R_{j,k}{}^{F}n_{j,k} \tag{4.44}$$

$${}^{F}\tilde{\rho}_{j,k} = ({}^{F}t_{j,k} - ({}^{F}t_{i,k-1} - t_{k-1,k})R_{k-1,k})^{F}R_{j,k}{}^{F}n_{j,k}) \tag{4.45}$$

$$e = \begin{bmatrix} {}^{F}\tilde{n}^{x}_{j,k} & {}^{F}\tilde{n}^{y}_{j,k} & {}^{F}\tilde{\rho}_{j,k} \end{bmatrix}^{T} \tag{4.46}$$

where ${}^{F}\tilde{n}_{j,k}$ and ${}^{F}\tilde{\rho}_{j,k}$ represent the expected values of ${}^{F}n_{j,k}$ and ${}^{F}\rho_{j,k}$ transformed into $S_{F_{i,k-1}}$ and $[{}^{F}\tilde{n}^{x}_{j,k},\ {}^{F}\tilde{n}^{y}_{j,k},\ {}^{F}\tilde{n}^{z}_{j,k}]$ represent x, y, z-coordinates of ${}^{F}\tilde{n}_{j,k}$ in $S_{F_{i,k-1}}$. The pair $(F_{i,k-1},F_{j,k})$ is considered coplanar if the following condition is satisfied

$$d(e) < \varepsilon \tag{4.47}$$

where ε is a measure of coplanarity calculated using χ^2-distribution with three degrees of freedom.

The problem with using only coplanar condition based on the Mahalanobis distance is that covariances (especially covariance P which describes uncertainty of transformation between two local maps) can become large due to the uncertainty in the current pose, i.e., when a vehicle travels on a difficult terrain and no loop closings are detected. That is why, in order to ensure global map accuracy, all surface pairs that pass condition (4.47) also have to pass an additional condition based on the absolute values of their parameters differences (e). This condition acts as a cut-off threshold in the matching process and ensures that the

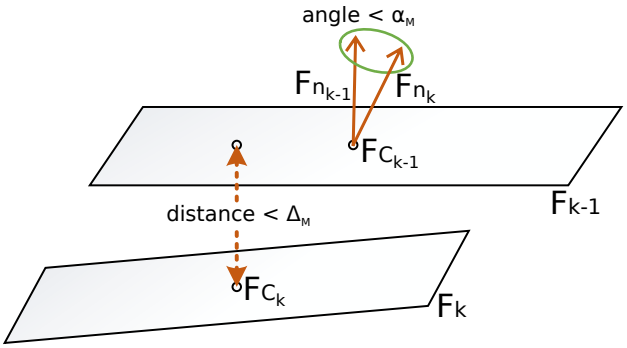

Figure 4.5: Second condition for matching planar surface segments.

angle between normals $\measuredangle({}^F\tilde{n}_{j,k}, {}^F n_{i,k-1})$ and distance $F\tilde{\rho}_{j,k}$ are smaller than the predefined thresholds α_M, Δ_M, respectively (Fig. 4.5). Mathematically, the condition can be expressed as

$$|cos^{-1}({}^F\tilde{n}_{j,k}{}^F n_{i,k-1})| < \alpha_M \qquad |F\tilde{\rho}_{j,k}| < \Delta_M. \tag{4.48}$$

After forming pairs of planar surface segments between M_{k-1} and M_k, all pairs are sorted into groups. Pairs belong to the same group if one or both of the following conditions are satisfied: i) pairs have one planar surface segment in common, and ii) both pairs have at least one planar surface segment in the same global planar surface. The result is a list L_F of N_L groups: $L_F = \{l_1 \dots l_{N_L}\}$. Each group in L_F contains planar surface segments from M_{k-1} and M_k.

Now for each group from L_F it is determined whether it represents a new global planar surface. Group l_c, $c \in \{1 \dots N_L\}$ represents a new global surface if none of the planar surface segments from l_c are already contained within another global planar surface. For every group that represents a new global planar surface ${}^n G_l$, parameters of ${}^n G_l$ are estimated and ${}^n G_l$ is added to the global map. The parameters of ${}^n G_l$ are estimated directly from the parameters of all planar surface segments contained within l_c. Therefore, before estimating the parameters of ${}^n G_l$, all parameters of all segments from the group l_c have to be transformed into the same coordinate frame $S_{F_{m,n}}$ of one of the planar surface segments from l_c. The transformation of the expected values of planar surface segment parameters is given by (4.44)-(4.46) and their covariance is

$$\tilde{\Sigma}_{q_{i,j}} = E\Sigma_{q_{i,j}}E^{\mathrm{T}} + CP_{n,i}C^{\mathrm{T}}. \tag{4.49}$$

Based on equations (4.46) and (4.49) the parameters of ${}^n G_l$ are estimated using maximum likelihood estimator. The problem which arises in the uncertainty model of plane perturbations is that uncertainty of z-coordinate of the normal is lost by imposing unit-normal constraint. To solve this issue, this unit-constraint

$${}^F\tilde{n}^z = \sqrt{1 - ({}^F\tilde{n}^x)^2 - ({}^F\tilde{n}^y)^2}$$

is added to the state model (4.46) and expanded state model is obtained $e^z = [{}^F\tilde{n}^x \; {}^F\tilde{n}^y \; {}^F\tilde{n}^z \; {}^F\tilde{\rho}]$. Unscented transform is used to determine its covariance Σ_{q^z}. Now maximum likelihood estimator is used to estimate ${}^G\tilde{n}_l$ and ${}^G\tilde{\rho}_l$ together with their covariances:

$$\Sigma_G = \left(\sum_{k=1}^{N_{l_c}} (\tilde{\Sigma}^{-1}_{q^z_k})\right)^{-1} \tag{4.50}$$

$$\left[{}^{G}\tilde{n}_l^x \quad {}^{G}\tilde{n}_l^y \quad {}^{G}\tilde{n}_l^z \quad {}^{G}\tilde{\rho}_l\right]^T = \Sigma_G\left(\sum_{k=1}^{N_{l_c}}(\tilde{\Sigma}_{q_k^z}^{-1}e_k^z)\right), \tag{4.51}$$

where ${}^{G}\tilde{n}_l$ is the normal of ${}^{n}G_l$ in $S_{F_{m,n}}$, ${}^{G}\rho_l$ is the distance of ${}^{n}G_l$ from $F_{m,n}$ in $S_{F_{m,n}}$ and N_{l_c} is the number of planar surface segments in l_c. At the end, we normalize estimated normal ${}^{G}\tilde{n}_l$.

The last four parameters that need to be estimated for ${}^{n}G_l$ are rotation matrices ${}^{G}R_l$ and ${}^{G}R_l^0$ and translation vectors ${}^{G}t_l$ and ${}^{G}t_l^0$ which transform the coordinate frame S_{G_l} into the coordinate frame S_n and S_0, respectively. In order to estimate ${}^{G}R_l$, the rotation matrix R_{min} that transforms unit normal ${}^{G}\tilde{n}_l$ into unit-normal $F_{n_{m,n}}$, has to be calculated. To preserve orientation of all planar surface segments, the rotation matrix R_{min}, which corresponds to minimum Euler angles, is determined using the following equation

$$R_{min} = I + [v]_\times + [v]_\times^2\frac{1-c}{s^2} \tag{4.52}$$

$$[v]_\times = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix}, \tag{4.53}$$

where $v = {}^{G}\tilde{n}_l^x - {}^{G}\tilde{n}_l^y$, $c = {}^{G}\tilde{n}_l^z$ and $s = \sqrt{({}^{G}\tilde{n}_l^x)^2 + ({}^{G}\tilde{n}_l^y)^2}$. Once the matrix R_{min} is calculated, the final rotation matrix ${}^{G}R_l$ can be determined as

$${}^{G}R_l = {}^{F}R_{m,n}R_{min}. \tag{4.54}$$

A vector ${}^{l}t_G$ can be calculated using the following equation

$${}^{G}t_l = {}^{F}R_{m,n}\begin{bmatrix} 0 \\ 0 \\ {}^{G}\tilde{\rho}_l \end{bmatrix} + t_{F_{m,n}}. \tag{4.55}$$

Finally, ${}^{G}R_l^0$ and ${}^{G}t_l^0$ can be easily calculated as

$${}^{G}R_l^0 = R_{0,n}{}^{G}R_l \tag{4.56}$$

$${}^{G}t_l^0 = R_{0,n}{}^{G}t_l + t_{0,n}, \tag{4.57}$$

Next step is to perform plane merging in order to reduce the number of planar surface segments in the global map. First, all contours of planar surface segments within l_c are transformed to S_{G_l}, and then the union of the transformed contours which are 2D polygons is performed. The contours are transformed to S_{G_l} by transforming their vertexes using the following transformation

$${}^{F}\tilde{p}_{i,j} = {}^{G}R_l^{-1}(R_{n,j}({}^{F}R_{i,j}{}^{F}p_{i,j} + {}^{F}t_{i,j}) + t_{n,j} - {}^{G}t_l), \tag{4.58}$$

where ${}^{F}p_{i,j}$ is the vertex of the contour of planar surface segment $F_{i,j}$ contained within l_c and ${}^{F}\tilde{p}_{i,j}$ represents its coordinates in the coordinate frame S_{G_l}. After all local planar surface segment from l_c have been transformed using equation (4.58), their normals will be aligned with ${}^{G}n_l$ and their contours will be expressed in S_{G_l}. The union of contours is done using 2D polygons which are generated from the transformed contours by simply taking their

x, y-coordinates. Finally, when all 2D polygons have been generated, union of polygons is performed using *Boost C++ libraries* [138]. *Boost* is used for creating the union because it offers fast performance and supports the union of multi-polygons and polygons with multiple holes. Once the union is complete, the resulting 2D polygon is assigned to the global planar surface ${}^{n}G_{l}$ as its contour.

For groups from L_F that have at least one planar surface segment ${}^{s}F_{i,j}$ which belongs to the global planar surface ${}^{n}G_{s}$, the only difference in the described process is that instead of transforming contour of the planar surface segment, the contour of the ${}^{n}G_{s}$ is transformed and used in the merging step. Here it becomes apparent why it is very important to assign a local map to the global planar surface since now all the equations used for transforming planar surface segments can be applied to the global planar surfaces by simply using ${}^{G}R_{s}$, ${}^{G}t_{s}$ and ${}^{G}\Sigma_{l}$ instead of ${}^{F}R_{i,j}$, ${}^{F}t_{i,j}$ and ${}^{G}\Sigma_{q_{i,j}}$ respectively. After the union has been created all the global planar surfaces used in the union are deleted because newly created global planar surface replaces them in the global map.

An example of the global map update after trajectory augmentation is shown in Fig. 4.6. Initially, there are two local maps M_1 and M_2. The groups generated after the matching phase are shown in Table 4.1. Three global planar surfaces are generated (${}^{1}G_1$, ${}^{2}G_2$ and ${}^{2}G_3$, Fig. 4.6a). When the state X_3 is added to the trajectory, the local map M_3 is segmented and after that the global map is updated by matching planar surface segments from M_2 and M_3. Only one segment from M_3 is matched, and the result of grouping is shown in Table 4.2. Since planar surface segment ${}^{3}F_{2,2}$ is already part of ${}^{2}G_3$ and ${}^{2}F_{3,2}$ is already part of ${}^{2}G_2$ the resulting global planar surface ${}^{1}G_4$ was estimated from ${}^{2}G_2$, ${}^{2}G_3$ and $F_{1,3}$. After the new global surface ${}^{1}G_4$ was generated, global surfaces ${}^{2}G_2$ and ${}^{2}G_3$ are deleted and the planar surface segments connected to them are connected to the ${}^{1}G_4$ (Fig. 4.6b).

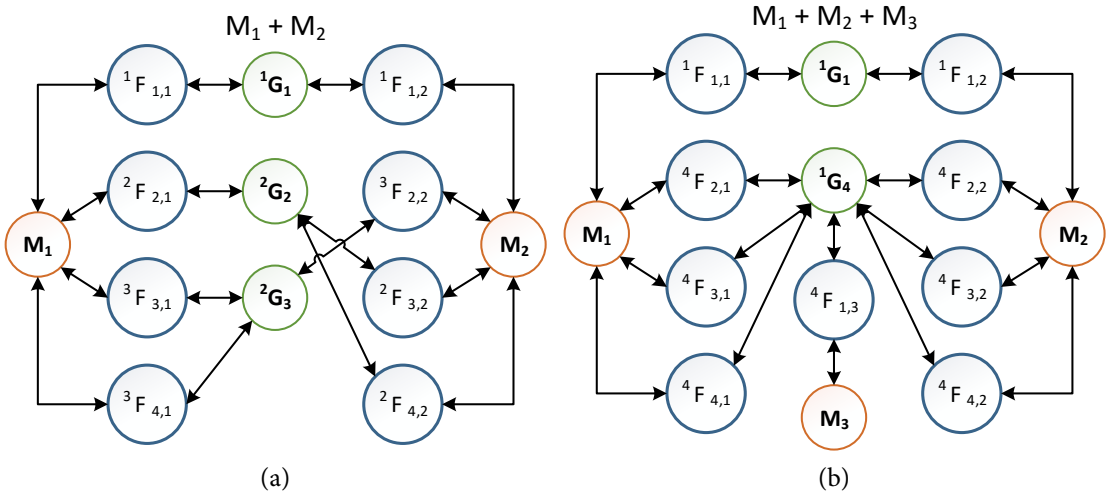

Figure 4.6: Examples of the global map update: a) after the update between M_1 and M_2, and b) after the update between M_2 and M_3.

GLOBAL MAP UPDATE AFTER LOOP CLOSING. If the loop closing is detected between a newly added state X_k to the the trajectory and an already existing trajectory state X_i, three additional steps are executed during the global map update:

Group	M_1 IDs	M_2 IDs
1	1	1
2	2	3,4
3	3,4	2

Table 4.1: Groups from matches between M_1 and M_2.

Group	M_1 IDs	M_2 IDs	M_3 IDs
1		2, 3	1

Table 4.2: Groups from matches between M_2 and M_3.

1. Updating planar surface segments parameters in M_k and M_i
2. Reformatting the global planar surfaces if necessary
3. Updating the global map by matching planar surface segments between M_k and M_i

The first step is initiated immediately after the trajectory is updated. The planar surface segments from M_k and M_i are matched using the same algorithm as if no loop closing is detected. Every matched pair is reported to the planar surface segments update algorithm which uses new SLAM trajectory to update their parameters. After the segments have been updated, the second step, i.e. reformatting the global planar surfaces, is performed. The global planar surface ${}^nG_l$ needs to be reformatted for two reasons: i) a planar surface segment within ${}^nG_l$ has been updated or, ii) one or more local maps containing planar surface segment that is part of ${}^nG_l$ are associated with trajectory states which poses were significantly changed. Reformatting the global planar surface ${}^nG_l$ ensures that all planar surface segments contained within ${}^nG_l$ satisfy the coplanarity conditions after the trajectory has been updated. The reformatting step is similar to the estimation of a new global planar surface. The difference is that, instead of matching planar surface segments from the local maps, planar surface segments contained within ${}^nG_l$ are matched with one-another instead. The result of the matching are groups of planar surface segments from ${}^nG_l$ that satisfy coplanarity conditions with one another. Although absolute poses of the trajectory states could significantly change due to the update, their relative positions could remain the same. In that case the result of the matching will be one group which represents the same global surface ${}^nG_l$. However, if relative positions change then several groups could exist, each representing a new global planar surface. If only one group exists there is no need to merge planar surface segments again and only values of parameters ${}^GR_l^0$, ${}^Gt_l^0$ are re-estimated. If several groups exist, parameters of the new global planar surface for each group are estimated and new global planar surfaces are added to the global map while the original global planar surface ${}^nG_l$ is deleted.

After reformatting is completed, the third and final additional step is performed. Since the states, between which loop closing is detected, are presumed to have similar local maps, matching them will result in numerous pairs that will connect newly segmented planar surface segments from M_k with global planar surfaces containing planar surface segments from M_j, thus creating ground for connecting planar surface segments from the future local maps with the same global planar surfaces. Updating of the global map by matching segments from M_k and M_i is the same process as the global map update between M_k and M_{k-1}. After this step, update of the global map occurs by matching local maps M_k and M_{k-1} just like in the case of no loop closing detection.

FINAL GLOBAL MAP GENERATION. In order to complete the update of the global map the only thing left to do is to transform all the newly built global planar surfaces into the coordinate frame S_0. This is done by transforming every point ${}^{G}p$ in contours of all newly created global planar surfaces using equation

$$ {}^{G}\tilde{p} = {}^{G}R_l^0({}^{G}p) + {}^{G}t_l^0 \tag{4.59} $$

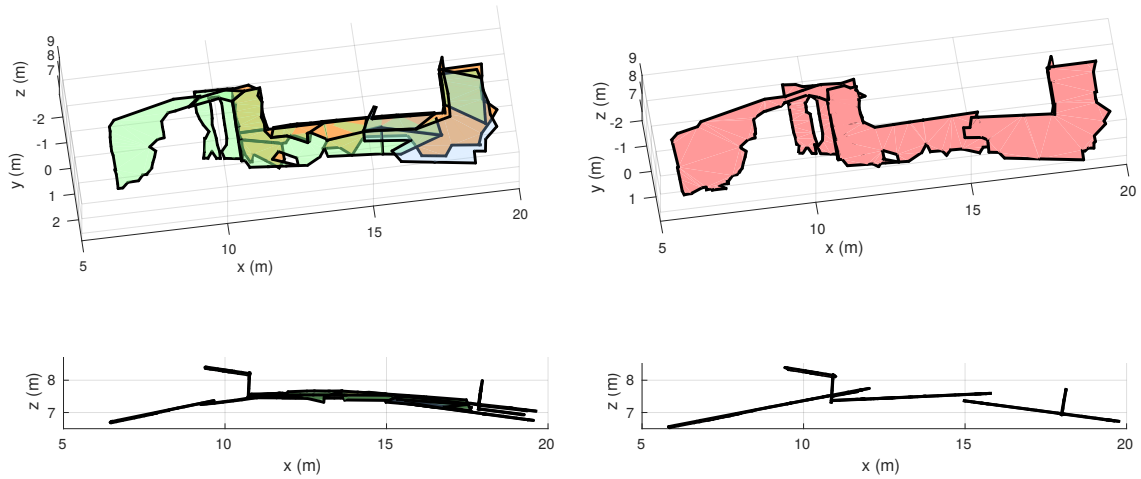

Figure 4.7: Left: Side and top view of curved wall consisting of planar surface segments from three consecutive local maps (blue are segments from M_{k-1}, orange are segments from M_k and green are segments from M_{k+1}); Right: Side and top view of curved wall represented in the global map by global planar surfaces.

After this step, the update of the global map is complete. However, there is a possibility that some of the global planar surfaces can still be merged with one another. The reason for this is explained by the following example. Let's say that a vehicle travels through a corridor and during that path it adds three new local maps (M_k, M_{k+1}, M_{k+2}). According to the algorithm, each wall of the corridor should be one surface. If during the recording of the LIDAR measurements for model M_{k+1} e.g. a group of people moves between corridor wall and the vehicle, the corridor surface will not be present in the model M_{k+1}. Since loop closing algorithm does not match local maps M_k and M_{k+2} there will be no connection established with the corridor planar surface segments from M_k and M_{k+2} and they will be displayed as two separate global planar surfaces in the global map. In order to solve this problem the global map has to be refined. Refinement process is done in the same way as updating the global map between two local maps, except that this time the input to the matching algorithm are global planar surfaces contained within the global map. The algorithm than matches only new global planar surfaces and global planar surfaces that were updated after the last state was added, with all unchanged global planar surfaces, and generates new groups of matched global planar surfaces. The rest of the process is the same as for the group of planar surface segments, described in the previous section. Once all the global planar surfaces from the group have been merged, the global planar surfaces within the group are deleted from the global map and replaced with the merged global planar surface. Figure 4.7 shows the example of a curved wall in the global map after three consecutive states are added to the trajectory and global map update process is completed.

UPDATING PLANAR SURFACE SEGMENTS. Updating local maps is one of the steps during the global map update in case the loop closing is detected between states X_i and X_j. Since in ESDSF back-end map landmarks are independent on each other, newly updated trajectory T_n can be used to update parameters of observed plane segments in local maps M_i and M_j independently.

As explained in the previous section, during the global map update after the loop closing between states X_i and X_j planar segments between M_i and M_j are matched immediately after the trajectory update is completed. For each pair $(F_{a,i}, F_{b,j})$ of coplanar surface segments detected during the matching step, parameters of one planar surface segment are considered as observations to its paired planar surface segment. To estimate updated parameters of the $F_{a,i}$, a priori state estimate $\hat{x}$, and its covariance matrix estimate Q_x, are set equal to the expected normal and offset of that planar surface and its uncertainty

$$\hat{x} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^{\mathrm{T}} \tag{4.60}$$

$$Q_x = \Sigma_{q^z_{a,i}} \tag{4.61}$$

Measurement z and covariance R_z are set by transforming parameters of $F_{b,j}$ into the $S_{F_{a,i}}$ using relative pose between X_i and X_j obtained from the updated SLAM trajectory

$$z = \begin{bmatrix} {}^F\tilde{n}^x_{b,j} & {}^F\tilde{n}^y_{b,j} & {}^F\tilde{n}^z_{b,j} & {}^F\tilde{\rho}_{b,j} \end{bmatrix}^T \tag{4.62}$$

$$R_z = \tilde{\Sigma}_{q^z_{b,j}}. \tag{4.63}$$

Finally, EKF a posteriori update is applied, which results in updated parameters x' of the planar feature $F_{a,i}$. Parameters $R'_{F_{a,i}}$ and $t'_{F_{a,i}}$ are estimated by aligning x with x' using minimal rotation matrix in the same way as in the global plane estimation. Once parameters of $F'_{a,i}$ are estimated, the same process is repeated for $F_{b,j}$, taking transformed parameters of $F_{a,i}$ as measurements and parameters of $F_{b,j}$ as a priori state estimate.

This concludes the description of all major steps included in the global map building and the global map optimization. In the remainder of this chapter algorithms for loop closing detection and calculating pose constraints from those loop closings will be presented.

4.3.4 *Loop closing detection*

Whenever a new state is added to the trajectory, the loop closing algorithm begins to search for possible loop closings between the newly added state X_k and all other states in the trajectory T_n. The loop closing detection is done by the already described algorithm FAB-MAP [107]. In addition to the 3D LIDAR robot was equipped with camera and images were recorded whenever new state was added into the trajectory. This image was sent to FAB-MAP in order to find matches between all previously sent images. However, not all loop closings reported by the FAB-MAP are used for the trajectory update, instead a condition is implemented which measures if the resulting update would have high enough impact on the trajectory accuracy. Although every loop closing would increase accuracy of the trajectory, trajectory update and the additional steps required during the global map update after the loop closing is detected are performance costly operations. This is why they should not be executed if the update impact on the trajectory accuracy is too small. In general,

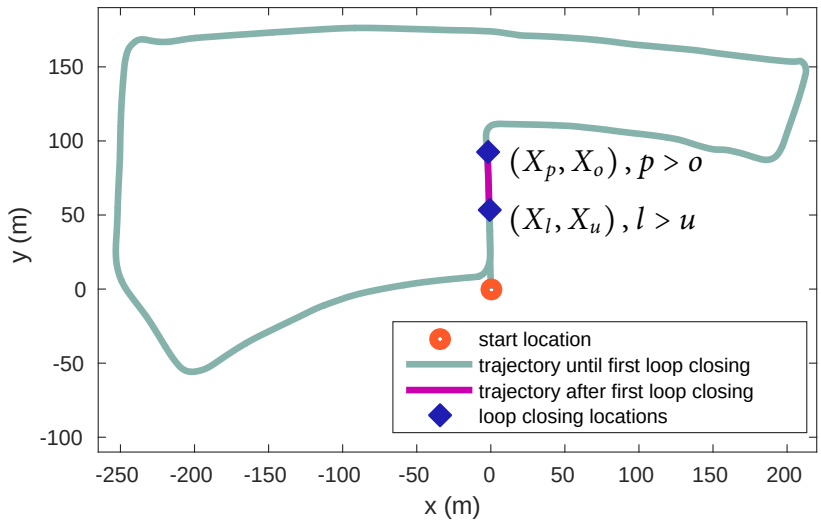

Figure 4.8: Example of a robot trajectory suitable for rejecting loop closings based on information gain. First loop closing occurred between states X_l and X_u, and second loop closing occurred between states X_p and X_o. The first state in the pair designates the location where loop closing occurred. States X_o and X_u were added to the trajectory when the robot traversed the area for the first time, while states X_p and X_l were added when it arrived at the same place for the second time.

impact of the update, started by the loop closing between states X_k and X_a, on the trajectory accuracy is proportional to the sum of cost functions $f_c(i, j)$ between all neighboring states in the trajectory moving from the state X_k to the state X_a. The cost function $f_c(i, j)$ takes into account both angle and distance differences between two states and is calculated as:

$$f_c(i, j) = e_{trans} + \alpha e_{rot}\,, \tag{4.64}$$

where α is the scaling factor, and

$$\Delta T = X_i^{-1} X_j = \begin{bmatrix} & & & \Delta x \\ & R & & \Delta y \\ & & & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \begin{aligned} e_{trans} &= \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \\ e_{rot} &= \arccos\left(\frac{\text{trace}(\Delta T)}{2} - 1\right) \end{aligned}\,. \tag{4.65}$$

However, the problem with always using sum of all cost functions between neighboring states to calculate accuracy impact of the loop closing arises if there were previous loop closing detections. For example let's consider situation illustrated in Fig. 4.8, where we want to measure accuracy impact of loop closing between states X_p and X_o. Although the sum of cost functions between all states from X_o to X_p is high, since the update occurred between states X_l and X_u, the overall impact on trajectory accuracy from closing the loop between states X_p and X_o is small. This is because a lot of information that would be gained from loop closing between states X_o and X_p was already gained by closing the loop between states X_l and X_u. In order to solve this problem, approach similar to [26] is used as a measure of accuracy impact of a loop closing. The idea is to build a graph ${}^{T}g$ from the pose graph incidence matrix ${}^{T}I$. The pose graph incidence matrix ${}^{T}I$ is $n \times n$ matrix, where n is the

number of states in the trajectory, whose elements are given by

$$ {}^{T}I_{ij} = \begin{cases} 1, & \text{if state } X_i \text{ is connected with state } X_j \\ 0, & \text{otherwise} \end{cases} \tag{4.66} $$

The states are connected if they are neighboring states (i.e. the state X_i is connected with states X_{i+1} and X_{i-1}) or if the pose constraint was formed between them (i.e. loop closing was detected and trajectory update executed). Nodes in the graph ${}^{T}g$ are represented by the states and connections between nodes N_i and N_j exist if the element (i,j) in the matrix ${}^{T}I$ is 1. Weight of the connection $w_{i,j}$ between the nodes N_i and N_j is

$$ w_{i,j} = \begin{cases} f_c(i,j), & \text{if } |i-j| = 1 \\ 0, & \text{otherwise} \end{cases} . \tag{4.67} $$

Calculating connection weight this way ensures that connections made by previous loop closing have 0 weight and will not increase measured topological distance between two states. Once the graph is generated, information gain from closing the loop between states X_i and X_j can be calculated by finding the shortest path from a node N_i to a node N_j using the A^* algorithm, and then calculating the total weight of the path. When calculated in such a way, the total weight of the path is also refereed to as the topological distance ${}^{T}d_{i,j}$ between states X_i and X_j. The higher ${}^{T}d_{i,j}$ the more information is gained from the loop closing. Let's again reefer to Fig. 5.2 for an example. The information gained from the first loop closing (X_l, X_u) is:

$$ {}^{T}d_{u,l} = \sum_{e=u+1}^{l} f_c(e-1,e) , $$

while the information gained from the second loop (X_p, X_o) is:

$$ {}^{T}d_{o,p} = \sum_{e=u+1}^{o} f_c(e-1,e) + f_c(u,l) + \sum_{e=l}^{p-1} f_c(e,e+1) . $$

If the update did not occur between (X_l, X_u), ${}^{T}d'_{o,p}$ would have the following value

$$ {}^{T}d'_{o,p} = \sum_{e=o+1}^{l} f_c(e-1,e) + \sum_{e=l}^{p-1} f_c(e,e+1) . $$

It can be seen that ${}^{T}d_{o,p}$ is much smaller than ${}^{T}d'_{o,p}$ since

$$ \sum_{e=u+1}^{o} f_c(e-1,e) + f_c(u,l) \ll \sum_{e=o+1}^{l} f_c(e-1,e) . $$

Now that the trajectory accuracy impact of the potential trajectory update can be measured, decision can be made which updates to perform and which to ignore. All loop closings detected by FAB-MAP, whose topological distance is larger than the predefined threshold Td_{max}, are reported to the Local Map Registration algorithm (LMR).

4.3.5 *Local maps registration*

In order to perform trajectory and map update, the relative pose between two states X_i and X_j must be estimated and reported to the SLAM back-end as a constraint. This is accomplished by the Local Map Registration (LMR) algorithm based on the approach described in [135]. Given two local maps representing two sets of 3D planar surface segments, LMR searches for the relative pose between these two local maps which maximizes overlapping between the surface segments of the first local map and the surface segments of the second local map, transformed by this pose into the reference frame of the first local map. The approach presented in [135] is primarily designed for use with RGB-D cameras, while the LMR applied in this SLAM solution represents an adaptation of this approach to registration of two local maps acquired by the 3D LIDAR. Only a brief description of the applied registration approach with emphasis on the differences between the method presented in [135] and the LMR applied here is presented. For more details on the method the reader is referred to [135].

The considered LMR approach generates multiple hypotheses about the pose of the reference frame S_j corresponding to the state X_j with respect to the reference frame S_i corresponding to the state X_i. The hypotheses are generated by selecting small sets of pairs $(F_{a,i}, F_{b,j})$ such that the surface segments selected from M_i have similar geometric arrangement to the corresponding segments selected from M_j. Then, EKF-based registration of the selected surface segments from M_i to the corresponding surface segments from M_j is performed, resulting in a hypothetical relative pose between X_i and X_j, which can be represented by $w_{i,j} = (R_{i,j}, t_{i,j})$, where $R_{i,j}$ is the rotation matrix defining the orientation and $t_{i,j}$ is the translation vector defining the position of X_j relative to X_i. In general, the number of possible combinations of feature pairs, which can be used for generating hypotheses, is very large. In order to achieve computational efficiency, a strategy for selection of feature pairs proposed in [134] and described in detail in [135] is applied. It is based on the feature ranking according to a measure of their usefulness in the pose estimation process. The result of this hypothesis generation process is a set of hypotheses about the pose $w_{i,j}$. Besides the correct hypothesis, this set usually contains many false hypotheses. The number of false hypotheses is significantly reduced by using the coarse information about the pose $w_{i,j}$ provided by SLAM trajectory to constrain the selection of planar surface segment pairs $(F_{a,i},F_{b,j})$ to those which satisfy coplanarity criterion and overlapping criterion with respect to this initial pose estimate ${}^{I}w_{i,j}$, as proposed in [135].

The initial pose estimate ${}^{I}w_{i,j}$, required by the LMR is defined by the rotation matrix ${}^{I}R_{i,j}$, translation vector ${}^{I}t_{i,j}$ and covariance of the transformation ${}^{I}P_{i,j}$. In order to calculate ${}^{I}w_{i,j}$, unscented transform is used. Inputs to the unscented transform are vector v_P

$$v_P = \left[t_i \; q_i \; t_j \; q_j\right]^{\mathrm{T}}, \tag{4.68}$$

where t_i, q_i, t_j and q_j are positions and orientation quaternions assigned to trajectory states X_i and X_j, and its covariance matrix Σ_P. Both v_P and Σ_P are obtained from the SLAM trajectory. Nonlinear function used in the unscented transform is equal to the measurement model (4.28) modified to operate on Euler angles instead of quaternions for representing rotation, since LMR requires rotation matrix as input.

The main difference between the method applied in [129] and the original LMR approach presented in [135] is in the hypothesis evaluation stage. In this stage, each generated hypothesis $w_{i,j}$ is assigned a conditional probability $P(w_{i,j}|M_j, M_i)$ representing the estimated probability that $w_{i,j}$ is the pose of X_j relative to X_i if M_j is a local map consisting of planar surface segments segmented in the state X_j and M_i is a local map consisting of planar surface segments segmented in the state X_i. The hypothesis with the highest estimated probability is selected as the final solution.

Assuming that the prior probabilities of all hypotheses are equal, maximizing $P(w_{i,j}|M_j, M_i)$ is equivalent to maximization of likelihood $p(M_j|w_{i,j}, M_i)$, i.e. the conditional probability of segmenting the local map M_j with particular parameters in the state X_j if the local map M_i is detected in the state X_i and the pose of X_j relative to X_i is $w_{i,j}$. The probability $p(M_j|w_{i,j}, M_i)$ is computed as follows:

$$p(M_j|w_{i,j}, M_i) = \prod_{F_{b,j} \in M_j} \max\{ \max_{F_{a,i} \in M_i} (p(F_{b,j}|w_{i,j}, F_{b,j} \equiv F_{a,i})), p(F_{b,j}) \}, \tag{4.69}$$

where $p(F_{b,j}|w_{i,j}, F_{b,j} \equiv F_{a,i})$ is the probability of detecting a planar surface segment $F_{b,j}$ with particular parameters in the state X_j if this segment represents the same surface as the segment $F_{a,i}$ detected in the state X_i, while $p(F_{b,j})$ represents the prior probability of detecting a planar surface segment with parameters equal to $F_{b,j}$.

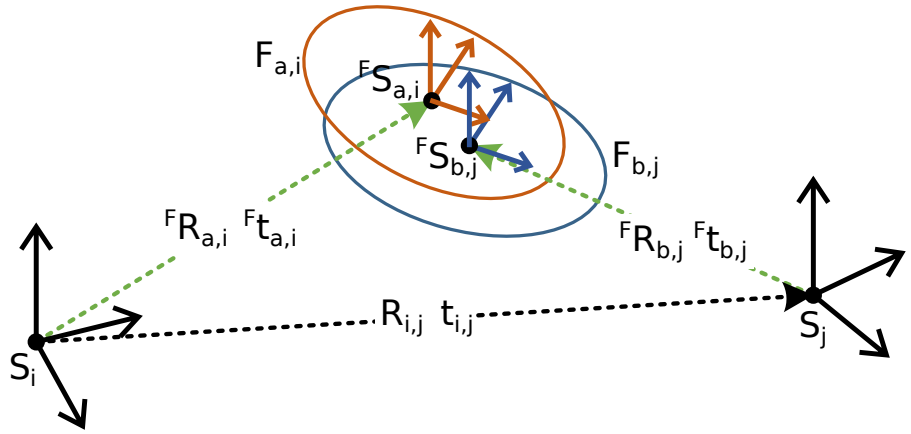

Figure 4.9: Relative pose between planar surface segments $F_{a,i}$ and $F_{b,j}$.

Probabilities $p(F_{b,j}|w_{i,j}, F_{b,j} \equiv F_{a,i})$ and $p(F_{b,j})$ are computed using the approach proposed in [135]. While in [135] only plane normal is considered in probability computation, in LMR used in the present SLAM solution, the plane offset is also considered in the same manner. The probability $p(F_{b,j}|w_{i,j}, F_{b,j} \equiv F_{a,i})$ for a particular pair $(F_{a,i}, F_{b,j})$ is computed only if surface segment $F_{b,j}$, transformed to the reference frame S_i of the local map M_i using the rotation matrix $R_{i,j}$ and translation vector $t_{i,j}$, overlaps sufficiently with $F_{a,i}$. Reference frames S_i and S_j of the local maps M_i and M_j are shown in Fig. 4.9, together with two planar surface segments $F_{a,i}$ and $F_{b,j}$ and their reference frames $S_{F_{a,j}}$ and $S_{F_{b,j}}$.

As stated before, the approach presented in [135] is originally designed for RGB-D cameras. Due to relatively narrow FOV of RGB-D cameras, in many cases, relatively small number of dominant surfaces is captured within the camera FOV. Therefore, in addition to supporting plane parameters of the detected planar surface segments, the segment shape

must also be considered in the place recognition process in order to allow reliable distinction between geometrically similar places. In order to include the information about the surface shape in the hypothesis evaluation stage, in [135] planar surface segments are represented by sets of square patches approximating the surface shape. However, 3D LIDAR has a 360° FOV, allowing it to capture many dominant surfaces in the local environment. The information provided by the plane parameters of the detected surfaces has shown to be sufficient for a reliable registration of two views. Furthermore, in contrast to the research presented in [135] where the discussed approach is used to recognize the particular place among a relatively large number of possible candidate local maps, in [129], the considered approach is used for registration of only two close local maps, making the discrimination of surface segments according to their precise surface shape unnecessary. Therefore, time consuming surface sampling is not applied and overlapping of the scene surface segment with a corresponding planar surface segment is measured by an alternative approach. Instead of counting overlapping surface patches, which is the approach applied in [135], overlapping of two corresponding planar surface segments is measured by approximating these surface segments with ellipsoids whose radii are defined by the eigenvalues of their covariance matrices Σ_p.

Overlapping between two planar surface segments $F_{a,i}$ and $F_{b,j}$ is measured by Mahalanobis distance between distributions of points of these two surface segments represented by their centroids ${}^F t_{a,i}$ and ${}^F t_{b,j}$ and covariance matrices $\Sigma_{p_{a,i}}$ and $\Sigma_{p_{b,j}}$, which is computed by

$$d_p(F_{a,i}, F_{b,j}; w_{i,j}) = ({}^F t_{a,i} - {}^F t_{b,j})^{\mathrm{T}} (\Sigma_{p_{a,i}} + \Sigma_{p_{b,j}} + \Sigma_w)^{-1} ({}^F t_{a,i} - {}^F t_{b,j}) \,, \tag{4.70}$$

where Σ_w represents the uncertainty of the estimated pose $w_{i,j}$. Covariance matrix Σ_w is added to the covariance matrix $\Sigma_{p_{i,j}}$ of the surface segment $F_{b,j}$ because this segment is transformed in the reference frame S_i using pose $w_{i,j}$. Covariance matrix Σ_w is defined using a simplifying assumption that the uncertainty in translation is equal in all directions and that the uncertainty in rotation around all three axes is equal. It is computed as follows

$$\Sigma_w = \sigma_t^2 I_{3\times 3} + \sigma_R^2 (r^{\mathrm{T}} r I_{3\times 3} - r r^{\mathrm{T}}) \,, \tag{4.71}$$

where $I_{3\times 3}$ is a unit matrix, σ_t and σ_R represent uncertainty of the estimated translation and rotation, respectively, and

$$r = {}^F t_{b,j} - t_{i,j} \,. \tag{4.72}$$

Parameters σ_t and σ_R are determined experimentally. Pair of planar surface segments $F_{a,i}$ and $F_{b,j}$ is considered as successful match if the following condition is satisfied

$$d_p(F_{a,i}, F_{b,j}; w_{i,j}) \leq \varepsilon_p \,, \tag{4.73}$$

where the threshold ε_p is computed according to a desired probability assuming χ^2 distribution of d_p distance.

At this moment the derived SLAM solution possess all the required components to work autonomously in complex 3D environments and produce compact and accurate map and accurate robot location. In the next section active SLAM component is presented which allows it to maintain the map and the trajectory accuracy while coupled with the exploration algorithm.

Algorithm 2: Active SLAM algorithm

loop:
if More than n_{max} states are added without the loop closing occurring **then**
  Calculate Euclidean distance ${}^{E}d_{k,j}$ between the current robot pose X_k and all other states $X_j, 1 < j < k$
  if ${}^{E}d_{k,j} \leq^{E} d_{max}$ **then**
    Calculate topological distance ${}^{T}d_{k,j}$
    if ${}^{T}d_{k,j} \geq^{T} d_{min}$ **then**
      Stop the robot from exploring
      Send the robot to state X_j
      if Loop closed **then**
        Send the robot to the exploration goal that was set before the exploration was stopped
      end if
    end if
  end if
end if

4.4 ACTIVE SLAM COMPONENT

The main question when using the active SLAM is how to define a criterion which determines when the robot motion should be influenced by the active SLAM. In [130] the main criterion is the build up of uncertainty in the robot current pose. This is measured by counting the number of consecutively added states into the trajectory without the loop closing occurring. Once this number exceeds the predefined threshold n_{max}, active SLAM algorithm tries to find suitable states for loop closing around the robot current pose. In contrast to the normal loop closing detection, which requires two states to have measurements taken from roughly the same pose, in this case it is enough that Euclidean distance between the state X_j and the current robot pose X_k is lower than the predefined threshold ${}^{E}d_{max}$. Once, such state is identified, topological distance between X_j and X_k is calculated the same way as if it is a loop closing detected by the FAB-MAP. If the topological distance is larger than ${}^{T}d_{min}$ robot is stopped and sent to close the loop in X_j. If, upon arrival at X_j the loop closing is not immediately detected, the robot follows previously traversed path, towards state X_{j+1}, until the loop detection occurs. Once the loop is closed, robot returns to its previous goal set by the exploration algorithm. These steps are summarized in the Algorithm 2.

The aim of any exploration algorithm is to find a minimal number of poses from where to take scans in order to build a detailed map of the environment. The algorithm needs to guarantee a complete exploration of the environment within a finite number of measurements. The exploration algorithm used in combination with the presented active SLAM solution is based on the exploration strategy described in [139] and [140], which is an extension of Ekman's exploration algorithm [141] by removing the rigid constraints on the range sensor and a robot localization. In [141] was shown that the exploration of polygonal environments guarantees a complete coverage considering no positional uncertainty and

an ideal range sensor. Under the assumption that in our SLAM the uncertainty is lower each time when active SLAM closes a loop, our exploration strategy completely explores the environment. Since the used path planning algorithm works only in 2D and since the active SLAM component was tested in indoor environment the 2D version of the exploration algorithm was chosen, due to its simpler integration and robustness.

The exploration starts with no a-priori information on the environment and after the first laser scan in 2D an initial environment model is generated. The model consists out of lines vectorized from the initial scan. Based on the initial map and information available from the first scan, the next best robot pose is calculated. The next candidate scan poses are defined 1m in front of the lines which separate explored and unexplored area, so-called jump edges. The jump edges are generated by connecting the two adjacent points in one scan if the distance between points is above some threshold value, i.e. connecting discontinuities in a range data. For details see [139].

Figure 4.10 shows generated candidate poses in 2D X_i and X_j from the current robot pose X_k. The jump edges are marked with red color. Among all the jump edges the next

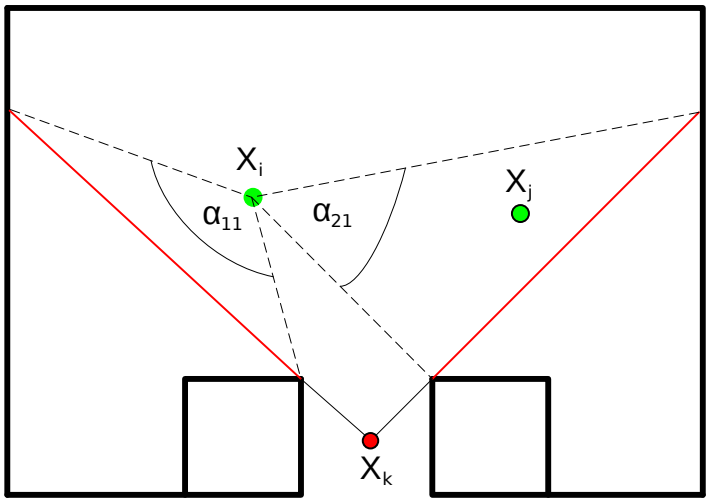

Figure 4.10: Jump edges from the current robot pose X_k

best scan position is chosen according to a criterion that maximizes the area explored in the next scan and minimizes a distance from the current robot position to the jump edge. Estimation of the explored area in the next step is done by the angle between potential candidate position and two jump edge ends as shown on Fig. 4.10. Also the angles according to the other jump edges are taken into account which leads to the following criterion for candidate position X_j:

$$I_j = k_1 \frac{1}{d_{k,j}} + k_2 \sum_{i=1}^{N} \alpha_{ij}, \tag{4.74}$$

where $d_{k,j}$ represents a path length from the current robot pose to the candidate pose, while α_{ij} refers to the angle between candidate poses X_i and X_j, and k_1 and k_2 are tuning parameters used to treat angles and path distance equally. In each step the exploration node communicates with a path planning module, receiving the distance to the all candidates position, i.e. path length. The best candidate scan position is sent to the planner module which drives the robot to it to take the scan. The whole procedure is then repeated. For

planning the path from one exploration or active SLAM goal to another we used solution presented in [142]. It uses receding horizon control approach for the trajectory execution based on the path planned by the D* algorithm [143].

4.5 FUNCTIONAL FLOW DIAGRAM OF THE PROPOSED SLAM SOLUTION

To summarize the presented passive SLAM solution, functional flow diagram of the entire system including front-end and back-end components is presented in Fig. 4.11. It depicts all steps, from predicting current robot pose using odometry, to refinement of the global map. Functional flow diagram consists of the same major components as the overall system concept diagram shown in Fig. 4.1, but presents them in more detailed view by segmenting them to several function blocks. All blocks represented with green color belong to the *SLAM backend* module and all orange blocks belong to the *Global map building* module. Each block marked with black color corresponds to related module of the overall system concept diagram. It is important to notice that since map building and localization are two separate processes, the orange blocks are executed in parallel with the green blocks. This means that once trajectory update is complete, *SLAM backend* does not wait for the global map building to finish but continues to augment and update the robot trajectory in the meantime. This ensures that timely global map operations, like update after the loop closing, do not affect pose accuracy.

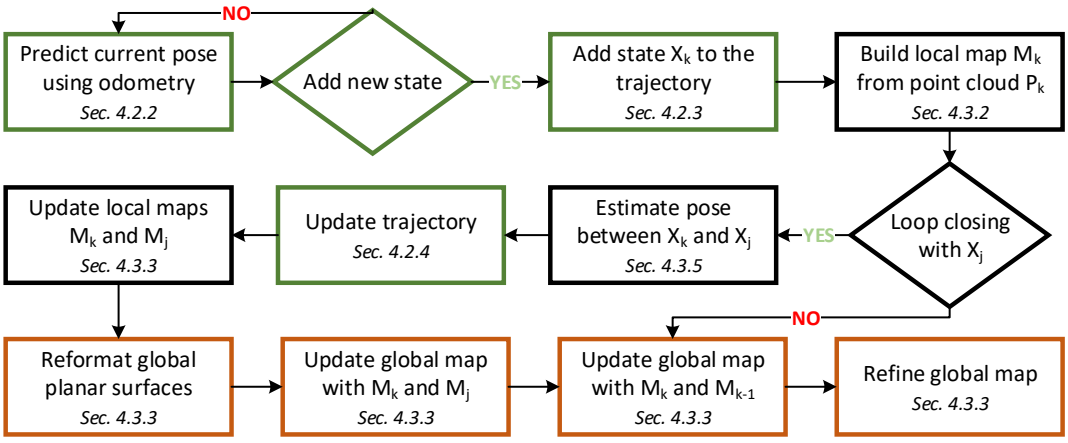

Figure 4.11: Functional flow diagram of the entire SLAM system.

This concludes the derivation of the SLAM system. In the sequel the experimental results, proving the effectiveness of the presented SLAM solution and its components, are presented.

4.6 EXPERIMENTAL RESULTS

In order to test the developed SLAM solution and all its components, experiments were conducted in three stages. First point cloud segmentation and registration algorithm was tested, then experiments were conducted using the complete SLAM solution and finally active SLAM component was tested. All algorithms were implemented in Robot Operating

System (ROS) [144] using C++ programming language and both experiments were executed on Lenovo Thinkpad P50 with 8 GB RAM and Intel Core i7@2.6 Ghz processor running 64-bit Ubuntu 14.04 LTS operating system.

4.6.1 *Test results for point cloud segmentation and registration*

Developed point cloud segmentation and local map registration algorithm, which for the purpose of this testing is dubbed (PCS-LMR), was compared with the state-of-the-art local 3D registration algorithms: two variants of Normal Distributions Transform (NDT) and two variants of ICP, using standardized benchmarking protocol [101] on the structured environment datasets. Also, PCS-LMR was compared to one global alternative, Minimally Uncertain Maximum Consensus (MUMC) method proposed in [72], using the "Collapsed car park" dataset[1]. The "Collapsed car park" dataset consists of 35 point clouds collected with a mobile platform and a 2D LIDAR mounted on a pan-tilt unit. The comparison was made with the currently available implementation of MUMC[2] which works more accurately than the original one described in [72], but does not have graph relaxation method. Because of this both algorithms were tested only for planar segmentation and registration on a consecutive point clouds and trajectory was constructed based on these measurements; no loop closing constraints were added nor pose graph optimizations performed. One of the main parameters which impacts the MUMC performance is the number of planar surface segments used in matching, i.e. "filter percentage threshold". In MUMC, the planar surface segments are sorted in the decreasing order of their statistical certainty, and only the top filter-percentage-threshold is used for matching. This means that the lower threshold values increase computation speed but decrease the accuracy.

This parameter was chosen to be the lowest possible which still produces relatively accurate registrations based on qualitative analysis of the trajectory and the map since neither a ground truth trajectory nor a 3D model is available. Figure 4.12 shows MUMC trajectories for three different threshold values and the trajectory obtained by PCS-LMR. As can be seen, the trajectories of PCS-LMR and the MUMC trajectory for the threshold of 60% ($MUMC_{60}$) are the most similar, MUMC trajectory for the threshold set at 40% ($MUMC_{40}$) is still close to the first two, while MUMC trajectory for the threshold of 20% ($MUMC_{20}$) is severely degraded. Since computation time for $MUMC_{40}$ is significantly lower than for $MUMC_{60}$% (Table 4.3), and there is no exact way to assess the accuracy of the trajectories, PCS-LMR was compared to the $MUMC_{40}$. The resolution of each of the three image planes used to project a point cloud in PCS-LMR was set to 512×663.

Table 4.3: Mean, maximum and minimum registration times of consecutive point clouds for PCS-LMR and MUMC with different filter thresholds. All times are in seconds.

	$MUMC_{60}$	$MUMC_{40}$	$MUMC_{20}$	Our
Mean	23.30	8.02	2.83	0.29
Max	123.84	35.00	7.02	0.42
Min	3.54	1.73	0.95	0.11

[1] http://robotics.jacobs-university.de/node/292

[2] provided by the courtesy of Prof. Pathak, Prof. Pfingsthorn and Prof. Vaskevicius

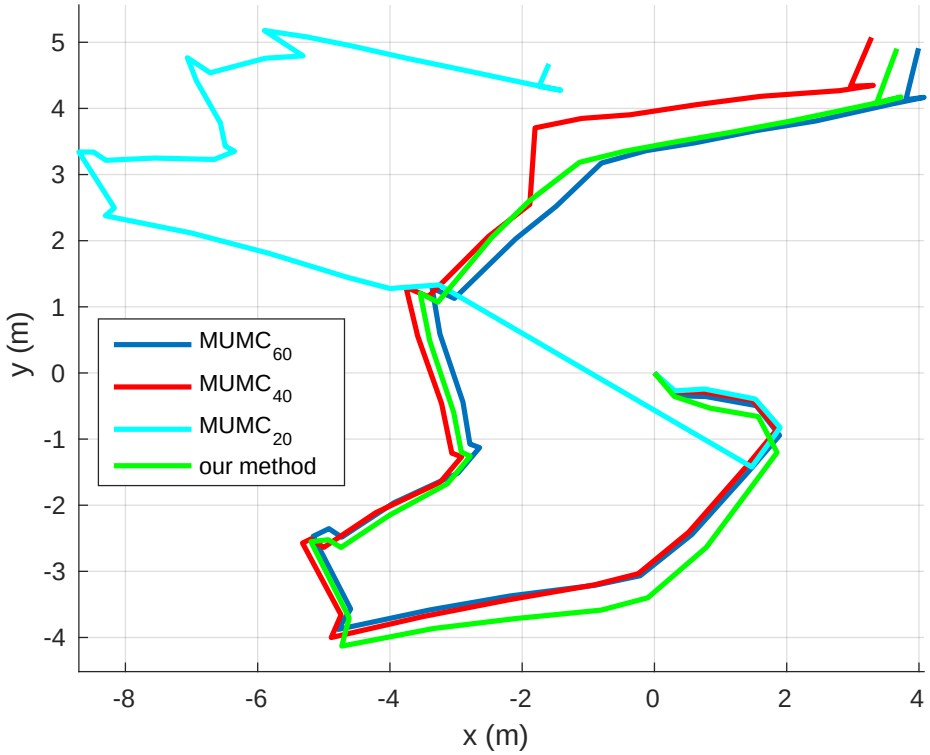

Figure 4.12: Estimated trajectories using PCS-LMR and using MUMC with different filter percentage threshold values.

The trajectory estimation accuracy was tested indirectly by observing the quality of mapping of dominant planar structures in the scene, e.g the walls, the floor, the ceiling etc. Once the trajectory has been estimated, it was used to transform the extracted planar surface segments to the coordinate frame of the first scan. Then, the comparison was made how well the planar surface segments extracted from the point clouds align with each other in the case the trajectory acquired from MUMC_{40} and in the case PCS-LMR trajectory was used. The planar map of the area constructed from all 35 scans using PCS-LMR is shown on Fig. 4.13. Some elements are not shown because smaller planes have been filtered out too improve visibility. All planar surface segments with the same color belong to the same point cloud. Three main groups of planar segments can be distinguished: first is the wall, second is the floor and third is the ceiling group. Figure 4.13 also shows zoomed parts of the map. From the zoomed parts it can be seen how well the planar surface segments from different scans align with each other for each of the three groups. Figure 4.14 shows the same map, but built with trajectory estimated using MUMC_{40}. Both PCS-LMR and the MUMC work well on this dataset although PCS-LMR does align planes somewhat better which can be best seen on the ceiling group.

Table 4.4 shows computation times for PCS-LMR and for MUMC_{40}. It can be seen that PCS-LMR is about 4 times faster in the segmentation and 26 times faster in the registration. Such high computation times for MUMC_{40} method are due to the fact that MUMC is a global method, i.e. does exhaustive search for the most consistent set of planar surface correspondences without an initial guess, while initial guess of PCS-LMR was set to zero pose with high uncertainty.

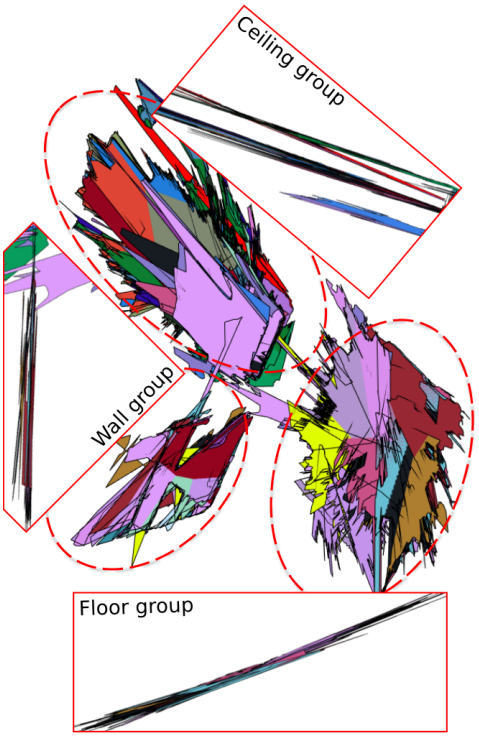

Figure 4.13: Planar map built using trajectory estimated by PCS-LMR.

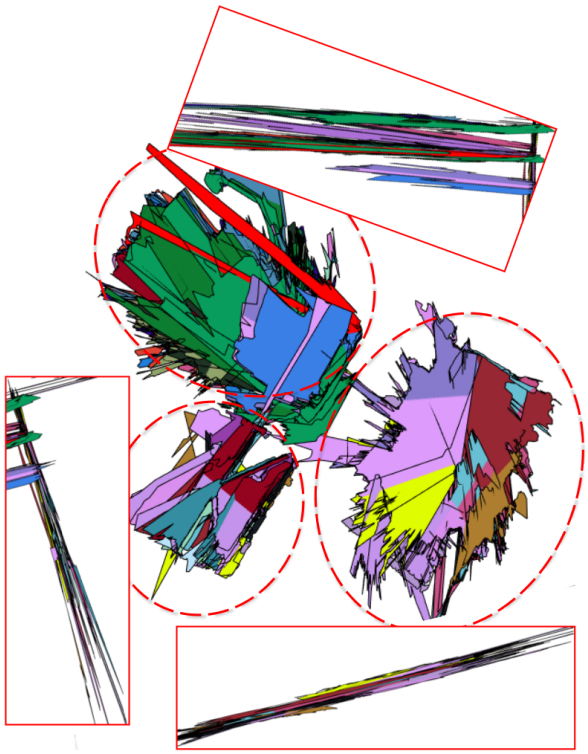

Figure 4.14: Planar map built using trajectory estimated by $MUMC_{40}$.

Table 4.4: Mean, maximum and minimum computation times for segmentation (seg). and registration (reg.) of point clouds. All times are in seconds.

	PCS-LMR			$MUMC_{40}$		
	Seg.	Reg.	Total	Seg.	Reg.	Total
Mean	0.17	0.29	0.46	0.70	8.02	8.72
Max	0.22	0.42	0.62	0.82	36.00	36.78
Min	0.13	0.11	0.22	0.59	1.73	2.35

The dataset used to compare PCS-LMR method with local 3D registration algorithms is the "Challenging data sets for point cloud registration algorithms"[3] described in [145]. This dataset consists of point clouds from 6 distinctive environments and covers both indoor and outdoor situations as well as structured and unstructured environments. The exact pose of every point cloud in relation to the first point cloud is provided by a highly accurate differential GPS solution. The dataset was used in [146] to test several state of the art approaches for 3D scan matching. In total 5 algorithms were tested: Point based ICP variant described in [100], Plane based ICP variant described in [99], P2D-NDT method described in [147], D2D-NDT method described in [106] and MUMC method from [72]. The protocol for testing is available on the dataset website. From each environment, 35 different pairs of scans are selected and for each pair 196 initial transforms are given. The initial transforms are derived from the ground truth transforms by adding different magnitude pose offsets and, depending on its difference from the ground truth, a pair is marked as easy, medium

[3] http://projects.asl.ethz.ch/datasets/doku.php?id=laserregistration:evaluations:home

or hard for matching. Details of the entire test protocol including the initial pose generation can be found in [101]. The results of the testing are available on the dataset web page for each algorithm except for MUMC, since MUMC does not make use of initial transform. The results for MUMC method are available in [146] and show how well the method performs depending on the overlap between two point clouds for each of the 35 pairs.

PCS-LMR was tested only on indoor datasets since it is designed to work in structured environments which contain enough planar structure (i.e. indoor and outdoor urban environments). Outdoor environments of the considered dataset are deficient in dominant planar surfaces, required for PCS-LMR, and hence on these environments PCS-LMR is not considered competitive with other methods. Two indoor environments that PCS-LMR wast tested with are the Apartment and the Stairs datasets. Apartment is the largest of the 6 datasets, consisting of 45 scans averaging 365000 points per scan. It includes dynamic conditions resulting from moving furniture between scans and consists of multiple reflecting surfaces. The Stairs dataset consists of 31 scans with average of 191000 points per scan. It is intended to test registration algorithms when there are rapid variations in scanned volumes and when the hypothesis of the planar scanner motion is incorrect.

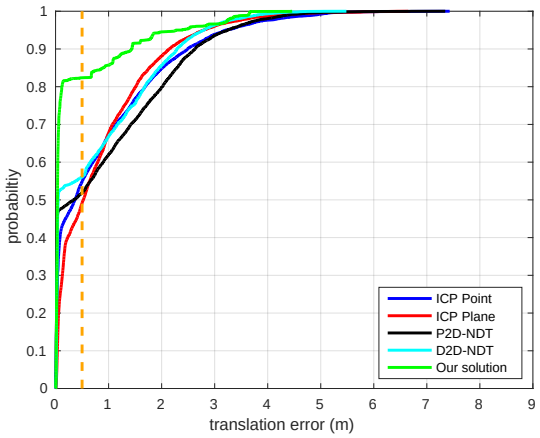

Figure 4.15: Cumulative translation error probability for the Apartment dataset (dashed line represents error of 0.5 m).

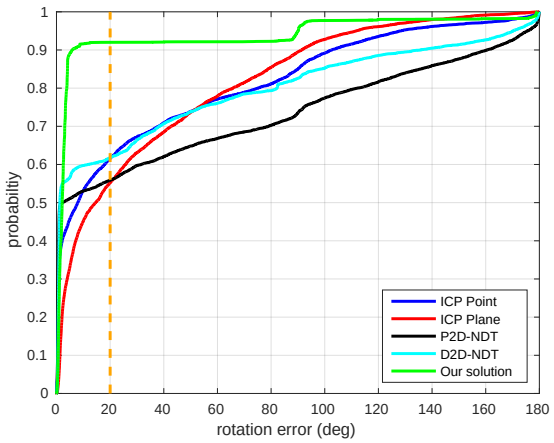

Figure 4.16: Cumulative rotation error probability for the Apartment dataset (dashed line represents error of 20°).

Figures 4.15 and 4.16 show the cumulative probability of the translation and orientation error, respectively, for the Apartment dataset after all 35*196 combinations were matched. It is assumed that the pose estimates which differ from the true values for more than 0.5 m with respect to translation or more than 20 degrees with respect to rotation are not useful for robot navigation and focus our analysis to data within these error bounds. It can be seen that PCS-LMR outperforms all other methods in accuracy both for translation and orientation. More than 82% of translation errors are lower than 0.5 m, and more than 90% of rotation errors are lower than 20°. However in the Apartment dataset PCS-LMR failed to provide relative pose estimate in 24 out of 6720 matchings (0.36%) which was automatically detected by the method itself.

Figures 4.17 and 4.18 show the cumulative probability of the translation and rotation error, respectively, for the Stairs dataset after all 6720 combinations were matched. It can be

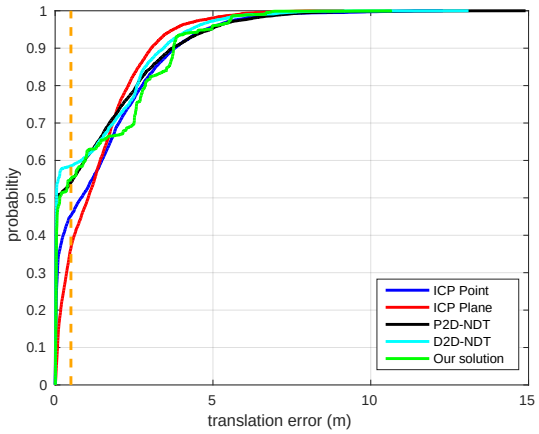

Figure 4.17: Cumulative translation error probability for the Stairs dataset (dashed line represents error of 0.5 m).

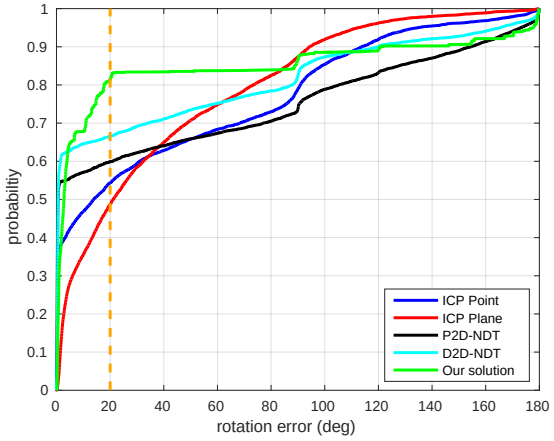

Figure 4.18: Cumulative rotation error probability for the Stairs dataset (dashed line represents error of 20°).

seen that PCS-LMR algorithm has translation error greater than 0.5 m for approximately 5% measurements more than D2D-NDT, but has more accurate rotation estimation. Rotation estimation error of PCS-LMR is less than 20° in approximately 80% cases, while D2D-NDT achieves this result in approximately 65% cases. For the Stairs dataset there were only 3 matchings without estimate (0.045%).

Computation times for the Apartment and Stairs datasets are shown in Tables 4.5 and 4.6. It can be seen that only D2D-NDT is slightly faster than PCS-LMR. The computation time for PCS-LMR was computed by combining the segmentation times for two point clouds and the time needed to perform registration. However, in reality almost always only one newly arrived point cloud has to be segmented while all past point clouds have already been segmented before. This is why a practical application computation time of our algorithm would be even smaller. Furthermore, it has to be mentioned that D2D-NDT and P2D-NDT methods were tested using Intel Core i7@3.5 Ghz while PCS-LMR was tested on Intel Core i7@2.6 Ghz. One more advantage of PCS-LMR is that the map is represented with planar surface segments resulting in much smaller memory consumption in comparison to map representations which consist of points.

Table 4.5: Mean, maximum and minimum computation times of all 6720 relative poses for the Apartment dataset.

	ICP Point	ICP Plane	P2D	D2D	PCS-LMR
Mean	4.37	2.35	0.62	0.22	0.36
Max	25.62	15.16	1.35	0.44	0.64
Min	0.71	0.26	0.09	0.13	0.23

Table 4.6: Mean, maximum and minimum computation times of all 6720 relative poses for the Stairs dataset.

	ICP Point	ICP Plane	P2D	D2D	PCS-LMR
Mean	2.45	1.50	0.86	0.22	0.35
Max	17.60	11.08	3.19	0.78	0.42
Min	0.21	0.15	0.06	0.07	0.29

4.6.2 *Test results for the SLAM system*

Developed SLAM system was tested on two datasets. One dataset was acquired indoor while driving a mobile robot through our university building. The other, outdoor dataset is available online[4] and was acquired by Ford Motor Company while driving their specially equipped Ford F-250 pickup truck.

INDOOR EXPERIMENT. The indoor experiment was conducted using equipment shown in Fig. 4.19. A mobile platform Husky A200 was driven with an average speed of 1m/s through our university building. It was equipped with Velodyne HDL-32E LIDAR and Xsens MTi-G-700 IMU sensor. Velodyne HDL-32E has a vertical field of view of 40° with angular resolution of 1.33°. Its measurement rate was set to 10Hz. Resolution of all three image planes used for projecting point cloud was set to 320 × 240.

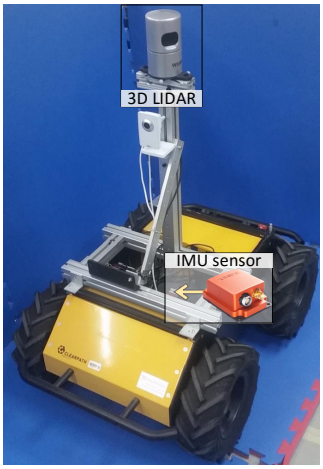

Figure 4.19: Husky with sensors

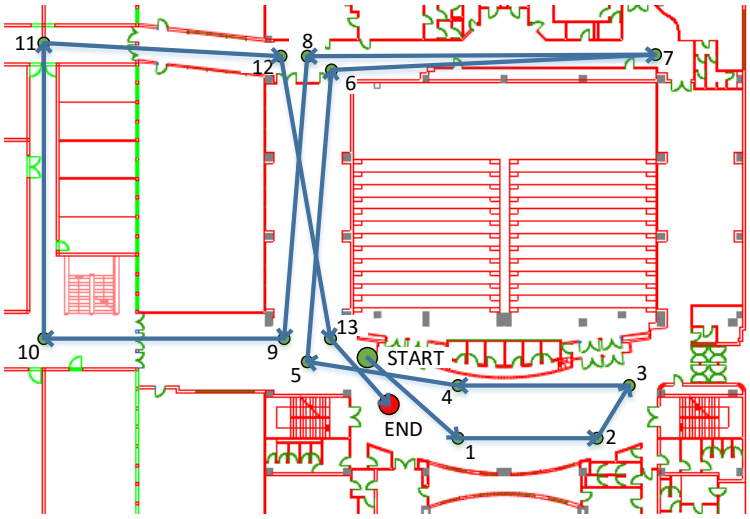

Figure 4.20: Ground plan of the environment and simplified robot's trajectory.

The ground plan of the testing environment and the simplified robot trajectory are depicted in Fig. 4.20. This environment provides challenging conditions for SLAM including many reflective surfaces (i.e. windows and marble floors) and moving people as shown in Fig. 4.21.

During the experiment, 320 states were added to the SLAM trajectory. The graph presented in Fig. 4.22 shows the computation time for the point cloud segmentation process for

[4] http://robots.engin.umich.edu/SoftwareData/Ford

Figure 4.21: Conditions in indoor environment.

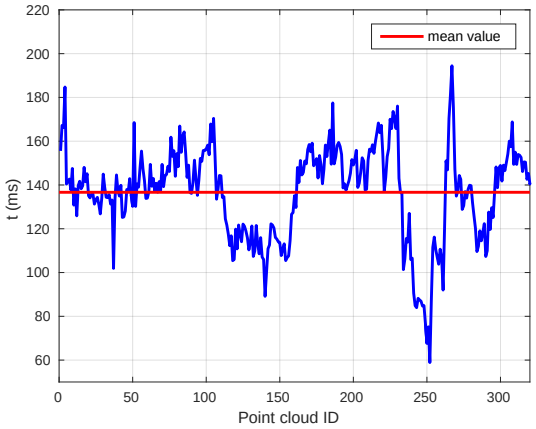

Figure 4.22: Point cloud segmentation time.

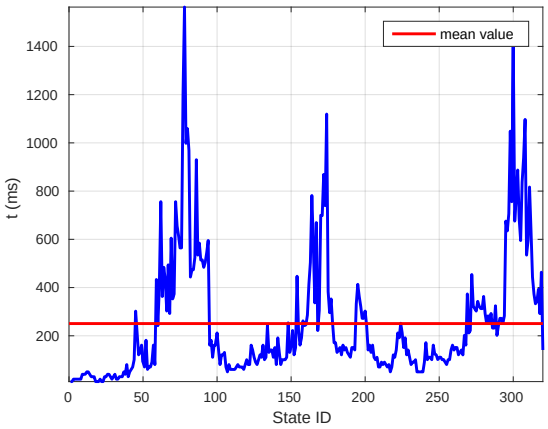

Figure 4.23: Global map update time.

every state added to the trajectory. It can be seen that it is always below 200 ms. Although ground truth trajectory could not be provided, the actual SLAM trajectory presented in Fig. 4.24 shows that the robot passed accurately through narrow passages like doors and corridors. Figure 4.24 also shows comparison between final SLAM and odometry trajectories. From the comparison it can be seen that SLAM has managed to estimate accurate trajectory although odometry used for prediction has accumulated high amount of error during the experiment. Both SLAM and odometry trajectory start from the initial robot's pose marked with green dot, and the ending poses are marked with red dots.

Figure 4.23 shows computation time of updating the global map after the new state is added to the trajectory. It can be seen that the mean time of the global map update is around 250 ms, but there are spikes in the computation time which are the result of loop closing detection after which additional steps described in Sec. 4.3.3 were performed. In ideal case, there would be only one spike after every loop closing and computation time would then return back to low values at the next global map update between consecutive local maps. However, in reality, computation time increases/decreases gradually before/after

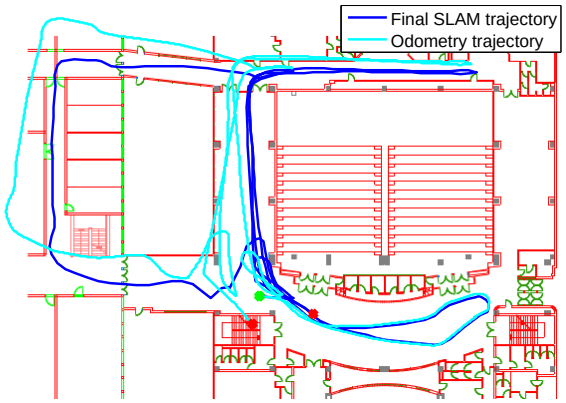

Figure 4.24: Comparison between the odometry trajectory (cyan) and SLAM trajectory (blue).

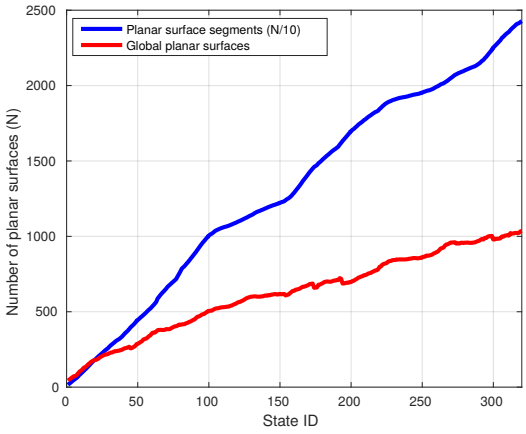

Figure 4.25: Total number of planar surface segments (scaled by 10) and global planar surfaces.

the loop closure. Because the LIDAR has very long range upon entering already mapped areas, planar surface segments that belong to the surfaces already included in the global map are segmented from newly acquired point clouds before the loop closing happens, and consequently planar surface segments are matched only with previous local maps which results in new global planar surfaces being added to the global map. They are combined with already existing ones in the refinement step, which is why computation time rises even before the loop closing. When the loop closing happens, additional steps further extend the computation time, especially if the trajectory changed significantly. Since mapping works in parallel with the trajectory estimation, new states are added to the trajectory while update of the global map after the loop closing is being performed. Because of this, local maps from newly added states are not immediately incorporated into the global map. When the global map update is done, there is more than one local map that has to be incorporated into the global map using algorithm described in Sec. 4.3.3. They are incorporated at the next update of the global map, but since there are several of them, this update also takes longer than when incorporating only one local map, and consequently more than one states could be added to the trajectory before the update is complete. This is why the computation time gradually decreases after the loop closing. From the map update time, it can also be seen that used loop closing technique performs well by selecting only highly informative loop closing states. Although the platform was moving along previously traversed trajectory over significant distances and came close to several of previously added states, only one loop closing was initiated.

Figure 4.25 shows the total number of planar surface segments contained within all global planar surfaces (blue) and the total number of global planar surfaces after every global map update (red). At the end of the experiment, out of the 24273 planar surface segments in all global planar surfaces, the final global map contains only 1036 surfaces, which proves that the number of planar surface segments merged in the global map is significant. It can also be seen that the number of planar segments in the global map sometimes decreases after the global map is updated with new local maps. This is direct result of merging multiple global planar surfaces in the same global planar surface.

Figure 4.26 shows the complete global 3D map of the area built by the SLAM system, after the last state was added to the SLAM trajectory. The roof plane was removed from the map in order to show interior structure. It can be seen that reobserving places does not introduce duplicate planar surface segments into the map. This is because localization has remained accurate and all re-observed planar surface segments are correctly merged into one global plane. It can also be seen that the final global map contains only static environment features, i.e. map building method managed to filter out the moving objects because either they represent outliers in the matching process or can not be described as strong planar features. Video of the experiment is available online[5]. In order to at least qualitatively estimate modelling accuracy of the SLAM system, 2D ground plan of the test area was extracted from the global 3D model and plotted over 2D CAD ground plan of the building in Fig. 4.27. It can be seen that they align well.

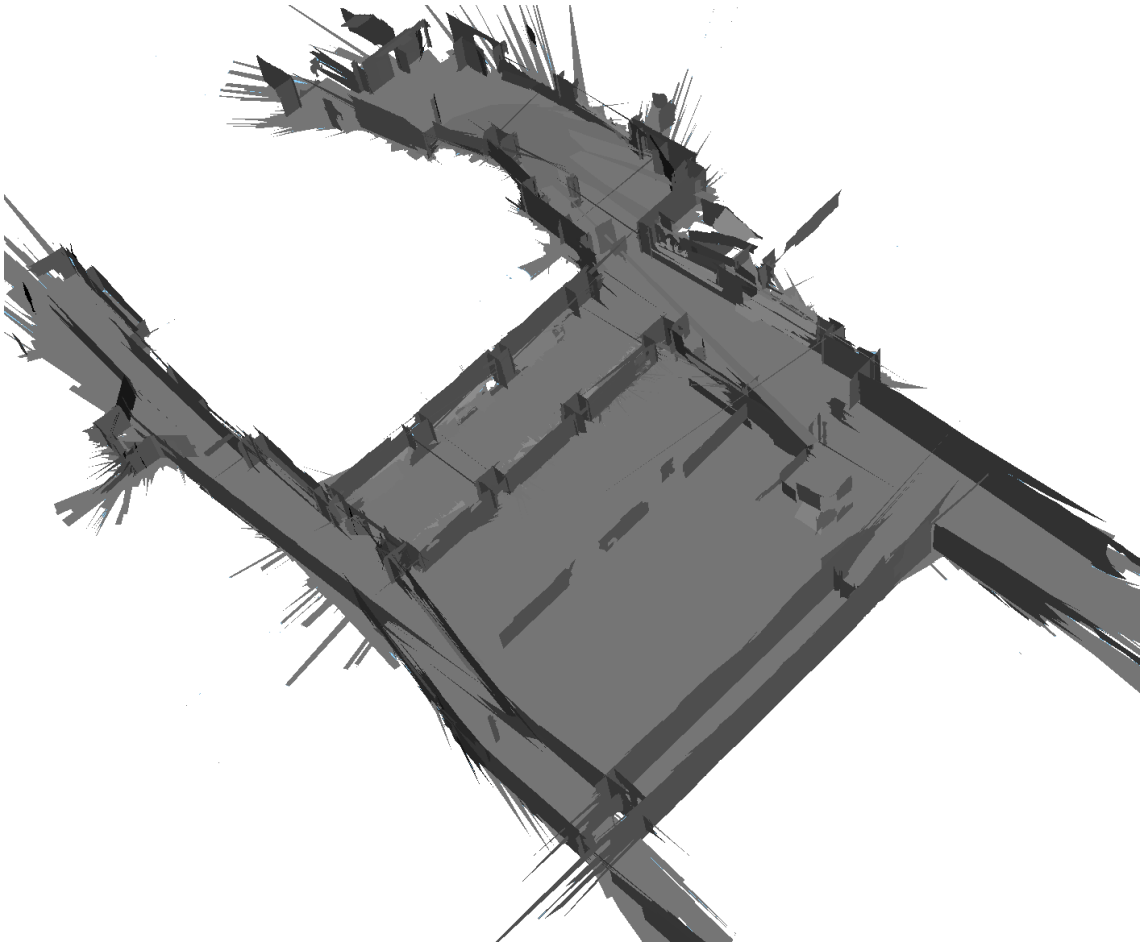

Figure 4.26: 3D model of the test area.

Outdoor experiment. Outdoor experiment was conducted using publicly available dataset from Ford Motor Company [148]. The dataset was acquired with a Ford F-250 pickup truck driving through downtown Dearborn. The vehicle was equipped with a professional (Applanix POS LV) and consumer (Xsens MTI-G) IMU, a differential GPS, a Velodyne

[5] https://youtu.be/vWoS_9wSNJw

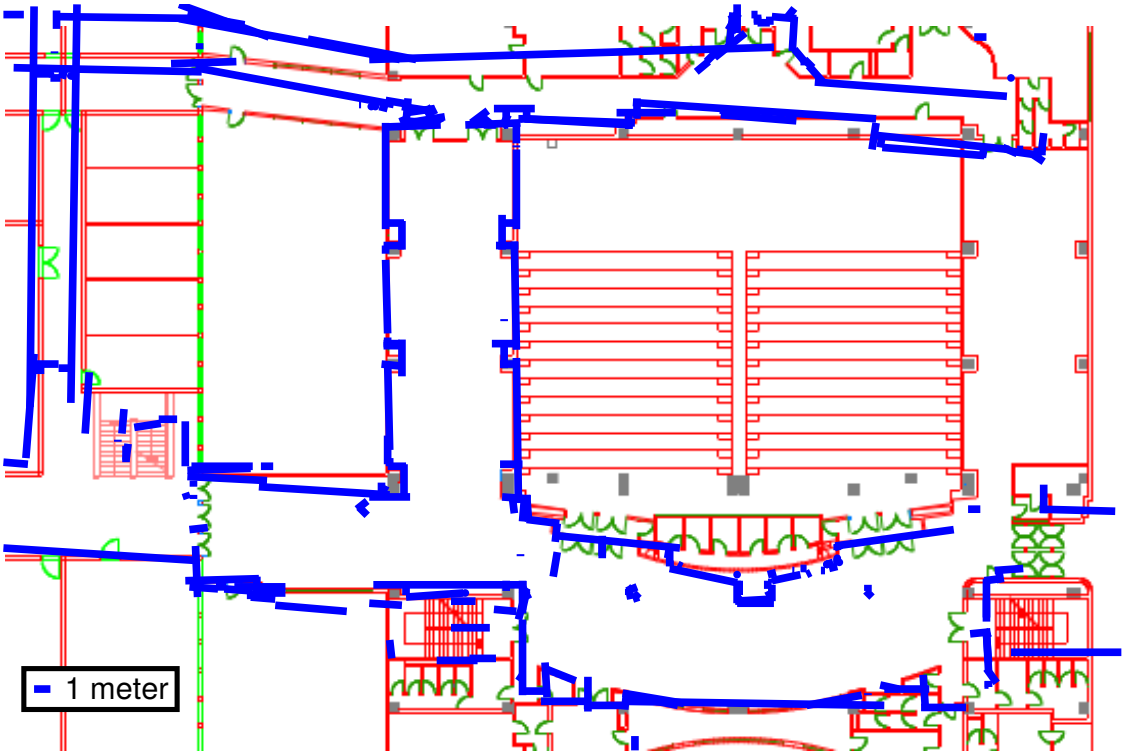

Figure 4.27: Ground plan from SLAM (blue) overlayed over the CAD ground plan (red).

HDL-64E 3D LIDAR, two push-broom forward looking Riegl LIDARs and a Point Grey Ladybug3 omnidirectional camera system. Velodyne HDL-64E has 64 vertical laser beams which is twice as much as Velodyne HDL-32E LIDAR used in the indoor experiment. The vertical FOV of the Velodyne HDL-64E is 40°, which is the same as Velodyne HDL-32E. Since rotation rate is also 10Hz it generates double the number of points per scan. The data from the differential GPS has been used as a ground truth. Odometry for prediction was acquired the same way as in the indoor experiment by fusing the data from Xsens IMU and the wheel encoders data contained within the Applanix POS LV raw sensor measurements. Figure 4.28 shows the test vehicle Ford F-250 equipped with sensors and sample images taken by the vehicle's camera while driving downtown Dearborn. It can be seen that there were multiple moving objects in the area as well as that the dataset was collected during daytime and represents real world scenario for urban environment. Average velocity of the vehicle was 20km/h while the maximum velocity was 45km/h. Total distance travelled was around 1.5km. Resolution of all three image planes used for projecting point cloud was set to 1024×297.

In order to show real time capability of the developed SLAM system even with LIDAR with higher resolution, every acquired point cloud was segmented and relative poses were calculated between consecutive local maps. Figure 4.29 shows segmentation times for point clouds (mean/max value is around 250ms/375ms) and Fig. 4.30 shows time required to compute relative poses (mean/max value is around 1.8ms/3ms). Segmentation time is larger than in indoor experiment since the resolution of every image plane used for projecting point cloud had to be increased in order to accommodate for higher resolution of Velodyne

Figure 4.28: Vehicle used in the dataset collection and sample images taken from the environment.

HDL-64E compared to Velodyne HDL-32E (64 instead of 32 vertical laser beams) and to allow projection of planar surface segments from larger distances.

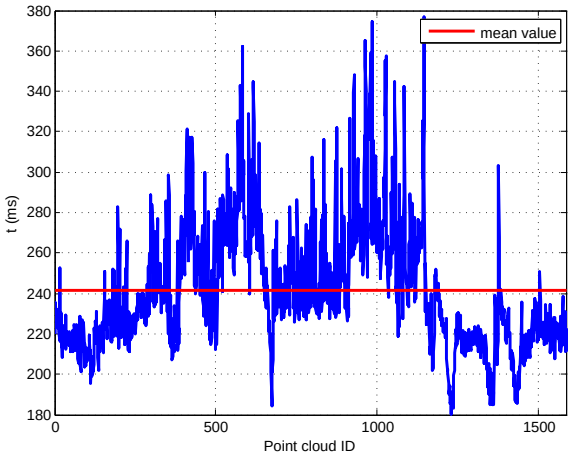

Figure 4.29: Point cloud segmentation time.

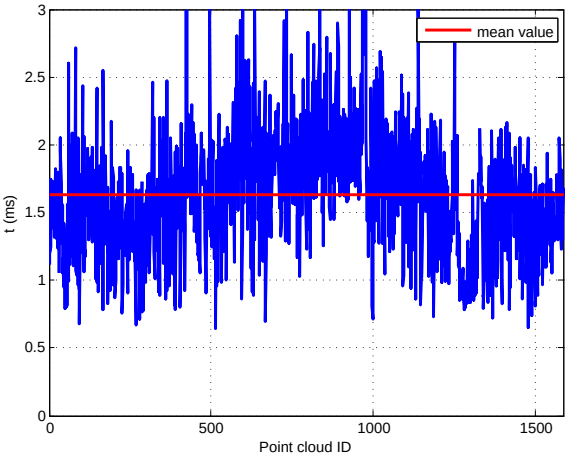

Figure 4.30: Relative pose computation time.

In total 280 states were added to the SLAM trajectory. SLAM, odometry and ground truth trajectories are shown in Fig. 4.31. Since loop closing is possible only at the end of the drive, first part of the trajectory does not change much because loop closing at the end has very little information gain for that part of the trajectory. Two loop closings were detected, first between states (X_{267}, X_{19}) and second between states (X_{280}, X_0). Corrections that SLAM made can be best seen on lower part of the trajectories (marked with dashes in Fig. 4.31) where the SLAM trajectory is much more precise than odometry trajectory and is almost identical to the ground truth trajectory. RMS error of SLAM and odometry trajectories with the respect to the ground truth trajectory has also been calculated

$$\begin{aligned} RMSE_{SLAM} &= \sqrt{\frac{\sum_{k=1}^{n} d(X_k, GT_k)}{n}} \\ RMSE_{odom} &= \sqrt{\frac{\sum_{k=1}^{n} d(O_k, GT_k)}{n}}, \end{aligned} \tag{4.75}$$

where n is the number of states in the trajectory, $d(X_k, GT_k)$ is Euclidean distance between state X_k of the SLAM trajectory and the ground truth at time step k and $d(O_k, GT_k)$ is

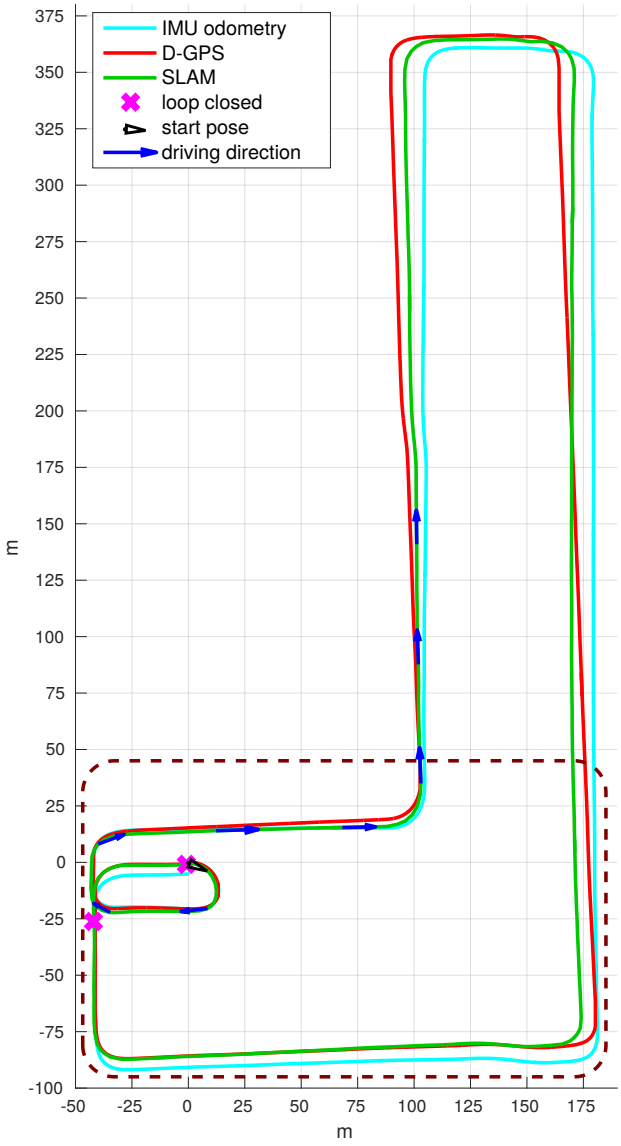

Figure 4.31: Comparison between odometry, ground truth and SLAM trajectory. Dashes mark the area where SLAM correction is most significant.

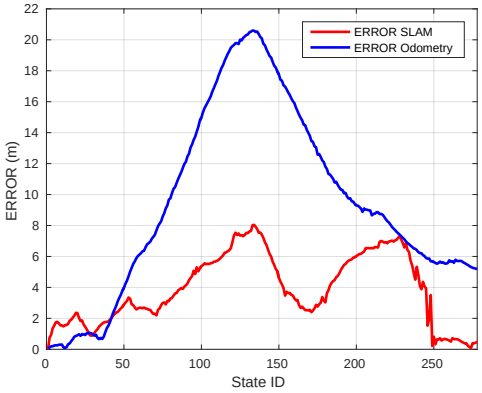

Figure 4.32: Absolute errors of final odometry trajectory (blue) and SLAM trajectory (red).

Euclidean distance between odometry and the ground truth trajectories at time step k. The calculated RMS errors are:

$$RMSE_{SLAM} = 4.48m \qquad RMSE_{odom} = 11.22m$$

The error of the SLAM trajectory is about 2.5 times smaller than the error of the odometry trajectory. What is more important, the final poses of the SLAM and the ground truth trajectories are almost the same, which means that if the vehicle had continued to move, its estimated pose would remain accurate whereas odometry error would increase further. This would result in even more expressed odometry versus SLAM RMS error ratio in the second lap with additional big loop closing events. Figure 4.32 shows the absolute error of the final odometry and the SLAM trajectories. The absolute error is calculated as Euclidean

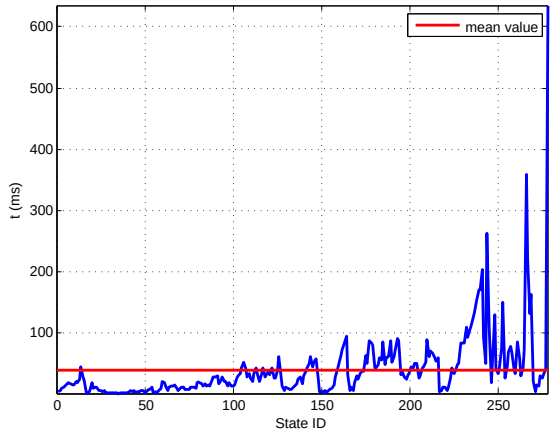

Figure 4.33: Global map update time after each augmented state.

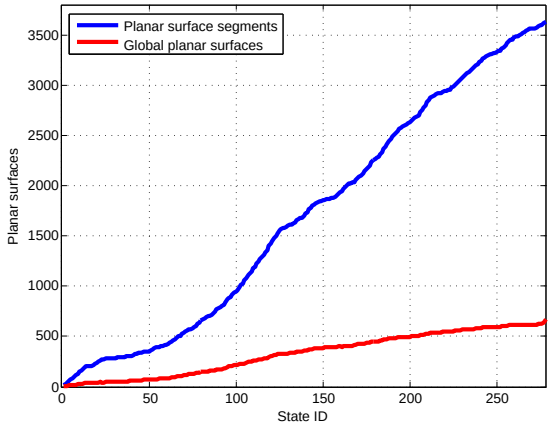

Figure 4.34: Total number of planar surface segments and global planar surfaces.

distance between the ground truth pose at time step k and SLAM state X_k (red) / odometry pose at time step k (blue). It can be seen that our SLAM system is able to recover from relatively large localization drift accumulated in large-scale environments.

The global map update time is shown in Fig. 4.33. We can seen that the time increases as more global planes are added to the map but remains in real-time domain throughout the experiment. Spikes in the update time are due to the loop closing detections. The reasons for the residual spikes are the same as in the indoor experiment. Figure 4.34 shows the number of global planar surfaces compared to the number of planar surface segments within them after each map update. At the end of the experiment, there were 3631 planar surface segments and only 667 global planar surfaces. Merging algorithm has reduced the number of planar surfaces in the global map by 5.45 times. Since there is no ground truth for the map in this dataset, only qualitative map accuracy analysis can be performed by showing generated planar global map together with the SLAM trajectory in Fig. 4.35. Several areas of the global map are zoomed in for better representation. As in the indoor experiment, it can be seen that there are almost no duplicate planes. However, some of the moving objects (e.g. cars) are present in the map (marked with dotted circles) since their speed was too low when first observed and could not be differentiated from the static objects. Video of the outdoor experiment is also available online[6].

4.6.3 *Test results for the active SLAM component*

In order to test the active SLAM component, the same platform from Fig. 4.19 was used. However, it was additionally equipped with the SICK LMS100-1000 2D LIDAR to accommodate for the exploration and path planning limitations. The robot was driven in the same indoor environment as for the indoor SLAM testing. As explained, there is no ground truth for the robot trajectory in that environment. However, accuracy of both map building and the robot trajectory achieved by the presented SLAM solution was already proven by previous two sets of experimental results. The main goal when testing the active SLAM was to prove that using active SLAM can drastically increase the accuracy of the exploration

[6] https://youtu.be/HboixGB2umY

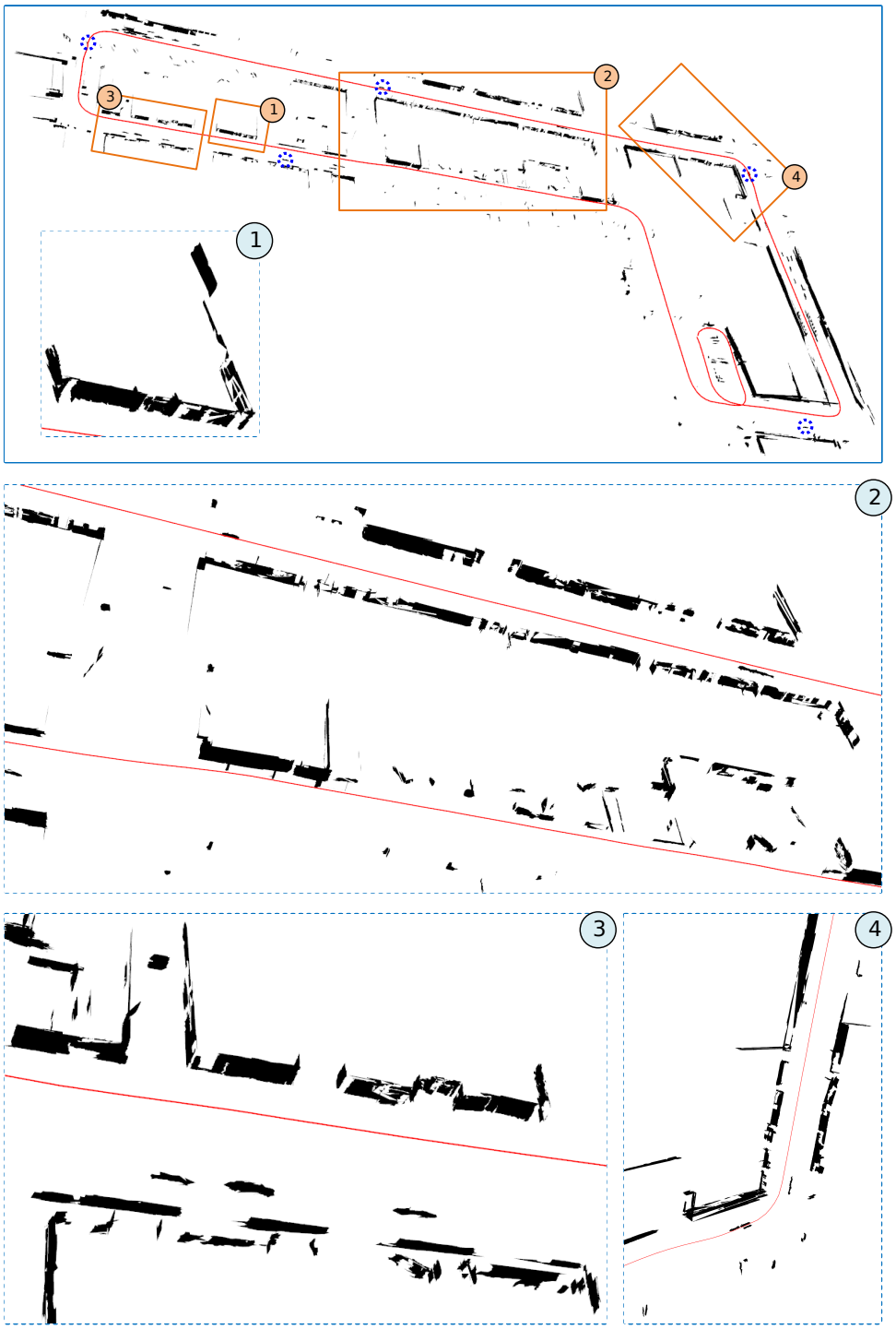

Figure 4.35: Global map generated from Ford dataset. Dotted circles represent moving objects present in the map.

algorithm in comparisons when it it not used. In order to do that two experiments in the same environment were completed: the first one with an active SLAM turned off and the second one with an active SLAM turned on. The accuracy was tested by comparing the 2D map created by the union of point clouds recorded by a 2D laser at each time step during motion and taken from locations estimated by the SLAM. From the map generated in this way it is easy to spot localization error. As the error increases, the point clouds that should

represent a scan of the same location overlap less and less, thus creating distortions in the map.

Figure 4.36 shows the generated map and the robot trajectory when an active SLAM was turned off. Although the trajectory in the first (left) room robot explored seems like loop closing should be detected, it was not. The robot's angle of rotation was too different and the RGB images did not match. This is a clear example where an active SLAM would help. Since this large loop closing was missed, odometry errors that accumulated over time were not corrected. As a result, we have a moderate localization error making distortions of the map in the first room explored and even more distortions in the second room that was explored.

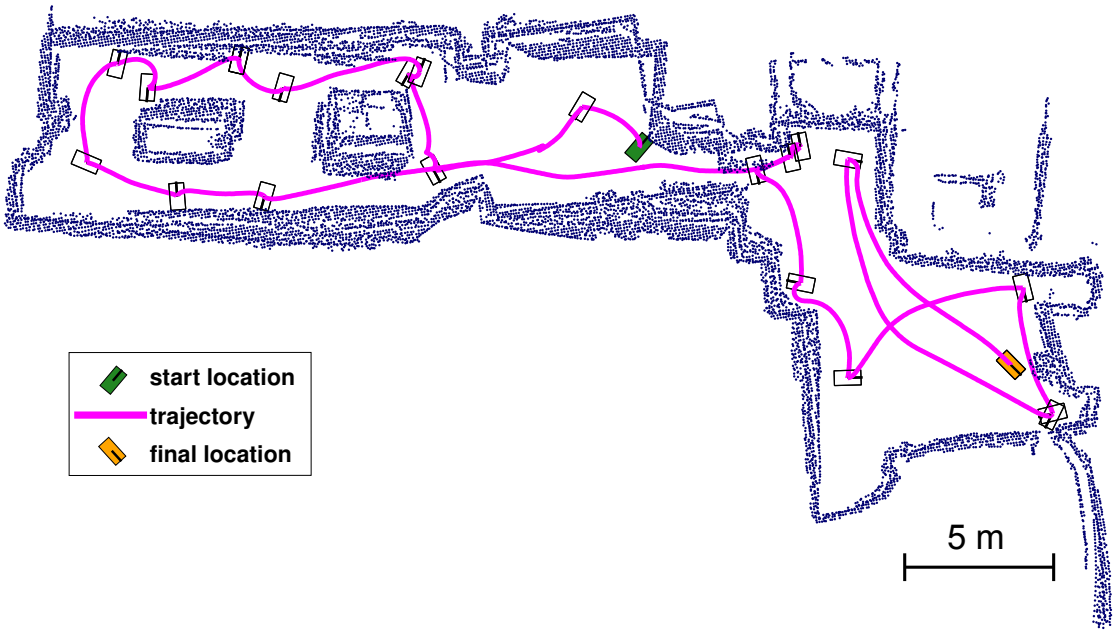

Figure 4.36: Generated map and trajectory without active SLAM. Black rectangles represent the robot footprint at all goals it was sent to.

Figure 4.37 shows a map of the same area but generated with an active SLAM turned on. Robot's trajectory is divided in three main parts. The first part (marked with magenta color) represents a robot's motion before an active SLAM cancelled the exploration. Since there were no major loop closures up till the state 18 was added, an active SLAM started searching for the possible loop closures immediately after the next trajectory augmentation. The maximum Euclidean distance (${}^{E}d_{max}$), for the second condition, was set to 8m. We can see that this condition was also met for several states during the robot's motion from state 18 to a location marked with red 'x'. The only condition left to be satisfied, in order for the loop closing process to start, was minimal topological distance. Parameter ${}^{T}d_{min}$ was set to 25m. When a robot arrived at the position marked with a red 'x', the state 2 satisfied this condition. In that moment the exploration was cancelled and the robot was sent to the state 2. When it arrived at the state 2 it continued to follow the previously traversed path up till the state 29 was augmented. This part of the robot's motion is shown in Fig. 4.37 as the green trajectory. When the state 29 was added into the trajectory, a loop closure was detected between the state 10 and the state 29. Since this was a large loop closure, the robot exited

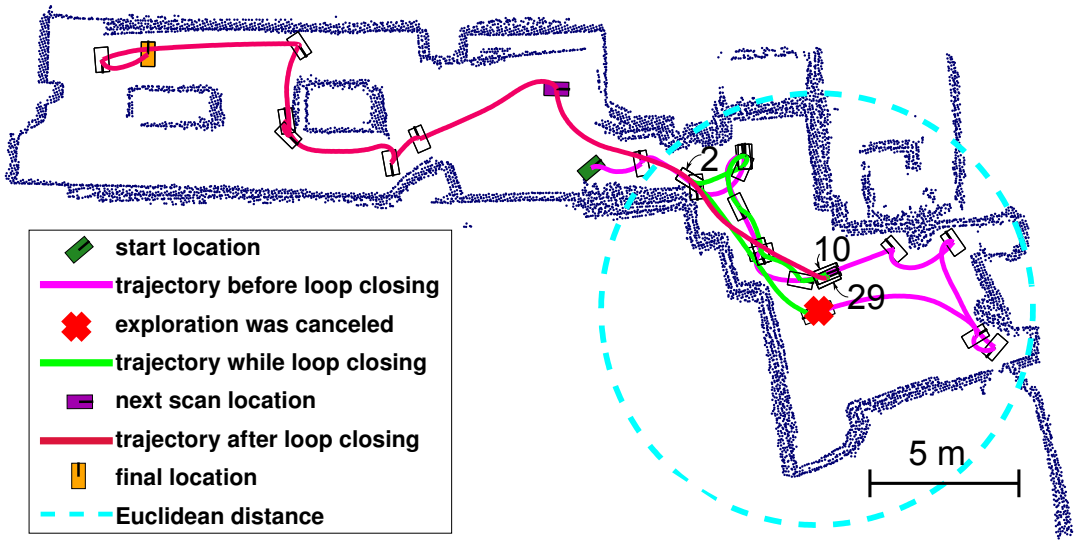

Figure 4.37: Generated map and trajectory with active SLAM. Black rectangles represent the robot footprint at all exploration goals the robot was sent to, with the exception of rectangle labelled with the number 29. It represents a SLAM state in which the robot closed the loop and continued to the previously set exploration goal. Cyan circle represents maximum Euclidean threshold ${}^{E}d_{max}$

loop closing process and started going to the goal previously set by the exploration (marked with purple icon in Fig. 4.37). As it can be seen from the last part of the robot's trajectory, no other location was suitable for an active SLAM to initiate the loop closing and the robot continued exploring until the whole area was explored. The generated map is much better compared to a map from experiment without an active SLAM. Both rooms have very little deformations. The passage between the two rooms is clear, scans of objects and walls in the second room that robot visited are almost completely overlapped and the angle between the two rooms is correct. These two experiments show that there is a considerable improvement in the map accuracy when the active SLAM is used.

The main drawback of the proposed solution is the requirement for switching between exploration and active SLAM goals. This is done due to the inability of the planner to take into account possible information gain from closing the SLAM loop. The negative side of this can best be seen from Fig. 4.37. The trajectory shown in the green color is traversed while the robot was searching for the loop closing required by the active SLAM and is a complete waste of time from the exploration perspective. This was solved in [149]. There, a modification to the D* algorithm was presented which allowed it to take into account negative costs of the cells used to plan the trajectory. The value of the negative cell is proportional to the information gain from closing the loop by moving the robot through that cell. This information gain is calculated using explained topological distance approach. Using this approach means that there is no requirement to stop the robot from exploring, but instead the possible loop closing states are taken into account during the planning of the trajectory to the exploration goal. Although, this drastically increases exploration speed, it does not have significant impact on the accuracy of the active SLAM which is why no additional experiments were conducted for comparing the SLAM accuracy with this algorithm.

4.7 SUMMARY

In this chapter, a fast active planar surface 3D SLAM solution based on ESDSF back-end was presented that is designed to work on full field of view 3D point clouds obtained from the 3D LIDAR measurements. Besides exploiting the sparsity of the ESDSF information matrix, there are three key elements in the derived SLAM front-end that allow the presented SLAM solution to work fast in large-scale environments. First is the efficient processing of 3D point clouds achieved by projecting them onto 2D image planes and then performing segmentation of projected point clouds into planar surface segments. Second is the adaptation of pose constraint calculation algorithm developed in [135], which was initially intended for use with RGB-D cameras. It has been modified to respect new uncertainty model of the 3D-LIDAR and to work with planar surface segments extracted from 360° field of view. Also, it takes into account SLAM trajectory as initial guess for generating pose constraint which reduces the number of outliers and speeds up the calculation time. Third key element is the algorithm for planar global map generation. Instead of simply transforming segmented planar surfaces into one coordinate frame, a new technique has been developed that combines all planar surface segments that lie on the same plane in the environment into one global planar surface whose parameters are estimated based on the uncertainty models of every planar surface segment contained within. Using this approach has significantly reduced the number of planar surface segments in the global map since all re-observed segments and segments that belong to the same plane are represented as one global planar surface. The result is the global map which requires much less memory and consequently allows fast processing. Additionally an active SLAM extension to the derived front-end is presented which gives the SLAM solution the ability to influence the robot movement and ensure more frequent loop closing detections, thus increasing the overall accuracy. It benefits from the used ESDSF back-end in two ways. First it uses its state representation to allow the robot to easily follow its previous path until loop detection occurs. Second, it does not use uncertainty as a fixed measure for cancelling the exploration since it can lead to a longer and repetitive loop closures, but instead counts the number of states added into the trajectory without the update step being triggered.

The effectiveness of the SLAM solution has been demonstrated on three sets of experiments. First set consist of experiments designed to test planar segmentation and registration algorithm. It was conducted on the two very different publicly available datasets and the algorithm was compared to 5 state-of-the-art-methods. The results proved that it can match and even outperform them in both accuracy and speed. The second set of experimental results was intended to test the accuracy of the complete SLAM solution. It was done using one publicly available dataset collected with commercially available vehicle and one dataset collected while the mobile platform was driving through our university building. These experiments confirmed that the proposed SLAM algorithm managed to significantly improve accuracy of both vehicles trajectories as well as generate planar global map with high reduction in the number of planar surface segments compared to the total number of planar surface segments in the local maps. Finally, the third set of experiments was conducted to test SLAM solution with the active component. It proved that the addition of this active component can significantly increase mapping and localization accuracy in the event that

SLAM is coupled with higher level algorithm such as the exploration algorithm.

5

Exactly sparse delayed state filter on Lie groups for Long-term SLAM

This chapter consists of two parts. The first part presents a second scientific contribution of the present thesis published in [150], which introduces a novel exactly sparse delayed state filter on Lie groups (LG-ESDSF). The second part of this chapter presents a third scientific contribution submitted for review in [151], which allows SLAM back-ends based on ESDSF to work over long-term while the robot is continuously moving through the same large-scale environment.

Regardless of the SLAM back-end version, there is always the need to estimate the robot pose and poses of map landmarks which inherently in 3D reside on $\mathsf{SE}(3)$. From the SLAM solutions overview given in the previous chapters, we can see that filtering solutions dominantly rely on the Euler angles or quaternions for representing these poses. Although sufficient, filtering with these representations within Euclidean frameworks does not represent the natural way of characterizing uncertainties and relations between the state vector elements. This problem was also encountered when using quaternions in ESDSF back-end in the previous chapter. Because of rotation quaternions requirement to remain unit quaternions normalization was required after every update. unit hypersphere (see works of [152] and [153]). This is in contrast to the state-of-the-art graph optimization back-ends, which relied more on using the insights of Lie groups and Lie algebras within the framework. This is one of the main reasons why accuracy of the filtering approaches was generally not on par with the accuracy of the graph based SLAM solutions. It is also the main reason why, after graph-optimization approaches solved computational complexity, they became dominantly used back-ends in the modern SLAM solutions. Some filtering approaches did utilize insights from Lie groups and applied it, to an extent, on the EKF and PF SLAM, as in [154], but in the end, resorted to the graph optimization framework and used Lie group insights to define and manipulate the graph edges, as in [155]. In the end, although some modifications were done to the filter itself, what lacked was a deeper structural change to the EKF itself. Changes in this field began to happen only recently with the introduction of the EKF on Lie groups (LG-EKF) in [156]. By representing the states on Lie groups, and performing filtering equations in the pertaining Lie algebra, LG-EKF is able to respect the geometry of the state space, thus achieving greater estimation accuracy of both the mean and the covariance. Solutions to the unscented Kalman filter on Lie groups (LG-UKF) also appeared in [157], then were followed by the continuous-discrete EKF on

Lie groups in [158], and invariant filters on Lie groups of [159]. However, in order to bear the potential for state-of-the-art SLAM performance, the solution to the information form of the LG-EKF, i.e., the extended information filter on Lie groups (LG-EIF), was still missing; nevertheless, this was solved recently in the work presented in [160]. Now, with the LG-EIF developed, the basis were set to develop a SLAM solution that is not only capable of using sparse structure of the SLAM information form, but which also respects the state space geometry by representing states on Lie groups. The ESDSF on Lie groups (LG-ESDSF) presented in [150] retains all the good characteristics of the classic ESDSF implementation, but also respects the state space geometry by negotiating uncertainties and employing filtering equations directly on Lie groups. LG-ESDSF derived this way is able to attain state-of-the-art performance comparable to graph optimization based SLAM back-ends. This is proven by direct comparison with g^{2}o in which LG-ESDSF achieves slightly better accuracy with much faster computation times. Moreover, LG-ESDSF coupled with stereo based front-end has been submitted for online evaluation protocol available within the KITTI dataset [84]. Currently, it ranks second among the stereo vision approaches and first among all the tested visual SLAM solutions.

Although respecting the state space geometry does enhance SLAM accuracy, all SLAM solutions have a limited operation time after which they are unable to perform trajectory and map optimization in real time. This is due to the accumulation of states which inevitably occurs regardless on the used approach to optimization. The algorithm presented in [151] allows the SLAM based on ESDSF to maintain real time performance by removing states that have little new information. This means that while the robot moves through the same environment information matrix remains almost constant in size. More importantly, this is achieved without destroying the sparsity of the information matrix thus ensuring continuous fast calculation of the update step.

The rest of the chapter is organized as follows. In the following section the preliminaries of the Lie group and Lie algebra are explained. Next, the derivation of the ESDSF on Lie groups is presented. Afterwards, details of the algorithm which allows real time performance of ESDSF based SLAM in the long-term are given. Finally, experimental results, proving the effectiveness of the LG-ESDSF and its long-term capability, are presented.

5.1 LIE GROUP AND ALGEBRA PRELIMINARIES

This section serves as a brief introduction to the necessary prerequisites for derivation of the ESDSF on Lie groups, while for a more rigorous treatment of the subject the reader is referred to [161]. A Lie group G is a group which has the structure of a smooth manifold. A tangent space $T_X(\mathsf{G})$ is associated to $X \in \mathsf{G}$ such that it is placed at the group identity, called Lie algebra $\mathfrak{g}$, and then transferred to any $X \in \mathsf{G}$ by applying corresponding (left or right) action ([162]). The Lie algebra $\mathfrak{g}$ is an open neighbourhood around the zero-element in the tangent space of G at the identity. In the present thesis matrix Lie groups are used which are usually the ones considered in engineering and physical sciences.

The matrix exponential $\exp_{\mathsf{G}}$ and logarithm $\log_{\mathsf{G}}$ establish a local diffeomorphism

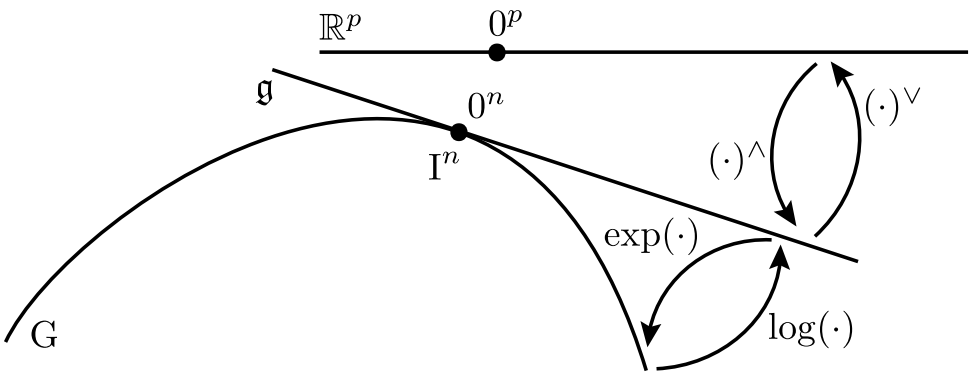

Figure 5.1: An illustration of mappings within the triplet of Lie group G, Lie algebra $\mathfrak{g}$ and the Euclidean space $\mathbb{R}^p$.

between the group and the pertaining algebra

$$\exp_{\mathsf{G}} : \mathfrak{g} \to \mathsf{G} \quad \text{and} \quad \log_{\mathsf{G}} : \mathsf{G} \to \mathfrak{g}\,. \tag{5.1}$$

The Lie algebra $\mathfrak{g} \subset \mathbb{R}^{n\times n}$ associated to a p-dimensional matrix Lie group $\mathsf{G} \subset \mathbb{R}^{n\times n}$ is a p-dimensional vector space defined by a basis consisting of p real matrices E_r, $r = 1, .., p$, often referred to as generators, see [163]. Furthermore, a natural relation between $\mathfrak{g}$ and $\mathbb{R}^p$ is given through a linear isomorphism by

$$[\cdot]^\vee_{\mathsf{G}} : \mathfrak{g} \to \mathbb{R}^p \quad \text{and} \quad [\cdot]^\wedge_{\mathsf{G}} : \mathbb{R}^p \to \mathfrak{g}\,. \tag{5.2}$$

For brevity, following notation of [164] will be used:

$$\exp^\wedge_{\mathsf{G}}(x) = \exp_{\mathsf{G}}([x]^\wedge_{\mathsf{G}}) \quad \text{and} \quad \log^\vee_{\mathsf{G}}(X) = [\log_{\mathsf{G}}(X)]^\vee_{\mathsf{G}}\,, \tag{5.3}$$

where $x \in \mathbb{R}^p$ and $X \in \mathsf{G}$. An illustration of the above mappings is given in Fig. 5.1. In addition, two more operators are required—the adjoint representation of a Lie group and Lie algebra, respectively denoted as Ad_{G} and ad_{G}. They appear due to general non-commutative nature of matrix Lie groups, i.e., $XY \neq YX$. However, the non-commutativity can be captured by the so-called adjoint representation of G on $\mathfrak{g}$ as follows

$$X \exp^\wedge_{\mathsf{G}}(y) = \exp^\wedge_{\mathsf{G}}(\mathrm{Ad}_{\mathsf{G}}(X)y)X, \tag{5.4}$$

where $X \in \mathsf{G}$, $y \in \mathbb{R}^p$. This can be seen as a way of representing the elements of the group as a linear transformation of the group's algebra. The adjoint representation of $\mathfrak{g}$, ad_{G}, is in fact the differential of Ad_{G} at the identity. A more detailed discussion on these concepts and the used notation can be found in [161].

The first thing required to make use of ESDSF on Lie groups, is to establish an error distribution on Lie groups. If a random variable describing the error, $\varepsilon \triangleq \log^\vee_{\mathsf{G}}(X^I)$, is tightly focused around the identity element X^I, it can be well described with a Euclidean Gaussian in the pertaining algebra $\varepsilon \sim \mathcal{N}_{\mathbb{R}^p}(\mathbf{0}^{p\times 1}, P)$ as in [165], and then it is said that X follows a concentrated Gaussian distribution (CGD) on G around the identity element (confer [158] for details). This distribution is then regarded as a distribution on Lie groups at the identity element, but which can further be translated over G by using the left action. Finally, by

combining the error distribution definition and left action, a random variable $X \in \mathsf{G}$ is defined as

$$X = \mu \exp_{\mathsf{G}}^{\wedge}(\varepsilon) \text{ , with } X \sim \mathcal{G}(\mu, P), \tag{5.5}$$

with mean value $\mu \in \mathsf{G}$, covariance $P \in \mathbb{R}^{p \times p}$, and $\mathcal{G}$ denoting the CGD as in [165]. In the derivation of LG-ESDSF, the special Euclidean group $\mathsf{SE}(3)$ is employed. Group details, together with the group operators are given in Appendix A.2.

In the next section, derivation of LG-ESDSF is explained using the Lie group fundamentals described here.

5.2 ESDSF ON LIE GROUPS

To derive the ESDSF on Lie groups several building blocks have to be developed. Fundamentally, the information form of the LG-EKF [158] is required and computation of the augmentation and marginalization of a CGD has to be computed. In [160] the problem of the information form was solved and the extended information filter on Lie groups (LG-EIF) was presented. The augmentation and marginalization of the CGD were presented in [164] in the context of iterated LG-EKF, where authors solved the prediction step by approximating the Chapman-Kolmogorov equation with a joint distribution and then marginalizing the posterior state. In this thesis the same train of thought is followed and this result is extended to the prediction equations of ESDSF on Lie groups.

5.2.1 *State space construction*

First, states X_i of the robot trajectory T_n are now represented by an $\mathsf{SE}(3)$ group element instead of quaternions and position vectors.

$$X_i = \begin{bmatrix} R_i & t_i \\ 0 & 1 \end{bmatrix}, X_i \sim \mathcal{G}(\mu_{X_i}, \Sigma_{X_{i,i}}), \tag{5.6}$$

where R_i is a member of the special orthogonal group $\mathsf{SO}(3)$, given as a 3×3 rotation matrix defining robot orientation in the global frame, and $t_i = [x_i, y_i, z_i]$ represents the robot position in the global frame. In contrast to the Euclidean ESDSF, the trajectory T_n is no longer a vector, but rather a block diagonal matrix consisting of n $\mathsf{SE}(3)$ elements, i.e.,

$$T_n = \begin{bmatrix} X_1 & 0 & 0 & 0 \\ 0 & X_2 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & X_n \end{bmatrix} \in \mathsf{G} = \mathsf{SE}(3) \times \ldots \times \mathsf{SE}(3). \tag{5.7}$$

However, following the idea of the information filter approach to LG-EIF presented in [160], rather than keeping the trajectory in the form of the matrix T_n, the states are stored in the

form of concatenated Lie algebra $\mathfrak{se}(3)$ elements

$$\begin{aligned} \tau_n = \log_{\mathsf{G}}^{\vee}(T_n) = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \log_{\mathsf{G}}^{\vee}(X_1) \\ \log_{\mathsf{G}}^{\vee}(X_2) \\ \vdots \\ \log_{\mathsf{G}}^{\vee}(X_n) \end{bmatrix}, \\ x_i \sim \mathcal{N}(\mu_{x_i}, \Sigma_{x_{i,i}}) = \mathcal{N}(\eta_{x_i}, \Lambda_{x_{i,i}}), \\ \tau_n \sim \mathcal{N}(\mu_n, \Sigma_n) = \mathcal{N}(\eta_n, \Lambda_n), \end{aligned} \tag{5.8}$$

where $x_i \in \mathfrak{se}(3)$ represents the state $X_i \in \mathsf{SE}(3)$ mapped to the Lie algebra, while τ_n can be seen as a whole trajectory T_n mapped to the Lie algebra. The relations between the information matrix Λ_n, the information vector η_n, the trajectory mean value in Lie algebra μ_n, and the covariance matrix Σ_n follow the same equations as in the standard ESDSF, i.e., $\mu_n = \Lambda_n^{-1}\eta_n$ and $\eta_n = \Sigma_n^{-1}\mu_n$.

5.2.2 *Motion model*

The motion prediction is assumed to follow a non-linear first order Markov process similarly to (4.4), except that the motion is now defined directly on G as

$$X_{n+1} = f(X_n, \Omega_n, w_n) = X_n \exp_{\mathsf{G}}^{\wedge}(\Omega_n + w_n), \tag{5.9}$$

where $X_n \in \mathsf{G}$ is the state, $w_n \sim \mathcal{N}_{\mathbb{R}^p}(0, Q_n)$ is a p-dimensional white Gaussian process noise, and $\Omega_n = [\Delta t, \Delta r]$ represents a robot displacement measured by odometry between X_n and X_{n+1}. Change in the position component is represented by $\Delta t = [\Delta x, \Delta y, \Delta z]$, while the change in rotation Δr is represented using the Lie algebra parametrization of the special orthogonal rotation group $\mathsf{SO}(3)$ (Euler-axis convention). The process noise covariance Q_n represents uncertainty of the odometry. The state covariance matrix is propagated as follows

$$\Sigma_{n+1} = \mathcal{F}_n \Sigma_n \mathcal{F}_n^{\mathrm{T}} + \Psi(\Omega_n) Q_n \Psi(\Omega_n)^{\mathrm{T}} \tag{5.10}$$

$$\mathcal{F}_n = \mathrm{Ad}\left(\exp_{\mathsf{G}}^{\wedge}(-\Omega_n)\right) + \Psi(\Omega_n)\mathscr{C}_k, \tag{5.11}$$

where Ψ is the right Jacobian of G (see [166]), while $\mathscr{C}_k$ denotes the linearization of the motion model (5.9) at X_n as in [158], which is given as

$$\Psi(v) = \sum_{m=0}^{\infty} \frac{(-1)^m}{(m+1)!} \mathrm{ad}(v)^m, \quad v \in \mathbb{R}^p, \tag{5.12}$$

$$\mathscr{C}_n = \frac{\partial}{\partial \varepsilon} \Omega\left(X_n \exp_{\mathsf{G}}^{\wedge}(\varepsilon)\right)|_{\varepsilon=0}. \tag{5.13}$$

Since $\Omega(\cdot)$ is not a function of the state X_n, (5.13) evaluates to zero, i.e., $\mathscr{C}_n = 0$, hence (5.10) evaluates to

$$\mathcal{F}_n = \mathrm{Ad}\left(\exp_{\mathsf{G}}^{\wedge}(-\Omega_n)\right). \tag{5.14}$$

For brevity, the following notation is introduced

$$\mathcal{Q}_n = \Psi_n Q_n \Psi_n^{\mathrm{T}}, \quad \Psi_n = \Psi(\Omega_n). \tag{5.15}$$

5.2.3 *Prediction step*

In the prediction step of the LG-ESDSF, similarly as in the prediction step of the ESDSF, the trajectory is first augmented with the new state X_{n+1}. Afterwards, depending on the marginalization threshold, the state X_n can be either kept or marginalized. Given that, if the threshold is exceeded, the state X_n is permanently kept as a part of T_{n+1}.

AUGMENTATION. Following the equations for state augmentation in standard ESDSF (4.13) and (4.14) and newly derived matrices $\mathcal{F}_n$ and $\mathcal{Q}_n$ the distribution parameters Λ_{n+1} and η_{n+1} associated to the trajectory τ_{n+1} evaluate to

$$\eta_{n+1} = \begin{bmatrix} \mathcal{Q}_n^{-1}(\mu_{x_{n+1}} - F\mu_{x_n}) \\ \eta_{x_n} - \mathcal{F}_n^T \mathcal{Q}_n^{-1}(\mu_{x_{n+1}} - \mu_{x_n}) \\ \eta_{x_{n-1}} \\ \vdots \end{bmatrix},$$

$$\Lambda_{n+1} = \begin{bmatrix} \mathcal{Q}_n^{-1} & -Q_n^{-1}\mathcal{F}_n & 0 \\ -\mathcal{F}_n^T Q_n^{-1} & \Lambda_{x_{n,n}} + \mathcal{F}_n^T \mathcal{Q}_n^{-1}\mathcal{F}_n & \Lambda_{x_{n,n-1}} \\ 0 & \Lambda_{x_{n-1,n}} & \Lambda_{x_{n-1,n-1}} \\ \vdots & \vdots & \vdots \end{bmatrix}.$$

MARGINALIZATION. If the marginalization threshold is not exceeded and the marginalization needs to be performed, the previous state X_n is removed, while the new state X_{n+1} becomes X_n. The equations for combining augmentation and marginalization are in the vein of the iterated LG-EKF prediction presented in [164]; however, for the information form and LG-ESDSF the procedure is different, which is why the resulting expressions for η_n and Λ_n are presented

$$\eta_n = \begin{bmatrix} \mathcal{Q}_n^{-1}\mathcal{F}_n\beta_n^{-1}\eta_{x_n} + \alpha_n(\mu_{x_{n+1}} - \mathcal{F}_n\mu_{x_n}) \\ \eta_{x_{n-1}} - \Lambda_{x_{n-1}x_n}(\eta_{x_n} - \mathcal{F}_n^T\mathcal{Q}_n^{-1}(\mu_{x_{n+1}} - \mathcal{F}_n\mu_{x_n})) \\ \eta_{x_{n-2}} \\ \vdots \end{bmatrix}$$

$$\Lambda_n = \begin{bmatrix} \alpha_n & \mathcal{Q}_n^{-1}\mathcal{F}_n\beta_n^{-1}\Lambda_{x_{n,n-1}} & 0 \\ \Lambda_{x_{n-1,n}}\beta_n^{-1}\mathcal{F}_n^T\mathcal{Q}_n^{-1} & \gamma_n & \Lambda_{x_{n,n-1}} \\ 0 & \Lambda_{x_{n,n-1}} & \Lambda_{x_{n-1,n-1}} \\ \vdots & \vdots & \vdots \end{bmatrix},$$

where

$$\begin{aligned} \alpha_n &= (\mathcal{Q}_n + \mathcal{F}_n\Lambda_{x_{n,n}}^{-1}\mathcal{F}_n^T)^{-1}, \\ \beta_n &= (\Lambda_{x_{n,n}} + \mathcal{F}_n^T\mathcal{Q}_n^{-1}\mathcal{F}_n), \\ \gamma_n &= \Lambda_{x_{n-1,n-1}} - \Lambda_{x_{n-1,n}}\beta_n^{-1}\Lambda_{x_{n,n-1}}. \end{aligned} \tag{5.16}$$

Since now equations for both augmentation and marginalization steps have been derived to operate on Lie algebra, this concludes the derivation of the prediction step of the LG-ESDSF.

5.2.4 *Measurement model and LG-ESDSF update*

The discrete measurement model on matrix Lie groups is given as

$$Z_{n+1} = h(T_{n+1}) \exp^{\wedge}_{\mathsf{G}'}(\nu_{n+1}), \tag{5.17}$$

where $Z_{n+1} \in \mathsf{G}'$, $h : \mathsf{G} \to \mathsf{G}'$ is a $\mathcal{C}^1$ function, G' is a p'-dimensional Lie group and $\nu_{n+1} \sim \mathcal{N}_{\mathbb{R}^q}(\mathbf{0}^{q\times 1}, R_{n+1})$ is zero-mean white Gaussian noise with covariance R_{n+1}.

The update step in LG-ESDSF occurs following similar rationale as in the case of the Euclidean ESDSF SLAM, i.e., it is performed whenever a loop closing between any two states X_i and X_j is detected, and a relative pose measurement is delivered. However, in reality, update will almost always occur between the newly augmented state X_n and one or more states already in the trajectory. For this reason, in the remainder of this section, it is assumed that the trajectory T_{n+1} is being updated after the loop closing occurred between X_n and X_j, $0 < j < n$. No generality is lost since all equations are valid if X_n is substituted with X_i $(i \neq j, 1 < i \leq n)$. Hence, the filter operates using the relative poses of the LMR module as measurements, and the measurement function evaluates to

$$h(T_{n+1}) = X_j^{-1} X_n \in \mathsf{SE}(3)\,. \tag{5.18}$$

Accordingly, the innovation term of the LG-ESDSF SLAM is modelled by the following equation

$$z_{n+1} = \log^{\vee}_{\mathsf{G}}\left(h\left(T_{n+1}\right)^{-1} Z_{n+1}\right)\,. \tag{5.19}$$

For calculating the updated estimates of the information matrix Λ_{n+1} and the information vector η_{n+1}, LG-EIF update equations from [160] are used

$$\begin{aligned} \eta^{-}_{n+1} &= \mathcal{H}^{\mathrm{T}}_{n+1} R^{-1}_{n+1} z_{n+1}\,, \\ \Lambda^{-}_{n+1} &= \Lambda_{n+1} + \mathcal{H}^{\mathrm{T}}_{n+1} R^{-1}_{n+1} \mathcal{H}_{n+1}\,, \end{aligned} \tag{5.20}$$

with R_{n+1} being the measurement uncertainty reported by LMR and matrix $\mathcal{H}_{n+1}$ is evaluated as in [158]

$$\mathcal{H}_{n+1} = \frac{\partial}{\partial \varepsilon}\left[\log^{\vee}_{\mathsf{G}}\left(h(T_{n+1})^{-1} h\big(T_{n+1}\exp^{\wedge}_{\mathsf{G}}(\varepsilon)\big)\right)\right]\Bigg|_{\varepsilon=0}. \tag{5.21}$$

For the $\mathsf{SE}(3)$ group calculating the relative pose between states X_n and X_j reduces to simple matrix inverse and multiplication. Given the measurement model (5.18), the matrix (5.21) evaluates to

$$\mathcal{H}_{n+1} = [\underbrace{0\cdots}_{1:j-1\text{ zero bl.}} \overbrace{-\mathrm{Ad}(X_{n+1}^{-1})\,\mathrm{Ad}(X_j)}^{j\text{th block}} \underbrace{\cdots 0\cdots}_{j+1:n-1\text{ zero bl.}} \overbrace{I}^{n\text{th bl.}}\ 0].$$

Even though a similar result was obtained in [164] for relative pose averaging, it is without derivation, which is why, for completeness, it is provided in the Appendix A.3. This result shows that, similarly to the Euclidean ESDSF, the matrix $\mathcal{H}_{n+1}$ remains sparse consisting of n 6×6 blocks, among which only blocks j and n are non-zero. However, as explained in [160] for LG-EIF, this does not complete the update step of LG-ESDSF since at this point

Algorithm 3: LG-ESDSF SLAM back-end pseudocode

Set initial values of Λ_1, X_1, η_1, x_1
loop:
Get odometry data Ω_n
Perform motion model (5.9) to get X_{n+1}
Calculate matrices $\mathcal{F}_n$ and $\mathcal{Q}_n$ from (5.11) and (5.15)
if permanently adding state X_n to T_n **then**
  Calculate η_{n+1} and Λ_{n+1}
  if Loop closed between X_n and X_j **then**
    Get relative pose measurement Z_{n+1}
    Calculate innovation z_{n+1} (5.19)
    Do update via (5.20)–(5.24)
  end if
else
  Calculate η_n and Λ_n
end if

the mean value $\mu^-_{n+1} = (\Lambda^-_{n+1})^{-1}\eta^-_{n+1}$ is in general a non-zero vector, thus in collision with the CGD definition (5.5). To overcome this issue, the state reparametrization is performed as proposed in [158], and the final formulae are given as follows

$$\mu^-_{n+1} = (\Lambda^-_{n+1})^{-1}\eta^-_{n+1} \tag{5.22}$$

$$\Lambda^+_{n+1} = \Psi(\mu^-_{n+1})^{-T}\Lambda^-_{n+1}\Psi(\mu^-_{n+1})^{-1} \tag{5.23}$$

$$\eta^+_{n+1} = \Lambda^+_{n+1}\log^\vee_{\mathsf{G}}\left(\exp^\wedge_{\mathsf{G}}(\Lambda_n^{-1}\eta_n)\exp^\wedge_{\mathsf{G}}(\mu^-_n)\right). \tag{5.24}$$

Note that η^+_{n+1} and Λ^+_{n+1} differ from η_{n+1} and Λ_{n+1} which are obtained only via augmentation in the prediction step. This concludes the derivation of the update step of the LG-ESDSF. Pseudocode of the entire LG-ESDSF SLAM back-end is given in Algorithm 3.

5.2.5 *Covariance estimation and computational complexity analysis*

The same analysis for performance and covariance estimation made for ESDSF in Sec. 4.2.5 is also valid for LG-ESDSF. The main difference is that LG-ESDSF operates in such a way that by the CGD definition (5.5), the uncertainties are assigned to variables located in the Lie algebra. Since algebra of the special orthogonal group is given in the form of the Euler axis representation, the uncertainties have to be associated to the same variable type. Because of this, instead of using the UT to map uncertainty from Euler angles to quaternion orientation representation, UT is applied to associate uncertainties to the Lie algebra variables, i.e. the Euler axis representation.

Computational complexity of the prediction step is also the same as in ESDSF since the same number of blocks in information matrix are changed during marginalization and augmentation. Also, the auxiliary vector is used to solve the problem of retrieving the past states for the prediction and update step the same way as explained for the ESDSF. Because of this similarities, one of the only computationally demanding step in LG-ESDSF is the

same as in ESDSF, computation of information matrix inversion. It is required after every update to retrieve updated μ_{n+1} as in ESDSF. However, by examining equations (5.22) to (5.24) it would appear that there is an extra inversion requirement of Λ_{n+1} in (5.22). To solve this issue (5.24) can be rewritten as follows

$$\begin{aligned}\eta^{+}_{n+1} &= \Lambda^{+}_{n+1} \log^{\vee}_{\mathsf{G}} \left(\exp^{\wedge}_{\mathsf{G}}(\Lambda^{-1}_{n} \eta_{n}) \exp^{\wedge}_{\mathsf{G}}(\mu^{-}_{n+1}) \right) \\ &= \Lambda^{+}_{n+1} \log^{\vee}_{\mathsf{G}} \left(\exp^{\wedge}_{\mathsf{G}}(\mu_{n}) \exp^{\wedge}_{\mathsf{G}}(\mu^{-}_{n+1}) \right) \\ &= \Lambda^{+}_{n+1} \mu^{+}_{n+1} ,\end{aligned} \tag{5.25}$$

which means that there is no need for final computation of μ_{n+1} as it is already calculated within η^{+}_{n+1}. Therefore, there is also only a single inversion of the information matrix in LG-ESDSF performed in equation (5.22).

The last potentially time consuming calculation, which occurs in LG-ESDSF and not in ESDSF, is the inversion of $\Psi(\mu^{-}_{n+1})$ required in (5.23) during the update. Although $\Psi(\mu^{-}_{n+1})$ does have the same dimension as Λ^{-}_{n+1}, it is also a sparse matrix and keeps a strictly tridiagonal block form (not effected by the update); hence, its inversion reduces to n inversions of a 6×6 matrix.

This analysis concludes the derivation of LG-ESDSF. In the following section long-term capability of SLAM systems based on LG-ESDSF is discussed and algorithm is presented which allows LG-ESDSF to operate in real time during long term use in scenarios in which robot continuously moves through the same large-scale environment.

5.3 LONG-TERM SLAM BASED ON LG-ESDSF

When the robot continuously moves inside the same area for a longer period of time, regardless of the used SLAM back-end and front-end, operation time of any SLAM system is limited. This is because the number of features in the map and/or states in the trajectory constantly rises and increases memory and computation requirements. In order to ensure long-term operation, number of states and features has to be reduced continuously.

If the environment being explored is reasonable in size, most pose graph SLAM systems will be able to build a complete map of the environment. The problems will occur when the robot continues to move repeatedly through the same environment, since the state space will continue to grow. The easiest solution would be to stop the SLAM system once the map has been built and use the constructed map to localize the robot within. Although straightforward, the main drawbacks of this solution are that:

1. The robot cannot explore new areas and then return to the explored part without reinitializing the SLAM.

2. The robot can no longer improve map accuracy by closing loops.

A better solution would be to allow SLAM to function continuously, but manage the increase in the number of features and/or trajectory states.

The solution to this problem, developed in the present thesis, works under the assumption that when the robot moves through an already explored environment, majority of the

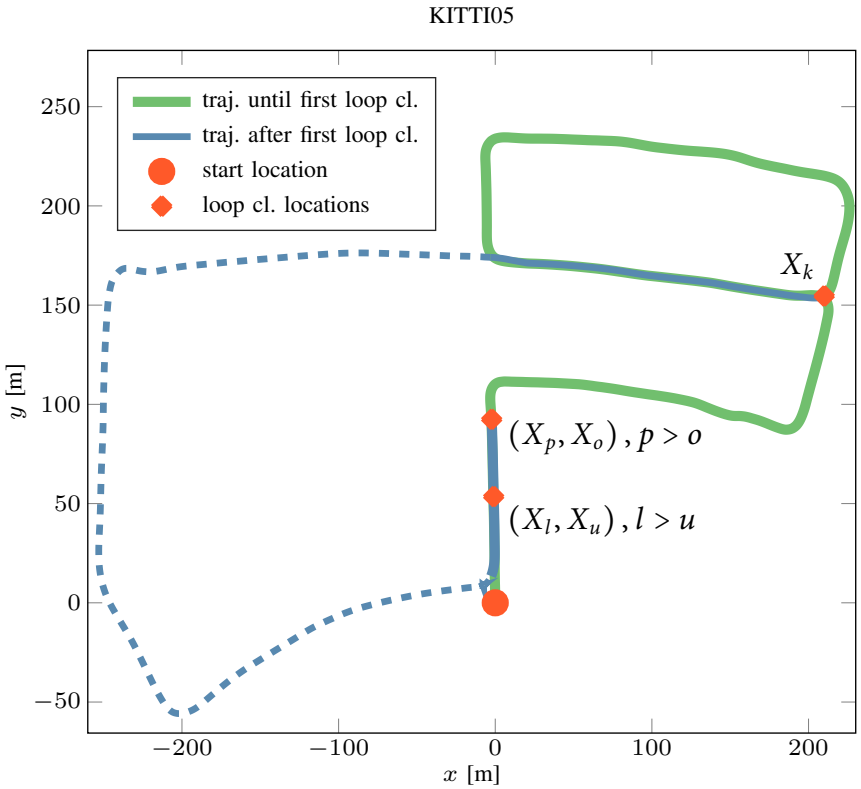

Figure 5.2: Example of a robot trajectory suitable for removing some already added states. First loop closing occurred in state X_k, second loop closing occurred between states X_l and X_u, and third loop closing occurred between states X_p and X_o. The first state in the pair designates the location where loop closing occurred. States X_o and X_u were added to the trajectory when the robot traversed the area for the first time, while states X_p and X_l were added when it arrived at the same place for the second time.

newly augmented states will have similar poses with the already existing states in the trajectory. Since the robot is equipped with the same sensor configuration, information gained from states with similar poses is very low. Given that, main goal of the long-term SLAM is formulated: *all states in the trajectory that have a similar pose to the states added previously, should be removed (marginalized) in order to preserve long-term real-time operation*. One could argue that there is no need to add new states that are close to the ones already in the trajectory in the first place. In the case of the SLAM base on ESDSF this is not true for two main reasons:

- Two states can only be marked as close enough after the update of the trajectory, because before the update, due to odometry error, states can be falsely further/closer from each other. By removing a state falsely detected as close, information is lost. On the other hand, if truly close states are kept, redundant information is retained.
- SLAM systems based on ESDSF need to add the state X_k to the trajectory, in order to perform update using information from closing the loop between the added state X_k and some other state X_j.

5.3.1 *Selection and marginalization of close states*

Let's consider a robot moving as shown in Fig. 5.2. The path between the two loop closings occurring at X_k and X_l, marked with the full line, contains states that were added during the traversal of the path that can be safely marginalized. This is because their measurement information is already contained within the states added before the first loop closing in X_k. Furthermore, since the loop closing also occurred in X_l, they truly have similar poses to states already contained in the trajectory. States augmented during the path marked with the dashed line should be kept, since they contain new measurement information. The states added before the loop closing in X_k could also be checked; however, this would be redundant as they were already checked when loop closing in X_k occurred. In order to determine how similar the poses of two states X_i and X_j are (i.e., how close they are) the same criteria (4.64) is used as for the computation of pose difference between two states during the calculation of the topological distance in the loop detection algorithm.

Now let's designate with β the set of all the states between X_l and X_p, from Fig. 5.2, that can be removed according to (4.64). Moreover, since the robot path between states X_l and X_p would be very similar to the path traversed between states X_u and X_o, without the loss of generality, let's assume that all the states X_i, $l < i < p$ are in β. The question arises, how to remove these states from the trajectory and from the information matrix? First, let's consider the case where states X_l and X_p, i.e., states which participated in loop closings, are not in β. The most important fact about marginalizing states added between two loop closings is that they are *connected only with the neighbouring states*. Given that, (4.18) and (4.19), which were used to marginalize X_n during the LG-ESDSF prediction step, can also be used here and the sparsity of the information matrix will remain preserved. The only changes in the information matrix after the removal of states β, besides size reduction, will be changes to the blocks related to states X_l and X_p. Moreover, since states in β are neighbouring states, they can be marginalized in a single block which increases computation speed in comparison to marginalizing just each state separately.

After successfully removing states in β, it has to be decided what to do with X_l and X_p. The simplest solution would be to never marginalize states included in the loop closings. However, in the case of the robot continuously moving through the same environment, a large number of loop closings can be expected; thus, continuously ignoring these states would inevitably result in the increase of the information matrix size. Therefore, states included in loop closings have to be marginalized in order to ensure the long-term capability. However, due to the nature of the loop closing procedure, only the state in which previous loop closing occurred needs to be marginalized, since the current state, e.g., X_p, will become in the next iteration the previous loop closing state. This is why in this example only X_l needs to be marginalized. The problem with removing loop closing state is that, as mentioned before, (4.18) and (4.19) should only be used if the state to be marginalized is connected only with its neighbouring states. However, this is never the case for states that participated in loop closings. If equations (4.18) and (4.19) were used for marginalization, as explained in the next section, this would have a negative effect on the sparsity of the information matrix. Possible solutions are to use an approach that preserves the sparsity and reduces the number of loop closings. In the present solution both have been implemented.

5.3.2 *Marginalization of states included in the loop closings*

In order to understand why marginalization of the state X_l would have a negative impact on the sparsity of the information matrix, let us assume that loop closing between X_l and X_u was the first and between X_p and X_o was the second loop closing (i.e., loop closing in X_k never occurred). Furthermore, let us assume that we have already marginalized all the states in β. If (4.18) and (4.19) were used to marginalize X_l, the result would be:

$$\Lambda = \begin{bmatrix} \Lambda_\alpha - \Lambda_{\alpha l}\Lambda_l^{-1}\Lambda_{l\alpha} & \Lambda_{\alpha\gamma} - \Lambda_{\alpha l}\Lambda_l^{-1}\Lambda_{l\gamma} \\ \Lambda_{\gamma\alpha} - \Lambda_{\gamma l}\Lambda_l^{-1}\Lambda_{l\alpha} & \Lambda_\gamma - \Lambda_{\gamma l}\Lambda_l^{-1}\Lambda_{l\gamma} \end{bmatrix}, \tag{5.26}$$

$$\Lambda_{\alpha l} = \begin{bmatrix} \cdots & \Lambda_{l,u} & \cdots & \Lambda_{l,l-1} \end{bmatrix}^T, \ \Lambda_{\gamma l} = \begin{bmatrix} \Lambda_{pl} \\ 0 \end{bmatrix}, \tag{5.27}$$

where Λ_α represents information from the states $\alpha = \{X_0,\ X_1 \cdots X_{l-1}\}$ augmented before the state X_l, Λ_γ represents information of states $\gamma = \{X_p,\ X_{p+1}\}$ and $\Lambda_{\alpha l}$, $\Lambda_{\alpha\gamma}$, and $\Lambda_{l\gamma}$ represent their cross-information. As the result of the following expression:

$$\Lambda_\alpha - \Lambda_{\alpha l}\Lambda_l^{-1}\Lambda_{l\alpha}, \tag{5.28}$$

two blocks, $\Lambda_{l-1,u}$ and $\Lambda_{u,l-1}$, will be inserted into the new information matrix, while four more blocks will be added by the following expressions:

$$\Lambda_{\alpha\gamma} - \Lambda_{\alpha l}\Lambda_l^{-1}\Lambda_{l\gamma}, \tag{5.29}$$

$$\Lambda_{\gamma\alpha} - \Lambda_{\gamma l}\Lambda_l^{-1}\Lambda_{l\alpha}. \tag{5.30}$$

In total, six new blocks will be inserted and seven existing blocks will be removed. While this may seem fine, the problems will start to occur during future marginalization. For example, after the third loop closing and marginalization of new β states, X_p is marginalized from the information matrix. The term $\Lambda_{\alpha p}$ then becomes

$$\Lambda_{\alpha p} = \begin{bmatrix} \cdots \Lambda_{p,u} & \cdots & \Lambda_{l,o} & \cdots & \Lambda_{l,l-1} \end{bmatrix}^T. \tag{5.31}$$

This would result in five more blocks from (5.28) and four more block from (5.30) and (5.29). Although (5.30) and (5.29) always add four blocks, number of new blocks added by (5.28) will continue to increase and the number of removed blocks will always remain seven. If this process would continue, the number of new blocks in matrix Λ would increase quadratically with the number of consecutively marginalized states that participated in loop closings. This describes the worse case scenario in which all the states from β are marginalized. If some states between consecutive loop closings are not in β and remain in the information matrix, or if we marginalization of loop closing states was periodically skipped, the loss of sparsity would be slowed down. However, in that case there would still be an increase in the number of states.

In order to solve the problem of losing sparsity of the information matrix, an approach to approximate the dense information matrix with a sparse one has to be found. Two most recent approaches applicable to SLAM based on ESDSF are the works presented in [167] and [168]. In [167] authors presented a solution for long-term SLAM by approximating

the dense sparse matrix with a sparse one using the Chow Liu tree (CLT) of [108]. CLT approximates a probability distribution $p(\theta)$ with $p'(\theta)$ in a way that (i) each variable is conditioned only on one other variable and that (ii) Kullback-Leibler divergence between p and p' is minimized:

$$p(\theta) = p(\theta_m) \prod_{i=1}^{m-1} p(\theta_i|\theta_{i+1}, \cdots, \theta_m) \tag{5.32}$$

$$\approx p(\theta_m) \sum_{i=1}^{m-1} p(\theta_i|\theta_{i+1}) = p'(\theta). \tag{5.33}$$

The authors used CLT to approximate only the elimination cliques disregarding constraints in the remainder of the pose graph. The authors then use the resulting CLT to compute the constraints which remain in the pose graph. In [168] the authors compute CLT from the entire information matrix which ensures taking into account all the pose graph constraints. Furthermore, the authors also introduced generic linear constraint (GLC) factors to ensure that the resulting information matrix will have full-rank. In the present solution the same sparsification method as in [168] is applied, but without the use of GLC factors as it is assumed that all measurements are such that the full-rank of the information matrix will always be preserved.

By using the aforementioned sparsification method, the sparsity of the information matrix can be ensured, regardless of the states chosen for marginalization. However, it should be noted that with every approximation some information is lost and error in the SLAM trajectory increases. Moreover, sparsification creates some computational overhead and increases the update complexity. Given that, sparsification is not performed after every marginalization of β, but only when the information matrix becomes so dense that its inversion takes too long to be acceptable for real-time operation.

Although updating trajectory after each loop closing increases SLAM accuracy, as explained in Sec. 4.3.4, closing larger loops affects accuracy more than closing the smaller ones. Furthermore, there are also some negative effects of performing update after every loop closing in the terms of long-term SLAM solution presented here:

1. Sparsification is required more frequently.

2. There will be less states in β that can be marginalized faster.

This is why topological distance approach for rejecting loop detections with little information gain, explained in Sec. 4.3.4, is also used here. This ensures, that by changing the minimum topological distance threshold ${}^{T}d_{min}$, requirement for sparsification can be controlled. If frequent repeated loop closings are allowed, sparsification is required more often, while at the same time little information is gained. By reducing the number of accepted loop closings more states can be marginalized out faster which increases the algorithm speed. Of course not too many consecutive loop closings can be rejected, as there is a risk of increasing the trajectory error beyond allowable tolerances. The complete algorithm for reducing the number of states is summarized in Algorithm 4.

Algorithm 4: State number reduction

Last update performed after loop closed between X_a, X_i
if Loop closed between X_b, X_j $(b > j)$ **then**
  Compute topological distance ${}^T d_{j,b}$
  if ${}^T d_{j,b} \geq {}^T d_{min}$ **then**
    Perform steps 10-12 from Algorithm 3
    For all states between X_a and X_b find f_c^m (4.64):
    $f_c^m = \min(f_c(m,n),\ 0 < n < a-1),\ a \leq m < b$
    Put all states satisfying $f_c^m \leq f_c^{max}$ in set β
    Put all neighbouring states from β into blocks
    Marginalize blocks using (4.18) and (4.19)
    Set $X_a = X_b$
    If required, perform sparsification
  end if
end if
* ${}^T d_{min}$ and f_c^{max} are predefined thresholds

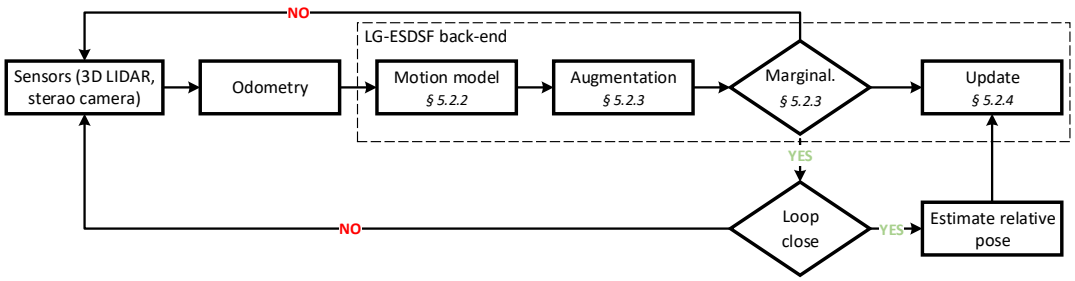

Figure 5.3: Schematic layout of the proposed LG-SLAM system.

5.4 EXPERIMENTAL RESULTS

In order to test the effectiveness of the LG-ESDSF back-end and its long-term capability it needs to be coupled with the SLAM front-end. In the remainder of this chapter complete SLAM solution based on LG-ESDSF is referred to as LG-SLAM. The schematic layout of LG-SLAM is shown in Fig. 5.3 The experimental testing of the LG-SLAM is divided into three different scenarios. First, LG-SLAM is compared with two state-of-the-art visual SLAM algorithms, namely ORB-SLAM and LSD-SLAM, which use g^{2}o as the back-end. Second, LG-ESDSF and g^{2}o back-ends are compared by using two different front-ends: (i) stereo odometry with feature tracking (SOFT) [169], and (ii) three-dimensional normal distributions transform (3D-NDT) [106]. Finally, the LG-SLAM long-term capability is tested by making it work continuously in the same environment. All algorithms were implemented using the C++ programming language. Testing machine was a computer with Intel Core i7@2.6 Ghz processor and 8 GB of RAM. For solving the sparse matrix equations the Eigen library [170] was used.

Testing was conducted using two public datasets, the KITTI vision benchmark suite [84] and the EuRoC dataset [171]. The KITTI dataset consists of 22 sequences recorded on

different routes under different conditions. Ground truth is provided only for the first 11 sequences while others are used for online evaluation. The dataset offers measurements acquired by 3D LIDAR Velodyne HDL-64E, 2 grayscale cameras Point Grey Flea 2 in stereo configuration and two color cameras Point Grey Flea 2 also in stereo configuration. Ground truth is provided by an accurate inertial navigation system OXTS RT 3003. For the recording of the dataset all sensors were mounted on a commercially available vehicle Volkswagen Passat. The EuRoC dataset contains in total 11 sequences, out of which five were recorded in a large machine hall and six were recorded in a so-called Vicon room (i.e., a room equipped with the Vicon motion capture system). For every sequence, measurements were recorded using the VI sensor [172] mounted on a hexacopter UAV AscTec Neo. The VI sensor provides stereo images and synchronized data from the inertial measurement unit (IMU). Depending on available texture, brightness, and UAV dynamics each sequence is labelled as easy, medium or difficult. Ground truth is provided by the Vicon motion capture system and a laser tracking system, depending on the environment.

Although the KITTI dataset provides its own metric, it does not evaluate absolute errors between the ground truth and the estimated results, but rather compares errors on parts of sequences that are from 100 to 800 metres long, hence the benefits obtained from loop closing only marginally affect the metric. As such, it is designed primarily for evaluating pure odometry rather than complete SLAM systems. Because of this the automatic evaluation tool developed for the EuRoC dataset available online[1] was also used, which relies on evaluation of the absolute error. However, it first tries to find the best fit between the tested and ground truth trajectories and then computes the error. This is why a metric based on (4.64) is also used and rotational e_{rot} and translation e_{tran} errors were evaluated separately without any fitting. The root-mean-squared-error (RMSE) was then calculated and is provided with the rest of the results. Herein, the error calculated using the online tool is referred to as e_F, while RMSE of the absolute translational error and absolute rotational error are referred to as e_{trans} and e_{rot}, respectively. Since in the KITTI dataset only tracks 0, 2, 5, 6, 7, and 9 provide suitable loop closures for SLAM front-end based on stereo, results are provided only for these tracks.

5.4.1 *Experimental comparison of LG-SLAM, ORB-SLAM, and LSD-SLAM*

Although the stereo version of LSD-SLAM is not available as open source, the results of testing LSD-SLAM, as well as ORB-SLAM, on the KITTI and EuRoC datasets are available in their respective papers. Since e_{trans} and e_{rot} could not be calculated, for this test, only e_F is provided in Tables 5.1 and 5.2 for all solutions if available (results for LSD-SLAM are available for only three out of 11 EuRoC sequences while both ORB-SLAM and SOFT failed to produce meaningful result for the final EuRoC track, probably because of incorrect calibration parameters). Results for the pure SOFT odometry are also provided, which is the only front-end used in this scenario, since both LSD-SLAM and ORB-SLAM are stereo visual SLAM solutions.

Given the results on the KITTI dataset in Table 5.1, it can be seen that LG-SLAM achieved the best results on all the tracks except in two cases. In particular, LSD-SLAM shows the

[1] http://vision.in.tum.de/data/datasets/rgbd-dataset/tools# evaluation

Table 5.1: Results of LG-SLAM, LSD-SLAM and ORB-SLAM on the KITTI dataset.

	e_F [m]			
	SOFT	LG-SLAM	ORB-SLAM	LSD-SLAM
KITTI00	3.36	1.18	1.3	**1.0**
KITTI02	5.52	3.12	5.7	**2.6**
KITTI05	1.54	**0.59**	0.8	1.5
KITTI06	0.96	**0.49**	0.8	1.3
KITTI07	0.4	**0.32**	0.5	0.5
KITTI09	2.42	**1.26**	3.2	5.6

Table 5.2: Results of LG-SLAM, LSD-SLAM and ORB-SLAM on the EuRoC dataset. MH stands for datasets recorded in machine hall, while V stands for dataset recorded in the Vicon room. E, M and D depict easy, medium and difficult sequences respectively.

	e_F [cm]			
	SOFT	LG-SLAM	ORB-SLAM	LSD-SLAM
MH_01_E	17.2	**3.6**	4.0	-
MH_02_E	7.8	5.0	**4.3**	-
MH_03_M	16.8	3.9	**3.5**	-
MH_04_D	32.8	7.5	**7.1**	-
MH_05_D	24.4	6.0	**5.3**	-
V1_01_E	10.8	**4.8**	8.7	6.6
V1_02_M	14.0	**4.8**	6.4	7.4
V1_03_D	32.7	**4.7**	7.2	8.9
V2_01_E	16.2	7.0	**6.1**	-
V2_02_M	22.4	8.2	**5.6**	-

best performance on sequences KITTI00 and KITTI02. It can also be seen that LG-SLAM significantly improved SOFT results in all the tracks. Figure 5.4 shows LG-SLAM, SOFT and ground truth trajectories for the sequence KITTI00. When the EuRoC dataset results shown in Table 5.2, are analysed it can be noticed that LG-SLAM outperformed both solutions in the sequences taken in the Vicon 1 room, while ORB-SLAM was better in 4 sequences from the Machine Hall and 2 sequences from Vicon room 2. As in the case of the KITTI dataset LG-SLAM again significantly improved the accuracy with respect to the SOFT odometry.

Additionally, LG-SLAM was compared online on the KITTI dataset, using the built-in evaluation protocol. For this purpose LG-SLAM was tested on the remaining 10 sequences and the overall result achieved by LG-SLAM can be seen in Table 5.3 together with other three best ranked visual SLAM solutions (two of them being ORB-SLAM and LSD-SLAM). The complete results with details are also available online[2], and, at the time of writing, the proposed approach ranks second among the stereo vision approaches and first among all tested visual SLAM solutions.

[2] http://cvlibs.net/datasets/kitti/eval_odometry.php

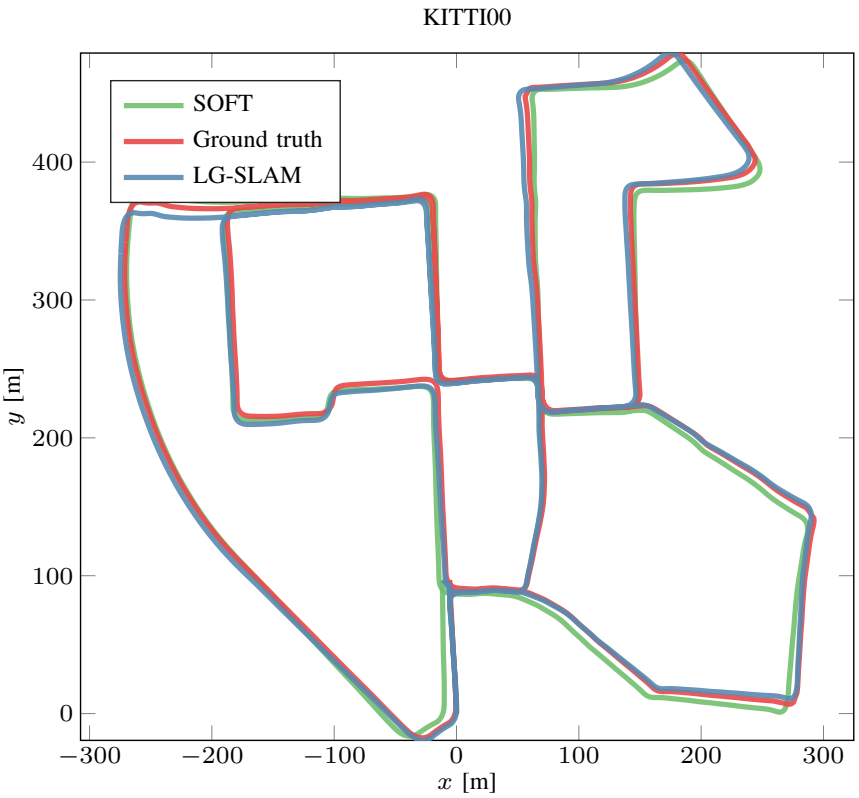

Figure 5.4: LG-SLAM results on the KITTI sequence KITTI00

Table 5.3: KITTI rankings of the state-of-the-art stereo vision SLAM systems at the time of writing.

Method	Transl.	Rot. [°/m]	Sensors
LG-SLAM	0.82 %	0.0020	stereo cameras
ORB-SLAM2	1.15 %	0.0027	stereo cameras
S-PTAM	1.19 %	0.0025	stereo cameras
S-LSD-SLAM	1.20 %	0.0033	stereo cameras

5.4.2 *Experimental comparison of LG-SLAM and g²o*

Even though both LSD-SLAM and ORB-SLAM use g²o as the back-end, the front-ends are different. This is why, in order to conduct a fair comparison between LG-ESDSF and g²o, clean g²o and LG-ESDSF back-ends have been used and coupled with SOFT and 3D-NDT front-ends. It was also ensured that both back-ends have exactly the same number of states, that states are added precisely after the same stereo pair or point cloud was processed, and that the same uncertainty model is used. First, both back-ends were compared on the KITTI dataset.

The results for all the three metrics are listed in Table 5.4 for the SOFT and in Table 5.5 for the 3D-NDT front-end. It can be seen that when using SOFT, LG-ESDSF outperforms g²o in 4 out of 6 sequences in all three metrics. When using 3D-NDT neither of the solutions has an advantage over the other. The error in the track KITTI02 is large, because NDT accumulated a rotational error in the beginning and the vehicle did not return to the same area again and the error could not be corrected, even though 3D-NDT performed quite

Table 5.4: Comparison of g^{2}o and LG-ESDSF on the KITTI dataset with the SOFT front-end.

	e_{trans} [m]	e_{rot} [deg]	e_F
	LG-ESDSF / g^{2}o		
KITTI00	4.30 / **4.18**	1.35 / **1.26**	**1.17** / 1.22
KITTI02	12.53 / **12.27**	**1.16** / 1.47	**3.10** / 4.19
KITTI05	**1.48** / 1.56	0.53 / **0.44**	**0.57** / 0.78
KITTI06	**1.09** / 1.53	**0.77** / 0.93	0.51 / **0.47**
KITTI07	**0.97** / 1.06	**0.63** / 0.68	**0.32** / 0.36
KITTI09	**2.62** / 2.65	**0.77** / 0.81	1.26 / **1.11**

Table 5.5: Comparison of g^{2}o and LG-ESDSF on the KITTI dataset with the 3D-NDT front-end.

	e_{trans} [m]	e_{rot} [deg]	e_F
	LG-ESDSF / g^{2}o		
KITTI00	**8.62** / 8.93	**3.60** / 4.71	**2.27** / 2.49
KITTI02	**98.87** / 120.54	**19.29** / 23.92	**47.13** / 65.51
KITTI05	**2.87** / 3.38	1.92 / **1.65**	1.34 / **1.01**
KITTI06	3.33 / **3.27**	2.25 / **1.25**	**1.98** / 2.08
KITTI07	1.22 / **1.08**	0.97 / **0.95**	0.64 / **0.62**
KITTI09	14.80 / **13.84**	4.05 / **3.35**	**3.34** / 3.35

well afterwards.

The computation time of optimization and update steps were also compared for g^{2}o and LG-ESDSF with both front-ends. Recorded times are given in Tables 5.6 and 5.7 for the SOFT and 3D-NDT front-ends, respectively. Provided are maximum t_{max}, minimum t_{min} and mean t_{mean} computation times for all 6 sequences. From the results it can be seen that all the computation times of the LG-ESDSF update steps are significantly lower compared to those of the g^{2}o optimization. Furthermore, since update time of LG-SLAM is only dependent on the number of loop closings and number of states in the trajectory, we can seen that computation times for LG-SLAM are similar regardless of the used front-end. On the other hand g^{2}o optimization times depend on the initial condition. However, it can also be noted that the minimum computation times of g^{2}o are much lower compared to its maximum computation times, which is due to fast optimization if loop closings appear frequently. Nevertheless, even minimum computation times of g^{2}o optimization are higher than that of LG-ESDSF update step. In KITTI07 and KITTI09 minimum optimization times of g^{2}o are similar to maximum optimization times, because there were very few loop closings in these trajectories.

Since the EuRoC dataset contains only stereo images, 3D-NDT algorithm could not be used. Hence LG-ESDSF and g^{2}o were compared only using the SOFT front-end. Results for all 10 sequences are provided in Table 5.8. As can be seen LG-ESDSF outperforms g^{2}o in about 80% of sequences. However, it should be noted that since trajectories are shorter, although more dynamic than in the KITTI dataset, the differences are also relatively small. Table 5.9 shows computation times of the update step for LG-ESDSF and optimization step

Table 5.6: Minimum, maximum and mean computation times of the LG-ESDSF update step and g^{2}o optimization on the KITTI dataset with the SOFT front-end.

	t_{min} [ms]	t_{max} [ms]	t_{mean} [ms]
	LG-ESDSF / g^{2}o		
KITTI00	11.09 / 50.54	52.10 / 987.17	28.83 / 461.26
KITTI02	32.99 / 49.21	43.66 / 869.65	37.92 / 342.02
KITTI05	8.47 / 40.98	24.42 / 640.31	15.08 / 338.96
KITTI06	7.48 / 35.71	13.50 / 318.87	10.03 / 230.61
KITTI07	5.89 / 184.29	8.07 / 213.08	6.54 / 203.94
KITTI09	13.10 / 458.67	16.58 / 494.56	14.90 / 481.31

Table 5.7: Minimum, maximum and mean computation times of the LG-ESDSF update step and g^{2}o optimization on KITTI dataset with the 3D-NDT front-end.

	t_{min} [ms]	t_{max} [ms]	t_{mean} [ms]
	LG-ESDSF / g^{2}o		
KITTI00	10.59 / 52.31	37.69 / 957.15	26.72 / 595.73
KITTI02	31.12 / 2.13	38.29 / 997.68	35.50 / 193.22
KITTI05	8.27 / 37.07	21.56 / 627.26	14.05 / 325.59
KITTI06	7.07 / 30.84	12.96 / 305.91	9.26 / 244.75
KITTI07	5.74 / 191.10	6.93 / 197.94	6.26 / 194.65
KITTI09	14.13 / 453.10	15.70 / 470.50	15.01 / 462.49

for g^{2}o. Same conclusions can be drawn as for the KITTI dataset, which states that LG-ESDSF completes the update step faster than g^{2}o completes the optimization. For example, maximum computation times (t_{max}) of the LG-ESDSF update step are between 10 to 30 times shorter than those of g^{2}o optimization.

As a general conclusion it cannot be said that one method clearly outperforms the other. Although, when all the tested situations are looked at, LG-ESDSF provides more accurate overall results, the differences are small in several cases. However, at the very least it can be said that LG-ESDSF is comparable to g^{2}o in accuracy, while its main advantage lies in shorter computation time of the update steps with respect to g^{2}o optimization. Moreover, the computation time of LG-ESDSF can be more easily predicted and accounted for. Another advantage is the fact that there is no need to set parameters like the maximum number of optimization steps and there is no problem of local minima in LG-ESDSF. In conclusion, it can be asserted that by negotiating uncertainties and employing filtering equations on Lie groups within the ESDSF framework, thus respecting the state space geometry, state-of-the-art SLAM performance was achieved with a veteran filtering-based SLAM approach.

5.4.3 *Evaluation of the long-term LG-SLAM performance*

In order to test Algorithm 4, which allows long term operation of LG-SLAM, the KITTI dataset was used again, particularly sequences KITTI00 and KITTI05. These sequences were chosen because they are among the longest ones and they have significant trajectory parts

Table 5.8: Comparison of g^{2}o and LG-ESDSF on the EuRoC dataset.

	e_{trans} [m]	e_{rot} [deg]	e_F
	LG-ESDSF / g^{2}o		
MH_01_E	**0.159** / 0.213	2.103 / **1.871**	**0.036** / 0.048
MH_02_E	**0.106** / 0.111	**1.054** / 1.282	0.050 / **0.048**
MH_03_M	**0.125** / 0.141	**2.101** / 2.449	**0.039** / 0.048
MH_04_D	0.292 / **0.280**	0.949 / **0.895**	**0.075** / 0.094
MH_05_D	**0.156** / 0.164	**0.980** / 1.258	**0.060** / 0.086
V1_01_E	**0.247** / 0.251	**2.321** / 2.423	0.048 / **0.047**
V1_02_M	0.093 / **0.088**	0.876 / **0.831**	0.048 / **0.044**
V1_03_D	**0.121** / 0.126	**1.509** / 1.566	**0.047** / 0.052
V2_01_E	**0.113** / 0.143	**1.897** / 2.137	**0.07** / 0.096
V2_02_M	**0.076** / 0.079	**1.537** / 1.893	0.082 / 0.082

Table 5.9: Minimum, maximum and mean computation times of the LG-ESDSF update step and g^{2}o optimization on the EuRoC dataset.

	t_{min} [ms]	t_{max} [ms]	t_{mean} [ms]
	LG-ESDSF / g^{2}o		
MH_01_E	0.24 / 2.46	3.22 / 104.16	2.10 / 59.77
MH_02_E	0.20 / 0.63	3.71 / 104.92	2.24 / 53.52
MH_03_M	0.26 / 1.00	7.36 / 184.89	3.71 / 76.49
MH_04_D	0.22 / 1.09	4.60 / 119.76	2.85 / 65.92
MH_05_D	0.38 / 4.03	4.93 / 139.78	2.98 / 72.96
V1_01_E	0.36 / 1.77	3.56 / 67.59	2.17 / 30.06
V1_02_M	0.20 / 1.67	3.83 / 78.18	2.17 / 30.33
V1_03_D	0.40 / 2.86	3.55 / 117.23	2.42 / 52.00
V2_01_E	0.48 / 6.19	2.46 / 20.43	1.19 / 13.53
V2_02_M	0.22 / 1.01	5.92 / 39.98	2.21 / 17.32

that overlap and a loop closing near the beginning. For the testing purposes two consecutive runs of each trajectory were simulated and SOFT was used for odometry. Although this does not completely simulate a real-world experiment, since images of the second run would not be identical, they would be similar enough. Since the goal in this scenario was not to test the accuracy of the front-end, it is asserted that this approach is adequate to test the long-term LG-SLAM approach. Another advantage of such a simulation is that, since images are identical, trajectory augmentation with almost every new state in the second run results in a loop closing, which is precisely the scenario that needs to be tested. The benefit is also that accurate ground-truth for both experiments is provided.

To test the behaviour of the long term LG-SLAM algorithm, first the simulation on both tracks with two runs was completed without using the Algorithm 4 in order to get a reference. Afterwards, the topological distance was used to reject unnecessary loop closings and the marginalization of all states that were selected as close enough between two consecutive loop closings was performed, but without including the states participating in the loop closing.

Table 5.10: Long-term performance evaluation on KITTI00

	e_F [m]	n	n_u	t_u [s]	n_{new}
NO_MARG	1.17	3865	2215	130.62	1949
$^{T}d_{min} = 80m$					
NO_LOOP_MARG	1.21	1758	106	2.77	109
ALL	1.27	1645	121	2.86	12
$^{T}d_{min} = 6m$					
NO_LOOP_MARG	1.18	2772	1140	47.42	986
ALL	1.19	1646	1183	27.81	18

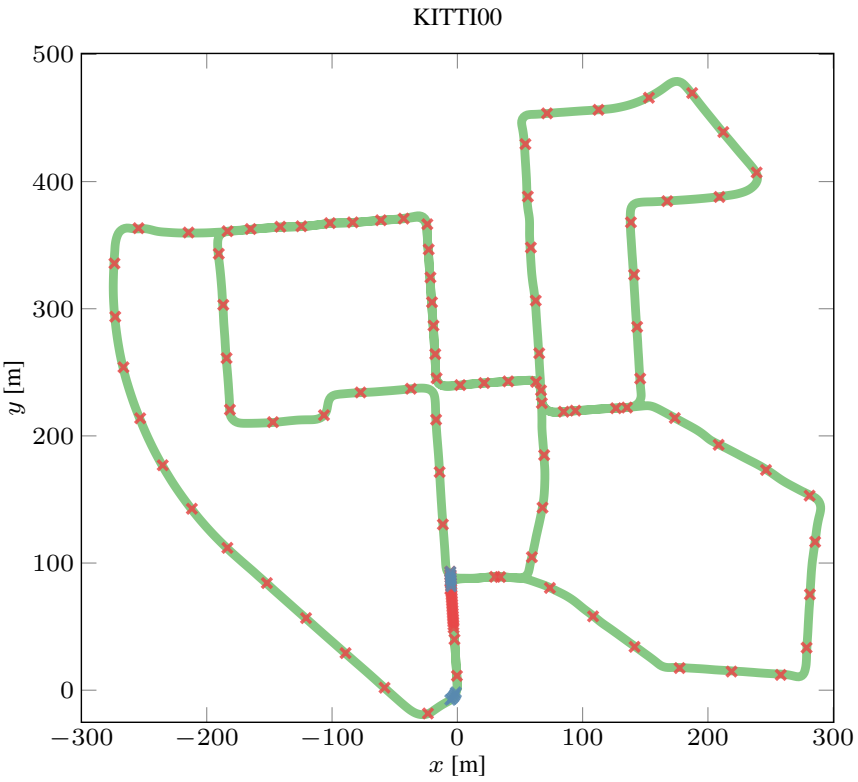

Figure 5.5: New states added in the second run of KITTI00 with $^{T}d_{min} = 80$ m. Green line represents trajectory in the first run, red × represent states added in the second run for the NO_LOOP_MARG case, and blue × represent states added for the ALL case. A significant reduction of added redundant states for the ALL case can be noticed.

This means that no sparsification was required. Finally, testing of all the components was performed by also including the state in which the loop closing was detected. The accuracy of final tracks was evaluated using e_F, the number of states n in the trajectory at the end was counted, total number of updates performed n_u was recorded, the total time required for all trajectory updates t_u was measured, and states added in the second run n_{new} were counted.

Table 5.10 shows the results for KITTI00. Label NO_MARG stands for results acquired without using the Algorithm 4. Label NO_LOOP_MARG stands for results when the state in which loop closing occurred was not marginalized and label ALL stands for results when all the steps of the algorithm were used. The algorithm performance was tested

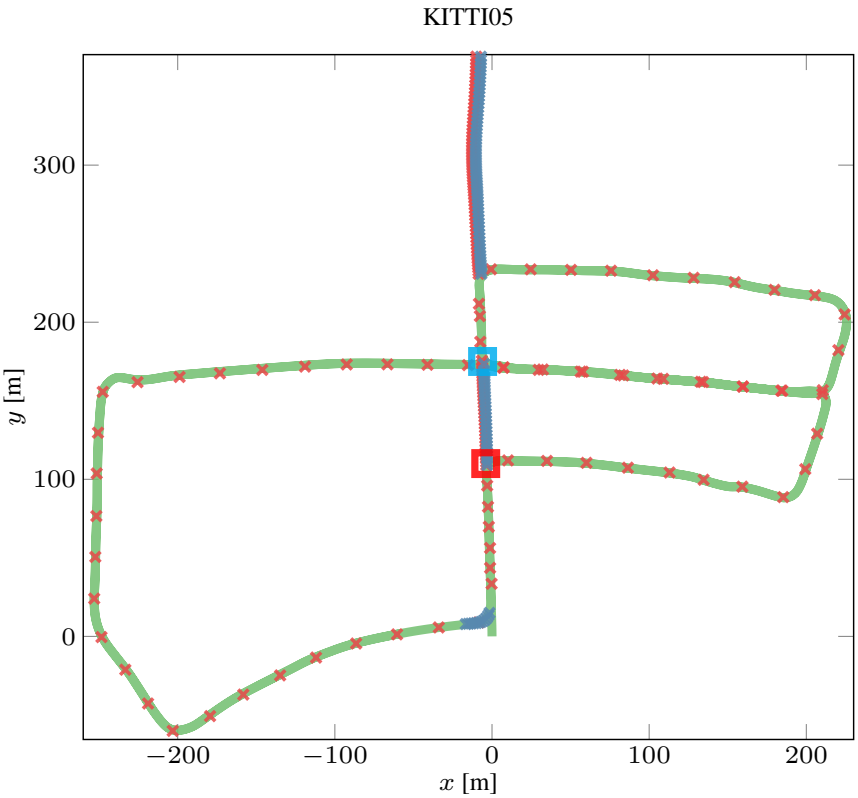

Figure 5.6: New states added in the second run of KITTI05 with ${}^{T}d_{min} = 50m$. Green line represents trajectory in the first run, red × represent states added in the second run for the NO_LOOP_MARG case, and blue × represent states added for the ALL case. Again, a significant reduction of added redundant states for the ALL case can be noticed. Red rectangle marks the area where the vehicle went right and missed the loop closing in the area marked with cyan rectangle during the first run, while in the second run it continued straight at the red rectangle and closed the loop at the cyan rectangle.

for two drastically different values of the minimal topological distance ${}^{T}d_{\min}$, one was set to 80 m and the other to 6 m. It can also be seen that when the algorithm was not used, error e_F for both runs seen in Table 5.10 remained the same as the error of the single run (confer e_F in Table 5.4). As expected, when rejecting updates and marginalizing blocks the error increases, while in the case of using sparsification, the error increase is even further noticeable. However, for all the tests the error increase remained small. When using the smaller ${}^{T}d_{\min}$, changes in the error were almost insignificant, since, in that case, fewer updates were rejected. This can be seen from the parameter n_u, which also indicates the correct behaviour of the loop closing rejection part. From Fig. 5.5 it can be seen that, when states in which update occurred were not marginalized, they were periodically added to the trajectory after each accepted update (red crosses in the figure). However, updates were performed only after ${}^{T}d_{\min}$ was exceeded. When states in which loop closing occurred were marginalized, only few states were added in the second run (blue crosses in the figure). This can also be confirmed by consulting the parameter n_{new}.

The most interesting parameter is probably t_u. As can be seen, when ${}^{T}d_{\min}$ is large, time required to complete all updates is practically the same when sparsification is enabled or disabled. This is because very few updates were performed and the effect of more dense

Table 5.11: Long-term performance evaluation on KITTI05

	e_F [m]	n	n_u	t_u [s]	n_{new}
NO_MARG	1.04	2150	1163	35.7	1149
$^T d_{min} = 50m$					
NO_LOOP_MARG	1.29	1072	88	1.31	172
ALL	1.29	997	96	1.30	107
$^T d_{min} = 5m$					
NO_LOOP_MARG	1.0	1633	650	14.58	669
ALL	1.01	993	669	8.31	105

information matrix was nullified by the time required to do sparsification. Much better effect of sparsification can be seen when $^T d_{min}$ is small. In this case it can be seen that t_u is much smaller when using sparsification in comparison to the case when it is not used. This confirms the expected effect of marginalizing the state in which loop closing was detected.

Results from the test conducted using KITTI05 are displayed in Table 5.11. Most of the conclusions drawn for the test using KITTI00 can also be made here. The difference is that in this case sparsification reduces accuracy even less. It can also be seen that in this case e_F is even slightly smaller when using marginalization with small $^T d_{\text{min}}$ than without performing any marginalization, which is due to rejecting some loop closings that had errors in their estimates. The e_F of two runs seen in Table 5.10 in this case is larger than in the case of only one run (confer e_F in Table 5.4). The reason for this is, because unlike in KITTI00, in KITTI05 two runs are different and one important loop closing is not detected until the end of the second run. This can be seen in Fig. 5.6. When robot arrived at the position marked with red rectangle in the first run, it turned right, while in the second run it went straight and closed the loop in the area marked with the cyan rectangle. From this figure it can also be seen that when robot entered a previously unexplored area (straight segments densely populated with 'x') the state vector was correctly augmented with new states.

From the results it can be said that the proposed algorithm works correctly and drastically reduces the number of unnecessary states in the trajectory.

5.5 SUMMARY

In this chapter a novel filtering SLAM back-end solution based on the ESDSF derived on Lie groups, dubbed LG-ESDSF was presented. The developed filter does not only retain all the good characteristics of the classic ESDSF implementation, the main being the exact sparsity of the information matrix, but also respects the state space geometry by negotiating uncertainties and employing filtering equations on Lie groups. In addition, a method which allows LG-ESDSF back-end to work over long-term, while the robot is continuously moving through already explored environments, was developed. In order to test the LG-ESDSF back-end, it was coupled with two different front-ends: one based on the stereo camera and one based on the 3D laser range sensor. Complete SLAM solution was named LG-SLAM. For testing it two publicly available datasets were used. First dataset was the KITTI dataset which was recorded with a commercially available vehicle equipped with a 3D LIDAR and

two pairs of stereo cameras. The second dataset was the EuRoC dataset recorded with an UAV equipped with a stereo camera. Results of LG-SLAM were compared with g^{2}o when coupled with the same front-ends and under the same conditions. Results have showed that the LG-SLAM achieves similar accuracy as g^{2}o, with significantly faster computation times of the update step in comparison to the g^{2}o optimization. LG-SLAM has also been evaluated using the KITTI online evaluation protocol, achieving the second best result among all the stereo odometry solutions and the best results among all the tested visual SLAM algorithms. Finally, LG-SLAM long-term performance was checked using the KITTI dataset. The results validated the effectiveness of LG-SLAM in such conditions. In the end it can be asserted that although graph-optimization solutions still offer several advantages over the filtering solutions, like easier applicability to a wider range of problems, judging from the newly proposed method, it can be said that filtering solutions deserve to be once again in the focus of the SLAM research.

6

Cooperative SLAM

In this chapter cooperative planar SLAM solution implemented on Lie groups, dubbed CLG-SLAM, published in [173] is presented. It allows simultaneous execution of SLAM tasks over multiple heterogeneous agents. The back-end of CLG-SLAM is based on the LG-ESDSF presented in Chapter 5. The front-end is the same as presented in the Chapter 4. It segments point clouds obtained from 3D LIDAR to represent map as set of planar surfaces in the same way as explained in Sec. 4.3.3 and loop closing is detected in the same way as described in Sec. 4.3.4. The main difference is that now global map is not being built on the agent but on the standalone cloud server from the planar segments sent by each agent.

CLG-SLAM is the most similar to the cooperative SLAM solution presented in [174], dubbed C^2TAM. C^2TAM is a mix between centralized and decentralized SLAM system in which each agent performs its own localization using computationally light visual odometry, while computational costly steps, including optimization and map building, are executed on an external server. The main idea behind CLG-SLAM is also to divide the computational load; however, there are three key differences between CLG-SLAM and C^2TAM. First, instead of graph-optimization back-end, CLG-SLAM uses developed LG-ESDSF SLAM back-end for each agent. Second, each agent performs its own trajectory estimation and optimization, while also maintaining local map which allows it to operate completely independently in the case of other agent and/or server failure. Third, map of the environment consists of planar surfaces which drastically reduces the memory and computation requirements when exchanging and using maps.

The rest of the chapter is organized as follows. First, brief introduction into the cooperative SLAM is given. Then, CLG-SLAM is explained in details. Finally, experimental results are presented which prove the effectiveness of CLG-SLAM.

6.1 INTRODUCTION

In general, SLAM main purpose is to continuously work in the background and provide estimated map and location to the higher level algorithms, e.g., exploration, navigation etc. In many cases these higher level algorithms would benefit from multiple agent cooperation in order to complete their tasks faster. For example, when considering the exploration task, which requires building a complete map of the area in the shortest possible time, several agents (e.g. ground and aerial robots) would complete the task much faster than a single agent. The easiest way to do this would be to explore a part of the environment with each

agent using one of the available single agent SLAM algorithms, and then merge local maps in a single global map. However, this solution has several drawbacks, and the most important one is the inability of one agent to use information from the other agents to increase its own mapping accuracy. SLAM algorithms that overcome this problem are called cooperative SLAM algorithms.

Cooperative SLAM has been developed in parallel with single robot SLAM algorithms and thus cooperative SLAM algorithms can be divided based on the same principles that SLAM front-ends and back-ends for single robot SLAM are divided. In addition, cooperative SLAM systems can be divided based on the way they handle specific requirements that arise in the multi robot environments. One of the main differences between cooperative SLAM solutions is whether they are centralized or decentralized. In the centralized SLAM system most of the computation is done on a predefined robot or even in the external source (i.e. computer server) while robots have only basic processing requirements needed to acquire and send measurements and receive commands. Examples of such system can be found in [175, 176]. Main advantage of centralized approach is cheaper robots since all computation is done in one place and easier data handling. Main disadvantages are lack of robustness since if the main machine goes offline none of the robots can continue to work and requirement for constant communication with the main machine.

Decentralized cooperative SLAM solutions like [177] use computation power of all robots to solve certain tasks. Robots within the system are divided into groups of which each is responsible for computing a certain type of task. Decentralized cooperative SLAM solutions can work even if one or more of the robots develops a malfunction. However, each robot must have more computational power and data handling is more complex since every robot must keep a track on what others are doing and what task is currently being executed.

Front-end part of the cooperative SLAM required to process the data from the sensors is similar to that of the SLAM system used for the single robot. However, SLAM front-end in cooperative SLAM has to be able to communicate with other robots and machines and exchange sensor data and other information. Depending on the bandwidth of the connection entire raw data can be sent if the bandwidth is high. This solution is also more suitable for centralized SLAM systems, since in that case robots do not need to process data, but simply relay it over the network. In case the bandwidth is limited and/or the agents have enough power to the process measurement data, before sending it, the data is filtered, its size reduced and only the useful part is then in some form transmitted over the network. Examples of such approach can be found in [178, 179].

Regardless of the SLAM front-end implementation, information sharing and distribution and other specifics of the cooperative SLAM front-ends, the main difference between every cooperative SLAM algorithm is the implementation of the SLAM back-end. As is the case for single robot SLAM, SLAM back-ends for cooperative SLAM can be divided into two main groups depending on the optimization techniques. One group contains SLAM systems that use filtering based back-ends, while the other group contains SLAM back-ends that rely upon graph optimization techniques like g^{2}o [55] and iSAM2 [56].

Of all approaches, the easiest transformation from a single to multi robot SLAM is done when using EKF based back-end. Implementation of cooperative EKF-SLAM [180, 181] is straightforward since in EKF-SLAM for one robot the state space consists of robot pose

and pose of the map landmarks. All landmarks and robot poses are connected through the covariance matrix. In the case of cooperative EKF-SLAM solutions, robot poses and landmarks extracted by all robots are simply added into the same state-space. However, cooperative EKF-SLAM suffers from the same problems as single robot EKF-SLAM. The requirement for linearization of both motion and measurement models and slow computation speed with high numbers of landmarks in the state space. Computational complexity is even more of a problem in cooperative solution, since number of landmarks increases more rapidly.

More advanced solutions to cooperative SLAM based on filtering approach use PF and EIF. Main advantage of PF multiple robot SLAM [175, 182] is its requirement to linearize only the measurement model while its real time computation is maintained using Rao-Blackwellization [41]. As explained before, using EIF for multiple robot SLAM [126] allowed simpler decentralization of the information acquired by each robot, but the main problem of EIF remained, slow computation speed when number of landmarks increases. This problem was partially solved in [1] with the introduction of SEIF, but sparsification also introduced inaccuracies.

However, as is the case with SLAM solutions for single agent today, state of the art cooperative SLAM algorithms use graph optimization back-ends [183, 184] and generally employ global graph consisting of subgraphs built by each agent. If relative pose between agents is not known in advance, subgraphs are not connected until the agents meet, whereas if the relative pose is known, the subgraphs are connected from the beginning. A more thorough analysis of all aspects and differences between various multiple agent SLAM algorithms can be found in [185].

6.2 Cooperative 3D planar SLAM based on LG-ESDSF

CLG-SLAM was designed with the four main goals in mind:

1. To perform computationally costly operations on the standalone server, while only computationally less demanding tasks are performed by the agents.
2. To be able to quickly exchange maps between agents and server.
3. To be able to continue to operate if cloud server or connection with it fails.
4. To use the experience of some agent for improving the accuracy of another.

Achieving these goals makes CLG-SLAM a fast and robust cooperative SLAM algorithm capable of running on variable number of heterogeneous agents that require only periodic wireless communication with the standalone server. The main assumption of CLG-SLAM is that relative poses between all agents are known at the beginning. Although this may seem to be a limiting factor, it is acceptable respecting its intended use for faster exploration. The reasoning is that the algorithm does not require to know absolute starting location, but only relative poses between one reference agent and all other agents. Also, it is only required for agents to be on the line of sight at the very beginning, hence scans taken from the 3D LIDAR can be initially matched using the LMR algorithm. Because of this, although condition

can be limiting for some tasks, in the case of multiple agent exploration application, this requirement is easily reachable.

6.2.1 *The overall concept of the proposed cooperative SLAM system*

The overview of the CLG-SLAM system is shown in Fig. 6.1. Each agent in the system builds its own trajectory using LG-ESDSF back-end. It also performs loop closings and segmentation of measurements into the planar surface segments. However, global map building algorithm explained in Sec. 4.3.3 is performed on a standalone Cloud Server (CS). This is possible because of the LG-ESDSF property to estimate trajectory independently from the global map. Once the new local planar map M_i is built, it is sent to the CS alongside with the current trajectory of the respective agent. After CS receives M_i, it incorporates its planar segments into the global map using current trajectory information. Currently updated global map is available at any time instant on the CS and any agent or higher level algorithm running alongside CS (e.g. exploration, navigation) can retrieve it. The incorporation of local planar maps from different agents into the global map is possible based on their trajectories, since their relative poses are known and every trajectory is built within the same global frame.

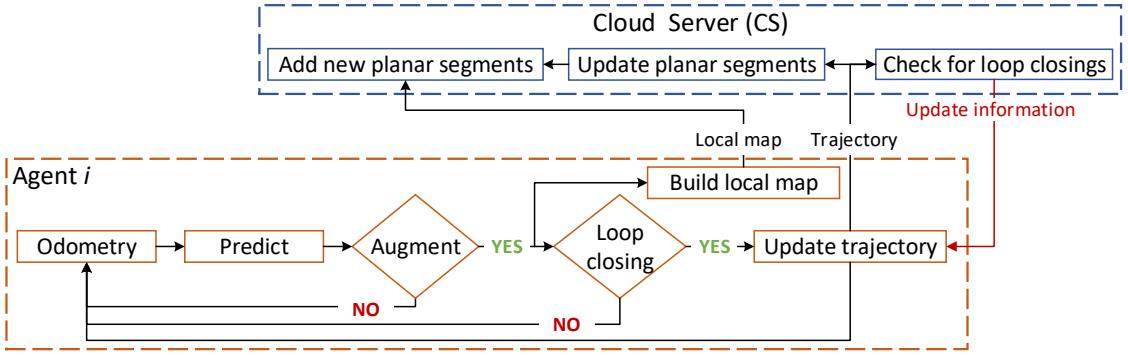

Figure 6.1: Overview of the proposed cooperative SLAM solution.

Besides allowing construction of the global map in the same frame from the start, the most important advantage from knowing relative poses between agents in the beginning is the ability to improve their entire trajectories and global map every time they meet later during the environment exploration. When the initial relative pose of the agents is unknown, their trajectories will be also corrected every time they meet, but the problem is that the trajectories and global map corrections are almost entirely limited to the area of the environment that was mapped by both agents before they have met. Although, since all robot states and map landmarks are connected they will all be corrected in the global frame, i.e. also outside the overlapping area, their relative poses can only be marginally corrected until further rendezvouses occur in different locations. Since in the case of multi-agent exploration the moments when the agent's trajectories intersect cannot be predicted but are supposed to be rare, knowing initial relative poses is of great importance for maximally exploiting information from every meeting.

The presented architecture of CLG-SLAM inherently achieves the first set goal, which is the separation of computationally more complex SLAM tasks from the less demanding ones.

As explained in Sec. 5.2.5, all steps of the LG-ESDSF have very low computational cost and as proven in the experiments presented in Sec. 4.6 local map building and pose constraint estimation is also computationally inexpensive. This means that the only computationally costly operation is the global map building which is accomplished by CS.

One could argue that building local planar maps on-board the agent is unnecessary and that it should also be performed by the CS. However, having local maps available on-board an agent helps achieve the second and the third set goal. Since only segmented local planar maps need to be transferred to the CS instead of the entire point clouds, local map transfer is simple and quick which fulfils the second set goal. The third goal, which requires the CLG-SLAM to be robust, is achieved since in the case CS or connection with it fails, local map can be used in combination with the trajectory for navigation to the predefined safe location. Moreover, since the entire system is event triggered, it is also robust to the time synchronization errors. If the new local map arrives while the CS is busy with incorporating previously received one, it will simply be incorporated as soon as CS becomes free. The final goal left to achieve is the ability of CLG-SLAM to use one agent information to increase the accuracy of another.

6.2.2 *Updating one agent with the information from another agent*

In order to use one agent's information to correct the trajectory of another, loop closing between their trajectories has to be detected. This is done by the CS for two reasons:

1. Only CS has the trajectory of every agent.
2. In order to find the loop closing between different agents, measurements from all the agents need to be checked which can be a costly computation.

The CS searches for possible loop closings in the similar way as a single agent does, using the algorithm for loop closing detection explained in Sec. 4.3.4. The example shown in Fig. 6.2 depicts trajectories from two agents: a (blue) and b (orange). Agents started to map the environment from poses X_0^a and X_0^b and at some point they arrived at poses X_i^a and X_j^b, respectively, after which the CS detected the loop closing. Once the pair (X_i^a, X_j^b) is identified for loop closing CS uses LMR to estimate relative pose $T_{i,a}^{j,b}$ between them. The CS then calculates uncertainties of states X_i^a and X_j^b. The trajectory which contains the state with the higher uncertainty is selected for correction based on the other trajectory. Let's say that the state X_j^b has higher uncertainty. In order to correct trajectory of the b-th agent, since every agent possesses only its own trajectory, update must be performed based on the relative pose between the state X_j^b and some other state X_k^b, $k \neq j$ from the same trajectory. Any state can be chosen from b-th agent's trajectory as X_k^b, but the goal is to choose the state with the smallest uncertainty because then the loop closing will have the largest impact on accuracy. The state that best satisfies both conditions is state X_1^b since it is the first state in the trajectory. Now in order to perform the update of b-th agent's trajectory, $T_{j,b}^{0,b}$ has to be found using new measurement $T_{i,a}^{j,b}$ and a-th agent's trajectory. Since all trajectories are within the same global frame this can be done as:

$$T_{j,b}^{0,b} = (X_0^b)^{-1}(X_i^a T_{i,a}^{j,b}). \tag{6.1}$$

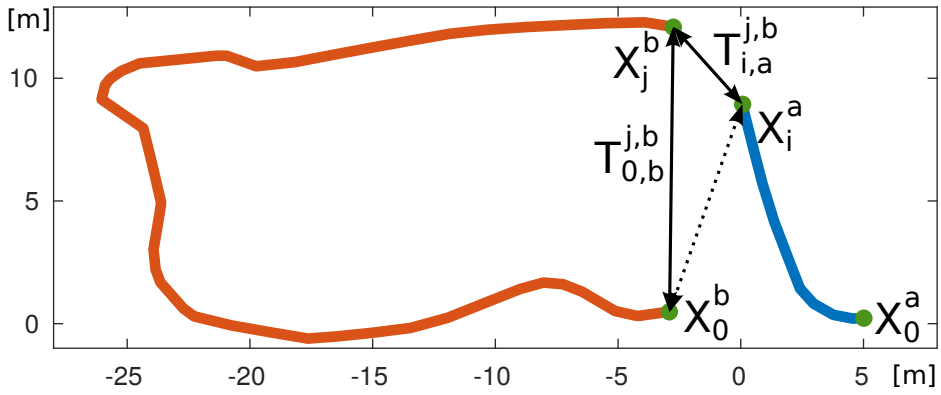

Figure 6.2: Example of b-th agent's trajectory (orange) update using information from a-th agent's trajectory (blue).

Finally the update information is sent to the agent immediately after calculation and the trajectory update is performed right after the new state is augmented into the agent's trajectory. The process is repeated for every loop closing detected by the CS. Since the loop is always closed between the first trajectory state X_1 and another state X_i, it will always impact the accuracy of all states between X_j, $0 < j \leq i$. If the initial relative pose between agents was not known upfront, this would not be possible since in equation (6.1) X_0^b and X_i^a would not reside in the same coordinate frame.

This concludes the derivation of CLG-SLAM. The next section presents the experimental results which prove its effectivness.

6.3 EXPERIMENTAL RESULTS

CLG-SLAM was tested on two datasets. One was the same indoor dataset recorded for testing the single agent planar SLAM explained in the Chapter 4 while the other was the KITTI dataset. In order to simulate cooperative behaviour the datasets were divided into subsets. Then, identical SLAM systems were ran as separate threads simulating individual agents. Each agent received data from one of the subsets. In both experiments, the CS was also run on the same computer as a separate thread and independently received data from each agent. All algorithms were implemented using C++ under ROS and run on portable computer Lenovo P50 with 8GB RAM and Intel Core i7@2.6 Ghz. Since CLG-SLAM is event based, no generality was lost by simulating all agents and the CS on the same computer as independent threads.

To test the ability of one agent to improve its accuracy using information from another agent, the experiments on both datasets were first performed without the CS sending loop closing information to the agents. This way every agent closed only loops detected by itself. Then the same experiment was simulated again, but this time the CS sent loop closing information to the agents.

6.3.1 *Test results on the indoor dataset*

The indoor dataset was separated into two subsets, which means cooperation was done between two agents. Since ground truth trajectory for the indoor dataset is not available,

S_{min} [MB]	S_{max} [MB]	S_{mean} [MB]
Point clouds		
1.17	2.18	2.08
Local maps		
0.03	0.20	0.11
Local maps size [MB] / Map size [MB]		
30.2 / 2.6		

Table 6.1: Sizes of point clouds, resulting local maps and global map.

results of experiments conducted were evaluated the same way as in Chapter 4. The 2D ground plan was generated by slicing the global map built by CS at certain height from the ground and was then overlayed over the ground-truth ground plan. The results, alongside

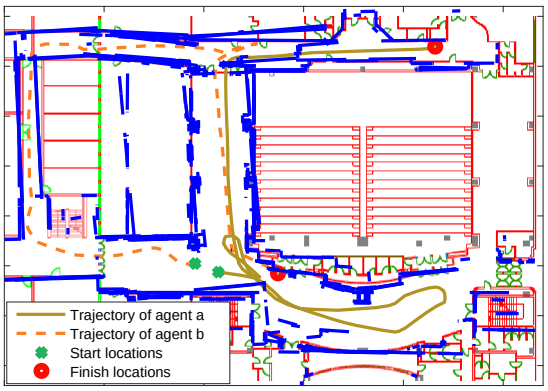

Figure 6.3: Ground plan in the case when CS did not send any information.

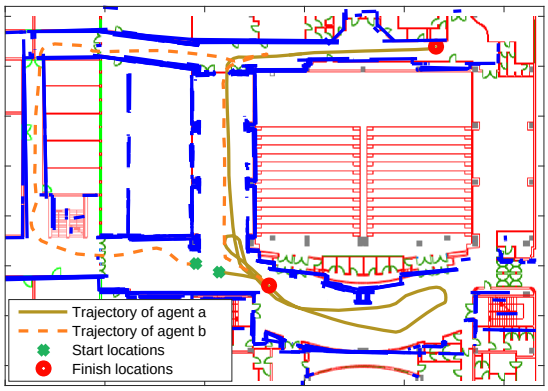

Figure 6.4: Ground plan in the case when CS sent loop closing information.

with agents' trajectories, are shown in Fig. 6.3 and in Fig. 6.4, for the case when CS did not sent anything to the agents, and for the case when CS sent loop closing information to the agents, respectively. As can be seen from the resulting figures, accuracy of the ground plan is much better in the case when CS sent information to the agents. Many duplicate planes that represent the same wall have been merged into single plane in the global map and walls have been correctly aligned.

Table 6.1 shows minimum, maximum and mean sizes of point clouds and the resulting local maps recorded by both agents. The segmentation of point clouds drastically reduces their size, thus allowing lower memory requirements for agents and faster transfer times over the network to the CS. Final row in Table 6.1 shows comparison between the cumulative size of all local maps and size of the final global map. It can be noticed that the final map size is much smaller than the size of all local maps which allows faster transfer of the global map and also faster computation times for the tasks which use global map, e.g. exploration, navigation, etc. Final global map built when the CS sent loop closing information is shown in 3D in Fig. 6.5.

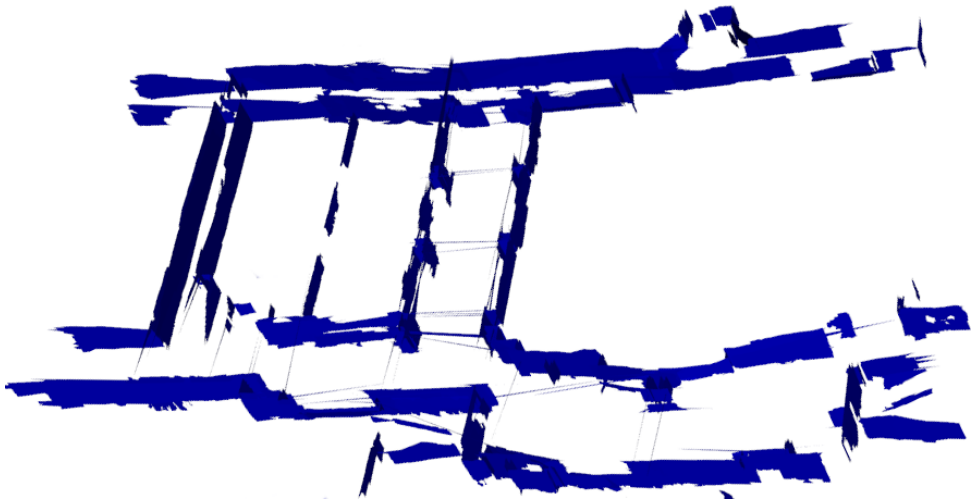

Figure 6.5: Global map built by CS when loop closing information was sent to the agents.

6.3.2 *Test results on the KITTI dataset*

Tests on the KITTI dataset were performed on the first track (KITTI00) which was divided into three subsets, each representing data for one of the three agents. Because the KITTI dataset does have the ground truth trajectory, it was used for comparing the accuracy of the final trajectories obtained by all three agents when CS did not send update information to the agents and when it did. For comparison the same metric based on (4.64) explained in Sec. 5.4 was used. The odometry on the KITTI dataset used for the prediction step in LG-ESDSF was estimated using the 3D-NDT algorithm. Trajectories of all three agents obtained when CS did not send update information are shown in Fig. 6.6, while Fig. 6.7 shows the trajectories when CS sent the update information. The results of comparison between complete trajectories obtained by all three agents and the ground truth trajectory for both cases are given in table 6.2. As can be seen, both rotational and translation errors decreased when CS sent the updates to the agents. The effect of these updates can best be seen by comparing trajectory of the first agent on Figs. 6.6 and 6.7. We can see that by closing the loop with the trajectory of the third agent, much of the accumulated rotational error in the trajectory of the first agent was corrected.

Errors when the updates were not sent		Error when the updates were sent	
e_{trans} [m]	e_{rot} [deg]	e_{trans} [m]	e_{rot} [deg]
9.15	3.76	8.76	3.2

Table 6.2: Comparison between the complete trajectory accuracy when CS did not send the updates and when the updates were sent.

Table 6.3 shows comparison between the point cloud sizes and the sizes of the resulting local maps. The same conclusions made for the indoor dataset can be made here. However, the sizes are higher than in the indoor experiment due to the difference in the LIDAR used to obtain the dataset. The indoor dataset was obtained with Velodyne HDL-32 which has 32 vertical beams, while the KITTI dataset was collected with Velodyne HDL-64, which has 64 vertical beams. Figure 6.8 shows the complete 3D model built by all three agents when

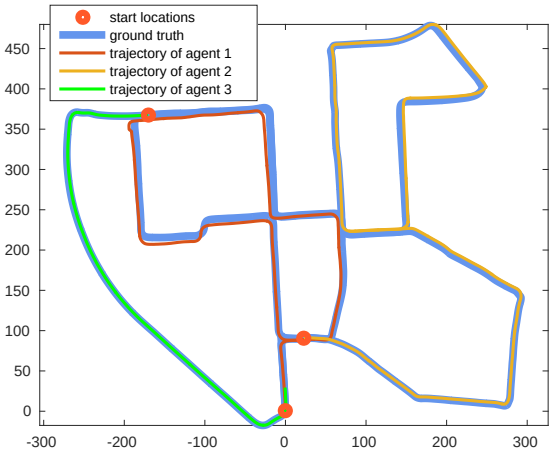

Figure 6.6: Trajectories of all three agents when CS did not send updates.

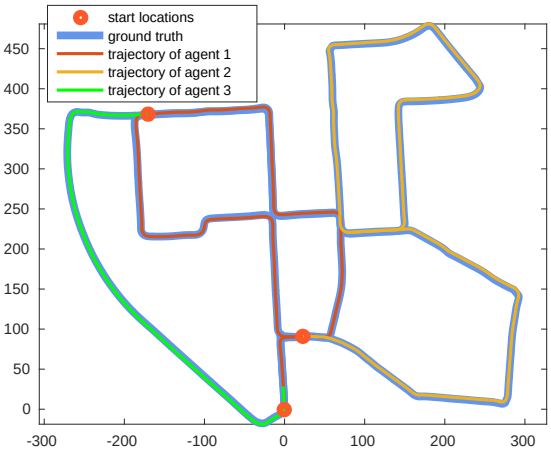

Figure 6.7: Trajectories of all three agents when CS sent updates.

S_{min} [MB]	S_{max} [MB]	S_{mean} [MB]
Point clouds		
2.83	3.38	3.55
Local maps		
0.36	0.78	0.1.34
Total point clouds [GB] / Total local maps [GB]		
15.0 / 1.28		

Table 6.3: Sizes of point clouds and the resulting local maps.

they received updates from the CS.

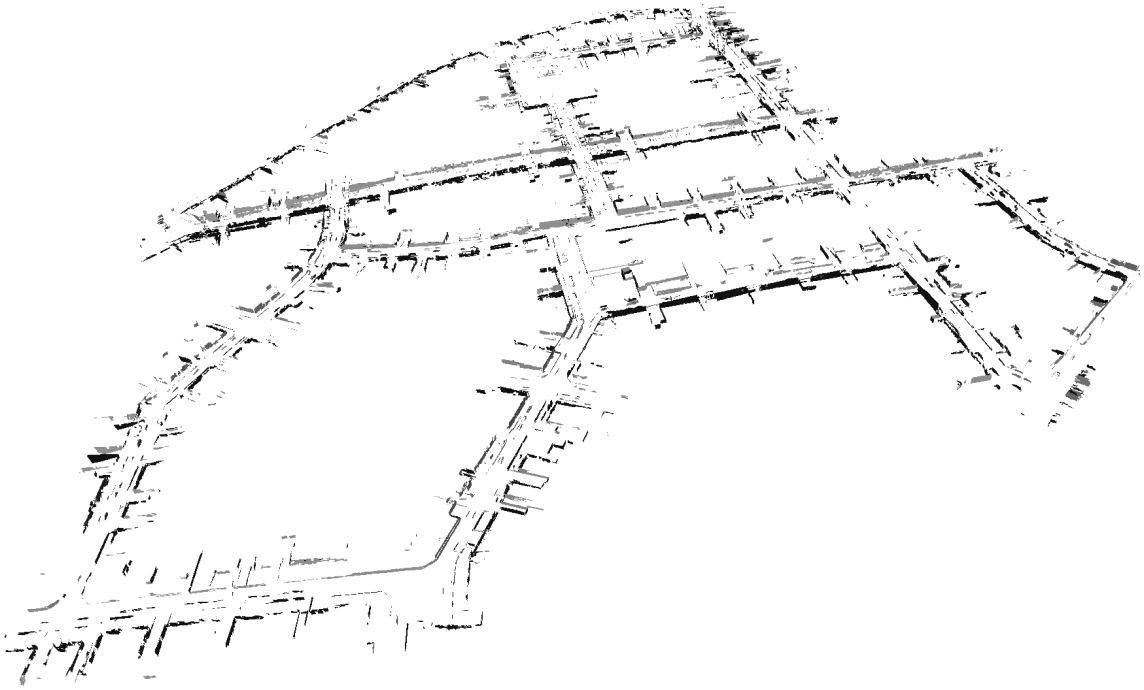

Figure 6.8: Global map built by the CS when loop closing information was sent to the agents.

6.4 SUMMARY

In this chapter the cooperative SLAM solution dubbed CLG-SLAM based on previously developed LG-ESDSF back-end and planar front-end was presented. CLG-SLAM was developed having four key goals in mind: (i) Separation of computationally complex SLAM operations from the less demanding ones, (ii) ability to increase mapping accuracy of one agent by using information from other agents, (iii) using compact map representation which allows easy map exchange between agents and server and (iv) robustness of the entire system by allowing each agent to function independently in case the CS or communication with it fails.

The first goal was achieved using LG-ESDSF back-end, which enables separation of trajectory estimation from the global map building. This separation enabled transfer of costly global map building operation to the CS, while thanks to the sparsity of the information form and fast planar segmentation of point clouds, trajectory estimation and local map building were able to run on computationally less capable agents. To enable one agent to increase accuracy of another, an algorithm was developed which runs on the CS and analyses measurements from all agents in order to find loop closing. After a suitable loop closing is detected, it determines which trajectory will be updated and sends the update information to the respective agent. Compact map representation was achieved by utilizing developed planar SLAM front-end which segments 3D point clouds into planar surfaces, thus drastically reducing their size while maintaining high level of details. Finally, robustness was achieved by allowing each agent to maintain its trajectory and local map which gives it the ability to know where it is and what is around it even if CS or connection with it fails. The effectiveness of CLG-SLAM was demonstrated using two real world datasets. One is the indoor dataset recorded in our faculty building and the other is the KITTI dataset.

7

Conclusion and outlook

The development of mobile robotics was first focused on producing mobile robots that are complementary with human abilities. However, within time, the need has arose for robots that can not only work alongside humans but also completely replace them in wide variety of tasks, emphasis being on tasks that can endanger humans. The increased development and popularity of mobile robots in the last decade has created an illusion that today's mobile robots already have such potential. However, accidents like the one in Fukushima nuclear power plant in Japan has demonstrated how far behind us, in certain tasks, robots really are. There was not a single robot in the world that was able to enter the damaged plant and do the repairs instead of humans. One of the main challenges in designing the mobile robot that can successfully replace humans in such situations is in developing algorithms than can accurately map the environment and estimate accurate robot's location in that map. These algorithms are also in the main focus of the present thesis.

There are three main concepts to solving the localization problem, each giving more autonomy to the mobile robot. The most simple and least autonomous method is the one which uses artificially placed beacons in the environment to localize the robot. Next and more autonomous method is when the map of the environment is built before the robot is sent to execute tasks in that environment. Although very accurate, this method requires a-priori knowledge of the environment and as such provides a limiting factor to the autonomy. The only method that offers full autonomy of the mobile robot is the one which allows the robot to build the map of the environment and localize the robot within it at the map at the same time. In mobile robotics, methods that solve this problem are known under common acronym SLAM (Simultaneous Localization and Mapping).

SLAM is formulated as a probabilistic problem which estimates probability distribution of map and robot pose conditioned upon all acquired measurements, current control inputs and previous robot pose. Two basic components of every SLAM solution are: its back-end, which deals with robot pose and map optimization, based on constraints produced by the second main component, the SLAM front-end. SLAM front-end deals with sensor measurements and consists of three main segments: (i) odometry segment used to estimate current robot pose based on consecutive measurements and previous pose, (ii) loop closing detection used for determining when the robot has arrived at previously visited locations, and (iii) pose constraints calculation used to estimate relative pose between measurements in which loop closing was detected. These constraints are then used in the back-end for pose and map optimization and are essential for maintaining accuracy in the SLAM system.

SLAM back-end approaches can be divided into two main groups. The first one uses filtering approach based on prediction and update steps. The second group comprises approaches which solve SLAM problem by defining it as a non-linear least squares problem and are referred to as the graph-based SLAM back-ends. There are four main filtering approaches to the SLAM back-end and they primarily differ based on the filter used. They are based on: (i) the Extended Kalman Filter (EKF), (ii) the Particle Filter (PF), (iii) the Extended Information Filter (EIF), and (iv) the Sparse Extended Information Filter (SEIF). Graph based SLAM solutions mainly differ by the terms included into the minimization criterion and by the optimizations used to make them more computationally efficient. In this thesis the main focus is on the filtering based approaches.

The first scientific contribution of the thesis is the SLAM solution based on Exactly Sparse Delayed State Filter (ESDSF) back-end and the planar based SLAM front-end. ESDSF is a special form of the EIF filter, which has the same state space as PF back-end. Instead of estimating only current pose of the robot and the poses of all map landmarks, it estimates robot's trajectory consisting of discrete past robot poses. As a result map landmarks become independent on each other and are conditioned only upon the discrete robot state at which they were extracted. This allows fast map estimation that can be done in parallel with the trajectory estimation. Moreover, by using this state space, the information matrix becomes exactly sparse and does not require sparsification step, like in SEIF, which makes ESDSF more accurate. This sparsity can then be exploited by a special sparse matrix solver to drastically decrease computation time.

Planar front-end is based on the planar segmentation algorithm which extracts planar segments from the 3D point clouds obtained by the 3D LIDAR. It performs this very fast because it projects the point clouds into three 2D image planes. Each point is defined by the pixel coordinates, and by the pixel value which is equal to the point distance from the LIDAR's coordinate frame. The Delaunay triangulation is then used to combine these pixel-points into planar segments. All segments extracted from the same point cloud are put into local maps which are afterwards used for the pose constraint calculation and the global map building. Pose constraint calculation is done using Local Map Registration (LMR) algorithm which matches segments from two local maps by checking their coplanarity and overlapping and then uses matched pairs for relative pose hypothesis generation.

To be more specific, the first scientific contribution is actually the global map building algorithm which builds global planar map from the planar segments in local maps. The easiest way to do that would be to simply transform all planar segments based on the SLAM trajectory into one coordinate frame. However, this would result in many duplicate segments which would occur when two local maps would consist of segments extracted from measurements taken from the same place. Moreover, planar surfaces like walls and floors would be represented by many different planar segments instead of one. The proposed algorithm solves this by combining all coplanar segments into a single global planar surface. Each global planar surface parameters are estimated based on the parameters of all local planar segments it consists of. Also, algorithm maintains connections between local maps and global surfaces and allows very fast updating of the global map when the trajectory optimization occurs.

The second scientific contribution of the thesis arises from the need of correct pose

representation in filtering based SLAM back-ends. Regardless of the SLAM version, there is always the need to estimate the robot's pose and poses of map landmarks which inherently in 3D reside on $\mathsf{SE}(3)$. Filtering solutions dominantly rely on the Euler angles or quaternions for representing those poses. Although sufficient, filtering with those representations within Euclidean frameworks does not represent a natural way of characterizing uncertainties and relations between the state vector elements. This problem was also encountered when using quaternions in ESDSF back-end because of the requirement on rotation quaternions to remain unit quaternions. Since state-of-the-art graph optimization back-ends rely more on using the insights of Lie groups and Lie algebras to represent pose, this was one of the main reasons why filtering approaches were generally not on par with the graph-based optimization based SLAM performance. Changes in this field began to happen only recently with the introduction of the EKF on Lie groups (LG-EKF) and the extended information filter on Lie groups (LG-EIF). The knowledge gained from the derivation of LG-EKF and LG-EIF was used to derive a novel implementation of ESDSF on Lie groups (LG-ESDSF). The LG-ESDSF allowed the ESDSF update and prediction equations to be computed directly on the Lie groups, which as a result had a positive impact on the overall accuracy. Derived LG-ESDSF is able to retain all the good characteristics of the classic ESDSF, main being the sparse information matrix, and achieve accuracy of state-of-the-art graph-optimization SLAM back-ends. Moreover, thanks to the usage of sparse matrix solvers, it is also considerably faster. This was proven by extensive testing on two publicly available datasets, one of them being the KITTI dataset which also has online evaluation protocol. LG-ESDSF was coupled with visual front-end based on stereo cameras and currently holds the best result of all the tested visual SLAM solutions.

The third scientific contribution of the thesis is also connected with LG-ESDSF and is based on the algorithm which allows LG-ESDSF back-end to maintain the ability to work online for a long period of time while the robot is moving trough the same environment. Once the robot builds the complete map of the environment, the SLAM algorithm can be turned off and some of the algorithms for localization based on the given map can be used. However, if we want the robot to continue exploring new areas at some point, or if we want to further increase the accuracy of the map and location, we need to reinitialize the entire SLAM which is complex and can never be as accurate as if the SLAM had continuously worked online. The problem is that if we leave SLAM online, the new states will be added continuously and consequently will result in inability to perform calculations on time for real-time operation. To solve this the new algorithm is introduced which removes states from the state space that hold very little new information. Also the algorithm preserves the sparsity of the reduced size information matrix thus allowing the SLAM to continue operating continuously over long periods.

The last (fourth) scientific contribution of the thesis is in allowing the developed SLAM solution based on the LG-ESDSF back-end and the described planar front-end to work cooperatively on multiple agents. This is especially useful when SLAM is coupled with higher level algorithms, like exploration algorithm, which greatly benefit from the multiple agents working towards the same goal. The easiest way to do this would be to explore a part of the environment with different agent, using one of the available single agent SLAM algorithms, and then merge local maps in a single global map. However, such solution

has several drawbacks, and the most important one is the inability of one agent to use information from other agents to increase its own mapping accuracy. The SLAM algorithms that can solve this problem are called cooperative SLAM algorithms. Cooperative SLAM algorithm presented in the present thesis uses state space formulation of the ESDSF to perform lower complexity operations on the agents, while the more complex operations are performed on a standalone server which communicates with the agents wirelessly. The agents perform trajectory optimization and local map building, while the server performs global map building. Local planar maps are easily sent to the server due to their small size alongside with the trajectory. The server uses trajectories of each agent to find the loop closings between them and calculate pose constraints. These constraints are then sent to each agent and thus the information of one agent is used to improve localization accuracy of other agents. Cooperative SLAM system built this way is also robust to the server or agent failure, since each agent has its own trajectory and local map which it can use to navigate safely. Moreover, developed cooperative SLAM is also immune to synchronization problems since it is event triggered.

As can be seen from many different algorithms and research fields covered within this thesis, the SLAM solution includes knowledge and theoretical advancements from large variety of different areas, probably more than any other algorithm in mobile robotics. As such, there is always a need for improvement in many different aspects.

Today, ever more powerful CPUs and GPUs allow highly complex algorithms to be processed online, and development in sensors technology has allowed modern robotic systems to perceive the environment better than ever. However, there is still the lack of robustness when combining all these algorithms, and more importantly there is still the lack of an algorithm that would allow perception of the environment similarly to the way as humans do. Advancements in these two areas will mark the development of future state-of-the-art SLAM algorithms.

When concerning the robustness in SLAM two key aspects are extremely important. The first one is the incorporation of measurements from many different sensors and perceiving those measurements as a single information used to build the map and estimate location with increased accuracy. The second important aspect is the ability of SLAM to recover from the incorrect loop closing detection. Today, although there are several solutions that can cope with few incorrect detections, multiple wrong pose constraints will render any SLAM algorithm useless.

Most important advancements in future SLAM algorithms will come from the better understanding of the sensor measurements. When robots will become capable of accurately segmenting objects from the environment and make connections between those objects, possibility of wrong loop detection or even wrong feature matching will become much less possible. The problem is that robot has to be able to do that regardless of the sensor setup, and has to do that in real-time. Although, current algorithms cannot quite cope with those scenarios, they are improving very fast, and advancements in the artificial intelligence and robot vision will soon make such scenarios a reality. When this does occur it will mark the moment in mobile robotics which will be of the same importance as the introduction of the first SLAM algorithm.

A

Appendicies

A.1 PLANAR SURFACE SEGMENT COVARIANCE TRANSFORMATION

The parameter uncertainties of planar surface segment $F_{i,m}$ are transformed from $S_{F_{i,m}}$ into local coordinate frame $S_{F_{j,n}}$ of planar surface segment $F_{j,m}$ as (matrices E, $\Sigma_{q_{i,m}}$, C and $P_{n,i}$ are defined in Sec. 4.3.3).

$$\tilde{\Sigma}_{q_{j,n}} = E\Sigma_{q_{i,m}}E^T + CP_{n,i}C^T \tag{A.1}$$

In general case, given surface segment pair $(F_{i,m}, F_{j,n})$, their perturbation vectors $(q_{i,m}, q_{jn})$ and rotation matrix $R_{m,n}$ and translation vector $t_{m,n}$ between their local maps M_m and M_n we can define a non-linear function h that transforms parameters of $F_{j,n}$ into $S_{F_{i,m}}$:

$$h = \begin{bmatrix} \begin{bmatrix} {}^F x_{j,n}^T \\ {}^F y_{j,n}^T \end{bmatrix} R_{m,n} \frac{{}^F z_{i,m} + \begin{bmatrix} {}^F x_{i,m} & {}^F y_{i,m} \end{bmatrix} s_{i,m}}{\sqrt{1 + s_{i,m}^T s_{i,m}}} - \frac{s_{j,n}}{\sqrt{1 + s_{j,n}^T s_{j,n}}} \\ r_{i,m} + ({}^F t_{i,m}^T - ({}^F t_{j,n} - t_{m,n})^T R_{m,n}) \frac{{}^F z_{i,m} + \begin{bmatrix} {}^F x_{i,m} & {}^F y_{i,m} \end{bmatrix} s_{i,m}}{\sqrt{1 + s_{i,m}^T s_{i,m}}} - r_{j,n} \end{bmatrix} \tag{A.2}$$

where first two rows represent transformation of the x, y coordinates of the normal ${}^F n_{i,m}$, third row represents transformation of the distance ${}^F \rho_{i,m}$, vector $s = [s_x \; s_y]$ is a part of perturbation vector q describing uncertainties of ${}^F n$ in S_F, r represents uncertainty ${}^F \rho$, and vectors ${}^F x$, ${}^F y$ and ${}^F z$ represent columns of the rotation matrices:

$${}^F R_{i,m} = \begin{bmatrix} {}^F x_{i,m} & {}^F y_{i,m} & {}^F z_{i,m} \end{bmatrix} \quad {}^F R_{j,n} = \begin{bmatrix} {}^F x_{j,n} & {}^F y_{j,n} & {}^F z_{j,n} \end{bmatrix}. \tag{A.3}$$

In order to get expected values of the transformed parameters e defined in equation (4.46) we evaluate h for $s = 0$ and $r = 0$

$$e = h|_{s=0,r=0} = \begin{bmatrix} \begin{bmatrix} {}^F x_{j,n}^T \\ {}^F y_{j,n}^T \end{bmatrix} R_{m,n} {}^F z_{i,m} \\ ({}^F t_{i,m}^T - ({}^F t_{j,n} - t_{m,n})^T R_{m,n}) {}^F z_{i,m} \end{bmatrix} \tag{A.4}$$

Jacobian matrix C of the function h given with (A.2) can be calculated as

$$C = \left.\frac{\partial h}{\partial w}\right|_{q_{i,m}=0, q_{j,n}=0} = \begin{bmatrix} \begin{bmatrix} {}^F x_{j,n}^T \\ {}^F y_{j,n}^T \end{bmatrix} J_\phi(\phi, {}_z^F j, n) & 0^{2\times 3} \\ -({}^F t_{j,n} - t_{m,n})^T J_\phi(\phi, {}_z^F j, n) & {}^F z_{j,n}^T R_{m,n}^T \end{bmatrix} \tag{A.5}$$

where vector $w = [\phi_{m,n}\ t_{m,n}]$, vector $\phi_{m,n} = [\alpha\ \beta\ \Theta]$ represents Euler angles corresponding to rotation matrix $R_{m,n}$, and Jacobian $J_\phi(\phi, {}^F z_{j,n})$ is calculated as

$$J_\phi(\phi, p) = \frac{\partial(R(\psi)p)}{\partial \psi}|_{\psi=\phi} \tag{A.6}$$

Calculation of matrix E from the function h is done using the following equation

$$E = \left.\frac{\partial h}{\partial q}\right|_{q_{i,m}=0, q_{j,n}=0} = \begin{bmatrix} \begin{bmatrix} {}^F x_{j,n}^T \\ {}^F y_{j,n}^T \end{bmatrix} R_{m,n} \begin{bmatrix} {}^F x_{i,m}^T \\ {}^F y_{i,m}^T \end{bmatrix} & 0 \\ ({}^F t_{i,m}^T - ({}^F t_{j,n} - t_{m,n})^T R_{m,n}) \begin{bmatrix} {}^F x_{i,m}^T \\ {}^F y_{i,m}^T \end{bmatrix} & 1 \end{bmatrix} \tag{A.7}$$

A.2 SPECIAL EUCLIDEAN GROUP SE(3)

The group $\mathsf{SE}(3)$ describes a 6 DoF rigid body pose and is formed as a semi-direct product of the Euclidean space vector $\mathbb{R}^3$ and the special orthogonal group $\mathsf{SO}(3)$[1], corresponding to translational and rotational parts, respectively. This group is defined as

$$\mathsf{SE}(3) = \left\{ \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \subset \mathbb{R}^{4\times 4} \,|\, \{R, t\} \in \mathsf{SO}(3) \times \mathbb{R}^3 \right\}.$$

A Euclidean space vector representing the pose of a rigid body consisting of position $t = [t_1\ t_2\ t_3]$ and orientation $\phi = [\phi_1\ \phi_2\ \phi_3]$ vectors is obtained by concatenating the two, $x = [t\ \phi]^{\mathrm{T}}$. Mapping of the pertaining Euclidean space to Lie algebra, i.e., $(\cdot)^\wedge_{\mathsf{SE}(3)} : \mathbb{R}^6 \to \mathfrak{se}(3)$, is then constructed as

$$x^\wedge_{\mathsf{SE}(3)} = \begin{bmatrix} \phi^\wedge_{\mathsf{SO}(3)} & t \\ 0 & 0 \end{bmatrix} \in \mathfrak{se}(3), \tag{A.8}$$

$$\phi^\wedge_{\mathsf{SO}(3)} = \begin{bmatrix} 0 & -\phi_3 & \phi_2 \\ \phi_3 & 0 & -\phi_1 \\ -\phi_2 & \phi_1 & 0 \end{bmatrix} \in \mathfrak{so}(3), \tag{A.9}$$

[1] The Euclidean space can be formed only by employing direct product, while other ways to concatenate Lie groups also exist, i.e., semi-direct product, twisted product etc.

while its inverse, $(\cdot)^{\vee}_{\mathsf{SE}(3)} : \mathfrak{se}(3) \to \mathbb{R}^6$, follows trivially from (A.8) and (A.9). The exponential, performing mapping $\exp_{\mathsf{SE}(3)} : \mathfrak{se}(3) \to \mathsf{SE}(3)$, is determined as follows:

$$\begin{aligned} \exp^{\wedge}_{\mathsf{SE}(3)}(x) &= \begin{bmatrix} C & Lt \\ \mathbf{0} & 1 \end{bmatrix} \\ C &= \exp^{\wedge}_{\mathsf{SO}(3)}(\phi) \\ &= \cos(|\phi|)I + (1-\cos(|\phi|))\frac{\phi\phi^{\mathrm{T}}}{|\phi|^2} + \sin(|\phi|)\frac{\phi^{\wedge}_{\mathsf{SO}(3)}}{|\phi|} \\ L &= \frac{\sin(|\phi|)}{|\phi|}I + (1-\frac{\sin(|\phi|)}{|\phi|})\frac{\phi\phi^{\mathrm{T}}}{|\phi|^2} + \frac{1-\cos(|\phi|)}{|\phi|^2}\phi^{\wedge}_{\mathsf{SO}(3)}. \end{aligned} \tag{A.10}$$

The logarithm, performing the mapping $\log_{\mathsf{SE}(3)} : \mathsf{SE}(3) \to \mathfrak{se}(3)$, is calculated by deconstructing X, and determining ϕ by using

$$\begin{aligned} \log_{\mathsf{SO}(3)}(X) &= \frac{\theta}{2\sin(\theta)}(X - X^{\mathrm{T}}) \\ \text{s.t.} \quad & 1 + 2\cos(\theta) = \mathrm{Tr}(X) \\ & \begin{cases} \theta \neq 0 & -\pi < \theta < \pi \\ \theta = 0 & \log(X) = 0 \end{cases}. \end{aligned} \tag{A.11}$$

Then, from (A.10), we can determine t. In order to determine the adjoints for $\mathsf{SE}(3)$, we need to deconstruct the state $X \in \mathsf{SE}(3)$ and the vector $x \in \mathbb{R}^6$. Firstly, we extract the rotation part C and the translation part t from X, and secondly, we split the translation part t and the orientation part ϕ from x. Then, the adjoints $\mathrm{Ad}_{\mathsf{SE}(3)}$ and $\mathrm{ad}_{\mathsf{SE}(3)}$ are

$$\mathrm{Ad}_{\mathsf{SE}(3)}(X) = \begin{bmatrix} C & tC \\ \mathbf{0} & C \end{bmatrix}, \ \mathrm{ad}_{\mathsf{SE}(3)}(x) = \begin{bmatrix} \phi^{\wedge}_{\mathsf{SO}(3)} & t^{\wedge}_{\mathsf{SO}(3)} \\ 0 & \phi^{\wedge}_{\mathsf{SO}(3)} \end{bmatrix}$$

A.3 Derivation of the Lie measurement Jacobian $\mathcal{H}$

When the loop closing occurs between the states X_i and X_j we need to evaluate

$$\mathcal{H}_{n+1} = \frac{\partial}{\partial \varepsilon}\left[\log^{\vee}_{\mathsf{G}}\left(h(X_i, X_j)^{-1} h\big((X_i, X_j)\exp^{\wedge}_{\mathsf{G}}(\varepsilon)\big)\right)\right]\Big|_{\varepsilon=0}$$

in order to perform update using (5.20). First we evaluate

$$h\big((X_i, X_j)\exp^{\wedge}_{\mathsf{G}}(\varepsilon)\big) = (X_j \exp^{\wedge}_{\mathsf{G}}(\varepsilon_j))^{-1} X_i \exp^{\wedge}_{\mathsf{G}}(\varepsilon_i) = \exp^{\wedge}_{\mathsf{G}}(-\varepsilon_j) X_j^{-1} X_i \exp^{\wedge}_{\mathsf{G}}(\varepsilon_i).$$

Then, we evaluate

$$\begin{aligned} & h(X_i, X_j)^{-1} h\big((X_i, X_j)\exp^{\wedge}_{\mathsf{G}}(\varepsilon)\big) = \\ &= (X_j^{-1}X_i)^{-1} \exp^{\wedge}_{\mathsf{G}}(-\varepsilon_j) X_j^{-1} X_i \exp^{\wedge}_{\mathsf{G}}(\varepsilon_i) \\ &= X_i^{-1} X_j \exp^{\wedge}_{\mathsf{G}}(-\varepsilon_j) X_j^{-1} X_i \exp^{\wedge}_{\mathsf{G}}(\varepsilon_i) \\ &= X_i^{-1} \exp^{\wedge}_{\mathsf{G}}(\mathrm{Ad}(X_j)(-\varepsilon_j)) X_j X_j^{-1} X_i \exp^{\wedge}_{\mathsf{G}}(\varepsilon_i) \\ &= X_i^{-1} \exp^{\wedge}_{\mathsf{G}}(\mathrm{Ad}(X_j)(-\varepsilon_j)) X_i \exp^{\wedge}_{\mathsf{G}}(\varepsilon_i) \\ &= \exp^{\wedge}_{\mathsf{G}}(\mathrm{Ad}(X_i^{-1})\,\mathrm{Ad}(X_j)(-\varepsilon_j)) X_i^{-1} X_i \exp^{\wedge}_{\mathsf{G}}(\varepsilon_i) \\ &= \exp^{\wedge}_{\mathsf{G}}(\mathrm{Ad}(X_i^{-1})\,\mathrm{Ad}(X_j)(-\varepsilon_j)) \exp^{\wedge}_{\mathsf{G}}(\varepsilon_i). \end{aligned}$$

Using Baker–Campbell–Hausdorff formula we obtain

$$\begin{aligned}
\mathcal{H}_{n+1} &= \frac{\partial}{\partial\varepsilon}\big[\varepsilon_i + \Phi(\varepsilon_i)\,\mathrm{Ad}(X_i^{-1})\,\mathrm{Ad}(X_j)(-\varepsilon_j) + \cdots\big]\big|_{\varepsilon=0} \\
&= [\underbrace{0\cdots}_{1:j-1}\overbrace{-\Phi(\varepsilon_i)\,\mathrm{Ad}(X_i^{-1})Ad(X_j)}^{j}\underbrace{\cdots 0\cdots}_{j+1:i-1}\overbrace{I + \frac{\partial}{\partial\varepsilon}(\Phi(\varepsilon_i)\,\mathrm{Ad}(X_i^{-1})\,\mathrm{Ad}(X_j)(-\varepsilon_j))}^{i}\underbrace{\cdots 0}_{i+1:n+1}]\big|_{\varepsilon=0} \\
&= [\underbrace{0\cdots}_{1:j-1}\overbrace{-\Phi(\varepsilon_i)\,\mathrm{Ad}(X_i^{-1})\,\mathrm{Ad}(X_j)}^{j}\underbrace{\cdots 0\cdots}_{j+1:i-1}\overbrace{I - \mathrm{Ad}(X_i^{-1})\,\mathrm{Ad}(X_j)\varepsilon_j\frac{\partial}{\partial\varepsilon}(\Phi(\varepsilon_i))}^{i}\underbrace{\cdots 0}_{i+1:n+1}]\big|_{\varepsilon=0} \\
&= [\underbrace{0\cdots}_{1:j-1}\overbrace{-\Phi(0)\,\mathrm{Ad}(X_i^{-1})\,\mathrm{Ad}(X_j)}^{j}\underbrace{\cdots 0\cdots}_{j+1:i-1}\overbrace{I}^{i}\underbrace{\cdots 0}_{i+1:n+1}] \\
\Phi(0) &= \sum_{n=0}^{\infty}\frac{B_n\,\mathrm{ad}(0)^n}{n!} = \frac{B_0\,\mathrm{ad}(0)^0}{0!} + 0 = 1.
\end{aligned}$$

Finally, we have

$$\mathcal{H}_{n+1} = [\underbrace{0\cdots}_{1:j-1}\overbrace{-\mathrm{Ad}(X_i^{-1})\,\mathrm{Ad}(X_j)}^{j}\underbrace{\cdots 0\cdots}_{j+1:i-1}\overbrace{I}^{i}\underbrace{\cdots 0}_{i+1:n+1}]. \tag{A.12}$$

BIBLIOGRAPHY

[1] S Thrun, Y Liu, D Koller, A Y Ng, Z Ghahramani, and H Durrant-Whyte. Simultaneous Localization and Mapping With Sparse Extended Information Filters. *The International Journal of Robotics Research*, 23(7-8):693–716, 2004.

[2] McKinsey Global Institute. *http://www.mckinsey.com/insights/mgi*, 2015.

[3] World Economic Forum. *http://www.weforum.org/*, 2015.

[4] European Commission. *http://ec.europa.eu/index_hr.htm*, 2015.

[5] Ioannis Kostavelis and Antonios Gasteratos. Semantic mapping for mobile robotics tasks: A survey. *Robotics and Autonomous Systems*, 66:86 – 103, 2015.

[6] IEEE Robotics and Automation Society. IEEE Standard for Robot Map Data Representation for Navigation. *IEEE Standard for Robot Map Data Representation for Navigation*, pages 1–54, 2015.

[7] Chuho Yi, Seungdo Jeong, and Jungwon Cho. Map representation for robots. *Smart Computing Review*, 2(1):18–27, 2012.

[8] I.A. Getting. Perspective/navigation The Global Positioning System. *IEEE Spectrum*, 30(12):43–47, 1993.

[9] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *International Conference on Mobile Computing and Networking*, pages 32–43, New York, NY, USA, 2000. ACM.

[10] de Melo Leonimer Flávio, Rosário João Mauricio, and da Silveira Junior Almiro Franco. Mobile Robot Indoor Autonomous Navigation with Position Estimation Using RF Signal Triangulation Positioning. *Positioning*, 4(1):16, 2013.

[11] Guoyu Fu, Jin Zhang, Wenyuan Chen, Fengchao Peng, Pei Yang, and Chunlin Chen. Precise localization of mobile robots via odometry and wireless sensor network. *International Journal of Advanced Robotic Systems*, 10(4):203, 2013.

[12] Drew Gislason. *Zigbee Wireless Networking*. Newnes, Newton, MA, USA, pap/onl edition, 2008.

[13] Mariana Rampinelli, Vitor Buback Covre, Felippe Mendonça de Queiroz, Raquel Frizera Vassallo, Teodiano Freire Bastos-Filho, and Manuel Mazo. An intelligent space for mobile robot localization using a multi-camera system. *Sensors*, 14(8):15039–15064, 2014.

[14] Carlos Sánchez, Pierluigi Taddei, Simone Ceriani, Erik Wolfart, and Vítor Sequeira. Localization and tracking in known large environments using portable real-time 3d sensors. *Computer Vision and Image Understanding*, 149:197 – 208, 2016.

[15] B. Behzadian, P. Agarwal, W. Burgard, and G. D. Tipaldi. Monte carlo localization in hand-drawn maps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4291–4296, Sept 2015.

[16] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1322–1328, 1999.

[17] Julian Mason, Susanna Ricco, and Ronald Parr. Textured occupancy grids for monocular localization without features. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5800–5806. IEEE, May 2011.

[18] C. F. Olson. Probabilistic self-localization for mobile robots. *IEEE Transactions on Robotics and Automation*, 16(1):55–66, Feb 2000.

[19] G Dissanayake, S Huang, Z Wang, and R Ranasinghe. A review of recent developments in Simultaneous Localization and Mapping. In *Industrial and Information Systems*, pages 477–482, 2011.

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

[21] T Bailey and H Durrant-Whyte. Simultaneous localization and mapping (SLAM): part II. *IEEE Robotics & Automation Magazine*, 13(3):108–117, 2006.

[22] C Cadena, L Carlone, H Carrillo, Y Latif, D Scaramuzza, J Neira, I Reid, and J J Leonard. Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[23] H Zhao, M Chiba, R Shibasaki, X Shao, J Cui, and H Zha. SLAM in a dynamic large outdoor environment using a laser scanner. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1455–1462, 2008.

[24] Hiroshi Morioka, Sangkyu Yi, and Osamu Hasegawa. Vision-based mobile robot's SLAM and navigation in crowded environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3998–4005, 2011.

[25] Chee Sing Lee, Daniel E. Clark, and Joaquim Salvi. SLAM with dynamic targets via single-cluster PHD filtering. *IEEE Journal on Selected Topics in Signal Processing*, 7(3):543–552, 2013.

[26] C. Stachniss, D. Hahnel, and W. Burgard. Exploration with active loop-closing for FastSLAM. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1505–1510, 2004.

[27] R Sim and N Roy. Global A-Optimal Robot Exploration in SLAM. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 661–666, 2005.

[28] Cyrill Stachniss, G Grisetti, and W Burgard. Information Gain-based Exploration Using Rao-Blackwellized Particle Filters. In *Robotics: Science and Systems (RSS)*, pages 65–72, 2005.

[29] Cindy Leung, Shoudong Huang, and Gamini Dissanayake. Active SLAM using model predictive control and attractor based exploration. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5031, 2006.

[30] Josep Aulinas, Yvan Petillot, Joaquim Salvi, and Xavier Lladó. The SLAM Problem: A Survey. In *Conference on Artificial Intelligence Research and Development*, pages 363–371. IOS Press, 2008.

[31] G Grisetti, R Kummerle, C Stachniss, and W Burgard. A Tutorial on Graph-Based SLAM. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[32] M W M G Dissanayake, P Newman, S Clark, H F Durrant-Whyte, and M Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001.

[33] Jose Guivant and Eduardo Nebot. Optimization of the Simultaneous Localization and Map Building Algorithm for Real Time Implementation. *IEEE Transactions on Robotics and Automation*, 17:242–257, 2001.

[34] Jose Guivant and Edurado Nebot. Improving computational and memory requirements of simultaneous localization and map building algorithms. *IEEE International Conference on Robotics and Automation (ICRA)*, 3:2731–2736, 2002.

[35] Juan D Tardós, José Neira, Paul M Newman, and John J Leonard. Robust Mapping and Localization in Indoor Environments Using Sonar Data. *The International Journal of Robotics Research*, 21(4):311–330, 2002.

[36] P. Kohlhepp, P. Pozzo, M. Walther, and R. Dillmann. Sequential 3D-SLAM for mobile action planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 722–729, 2004.

[37] Jan Weingarten and Roland Siegwart. 3D SLAM using planar segments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3062–3067, 2006.

[38] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007.

[39] Javier Civera, Oscar G Grasa, Andrew J Davison, and J M M Montiel. 1-Point RANSAC for EKF Filtering. Application to Real-Time Structure from Motion and Visual Odometry. *Journal of Field Robotics*, 27(5):609–631, 2010.

[40] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *National Conference on Artificial Intelligence (AAAI)*, pages 593–598. AAAI, 2002.

[41] Michael Montemerlo, Sebastian Thrun, Daphne Roller, and Ben Wegbreit. FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping That Provably Converges. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1151–1156, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.

[42] Zhan Wang, Shoudong Huang, and Gamini Dissanayake. *Implementation Issues and Experimental Evaluation of D-SLAM*, pages 155–166. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[43] Matthew Walter, Ryan Eustice, and John Leonard. A provably consistent method for imposing sparsity in feature-based slam information filters. In *International Symposium on Robotics Research (ISRR)*, pages 214–234, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[44] R M Eustice, H Singh, and J J Leonard. Exactly Sparse Delayed-State Filters for View-Based SLAM. *IEEE Transactions on Robotics*, 22(6):1100–1114, 2006.

[45] T Davis. *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2006.

[46] Frank Dellaert and Michael Kaess. Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing. *The International Journal of Robotics Research*, 25(12):1181–1203, 2006.

[47] K Konolige, G Grisetti, R Kümmerle, W Burgard, B Limketkai, and R Vincent. Efficient Sparse Pose Adjustment for 2D mapping. In *EEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 22–29, 2010.

[48] Kurt Konolige. Sparse Sparse Bundle Adjustment. In *British Machine Vision Conference*, pages 102.1—-102.11. BMVA Press, 2010.

[49] Taihú Pire, Thomas Fischer, Javier Civera, Pablo de Cristóforis, and Julio Jacobo-Berlles. Stereo parallel tracking and mapping for robot localization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1373–1378, 2015.

[50] G Bourmaud and R Mégret. Robust large scale monocular visual SLAM. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1638–1647, 2015.

[51] Frank Dellaert. Factor Graphs and GTSAM: A Hands-on Introduction. Technical report, GT RIM, 2012.

[52] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. http://ceres-solver.org, 2010.

[53] Lukas Polok, Viorela Ila, Marek Solony, Pavel Smrz, and Pavel Zemcik. Incremental Block Cholesky Factorization for Nonlinear Least Squares in Robotics. In *Intelligent Autonomous Vehicles Symposium (IFAC)*, Berlin, Germany, 2013.

[54] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, 24(6):1365–1378, 2008.

[55] R Kuemmerle, G Grisetti, H Strasdat, K Konolige, and W Burgard. g2o: A General Framework for Graph Optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3607–3613, Shanghai, China, 2011.

[56] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, 2012.

[57] Jakob Engel, Thomas Schops, and Daniel Cremers. *LSD-SLAM: Large-Scale Direct Monocular SLAM*. Springer International Publishing, 2014.

[58] J Engel, J Stueckler, and D Cremers. Large-Scale Direct SLAM with Stereo Cameras. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1935–1942, 2015.

[59] R Mur-Artal, J M M Montiel, and J D Tardós. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[60] Raul Mur-Artal and Juan D Tardós. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. *IEEE Transactions on Robotics*, PP(99):1–8, 2017.

[61] Raul Mur-Artal and Juan D. Tardós. Visual-inertial monocular SLAM with map reuse. *IEEE Robotics and Automation Letters*, abs/1610.05949, 2016.

[62] Richard a Newcombe, David Molyneaux, David Kim, Andrew J. Davison, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: Real-Time Dense Surface Mapping and Tracking. *IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, 2011.

[63] M. Kaess. Simultaneous localization and mapping with infinite planes. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4605–4611, Seattle, WA, May 2015.

[64] L. M. Paz, P. PiniÉs, J. D. TardÓs, and J. Neira. Large-scale 6-dof slam with stereo-in-hand. *IEEE Transactions on Robotics*, 24(5):946–957, Oct 2008.

[65] Renato F. Salas-Moreno, Richard a. Newcombe, Hauke Strasdat, Paul H J Kelly, and Andrew J. Davison. SLAM++: Simultaneous localisation and mapping at the level of objects. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1352–1359, 2013.

[66] Renato F. Salas-Moreno, Ben Glocken, Paul H. J. Kelly, and Andrew J. Davison. Dense Planar SLAM. *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 157–164, 2014.

[67] Xiang Gao and Tao Zhang. Robust rgb-d simultaneous localization and mapping using planar point features. *Robotics and Autonomous Systems*, 72:1 – 14, 2015.

[68] Thomas Whelan, Michael Kaess, Hordur Johannsson, Maurice Fallon, John J. Leonard, and John Mcdonald. Real-time large-scale dense rgb-d slam with volumetric fusion. *The International Journal of Robotics Research*, 34(4-5):598–626, April 2015.

[69] E. Olson. M3rsm: Many-to-many multi-resolution scan matching. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5815–5821, May 2015.

[70] Kurt Konolige, Giorgio Grisetti, Rainer Kummerle, Wolfram Burgard, Benson Limketkai, and Régis Vincent. Sparse pose adjustment for 2d mapping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, 10 2010.

[71] Paloma de la Puente and Diego Rodriguez-Losada. Feature based graph {SLAM} with high level representation using rectangles. *Robotics and Autonomous Systems*, 63, Part 1:80 – 88, 2015.

[72] Kaustubh Pathak, Andreas Birk, Narunas Vaskevicius, Max Pfingsthor, Soren Schwertfeger, and Jann Poppinga. Online 3D SLAM by Registration of Large Planar Surface Segments and Closed Form Pose-Graph Relaxation. *Journal of Field Robotics*, 27(1):52–84, 2010.

[73] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, pages 147–151, 1988.

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

[75] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision (ECCV)*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[76] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, Nara, Japan, November 2007.

[77] C. Forster, M. Pizzoli, and D. Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22, May 2014.

[78] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2):249–265, April 2017.

[79] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. On-manifold preintegration for real-time visual–inertial odometry. *IEEE Transactions on Robotics*, 33(1):1–21, Feb 2017.

[80] 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.

[81] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart. Robust visual inertial odometry using a direct ekf-based approach. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 298–304, Sept 2015.

[82] Kurt Konolige, Motilal Agrawal, and Joan Solà. *Large-Scale Visual Odometry for Rough Terrain*, pages 201–212. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[83] Igor Cvišić, Ivan Marković, Josip Ćesić, and Ivan Petrović. SOFT-SLAM: Stereo Visual SLAM for High-Speed Autonomous UAVs. *Journal of Field Robotics*, 2017.

[84] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, 2012.

[85] Andrea Censi. An ICP variant using a point-to-line metric. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 19–25, May 2008.

[86] Javier Gonzalez and Rafael Gutierrez. Direct motion estimation from a range scan sequence. *Journal of Robotic Systems*, 16(2):73–80, 1999.

[87] M. Jaimez, J. G. Monroy, and J. Gonzalez-Jimenez. Planar odometry from a radial laser scanner. a range flow-based approach. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4479–4485, May 2016.

[88] Albert Diosi and Lindsay Kleeman. Fast laser scan matching using polar coordinates. *The International Journal of Robotics Research*, 26(10):1125–1153, 2007.

[89] C. McManus, P. Furgale, and T. D. Barfoot. Towards appearance-based methods for lidar sensors. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1930–1935, May 2011.

[90] A Nuchter, K Lingemann, and J Hertzberg. 6D SLAM-3D Mapping Outdoor Environments. *Journal of Field Robotics*, 24:699–722, 2007.

[91] C. McManus, P. Furgale, B. Stenning, and T. D. Barfoot. Visual teach and repeat using appearance-based lidar. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 389–396, May 2012.

[92] Hang Dong and Timothy D. Barfoot. *Lighting-Invariant Visual Odometry using Lidar Intensity Imagery and Pose Interpolation*, pages 327–342. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[93] C. H. Tong and T. D. Barfoot. Gaussian process gauss-newton for 3d laser-based visual odometry. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5204–5211, May 2013.

[94] S. Anderson and T. D. Barfoot. Towards relative continuous-time slam. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1033–1040, May 2013.

[95] Robert Zlot and Michael Bosse. *Efficient Large-Scale 3D Mobile Mapping and Surface Reconstruction of an Underground Mine*, pages 479–493. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[96] 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, October 2012.

[97] Hartmut Surmann, Andreas Nüchter, and Joachim Hertzberg. An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments. *Robotics and Autonomous Systems*, 45(3-4):181–198, 2003.

[98] S Thrun, D Hähnel, D Ferguson, M Montemerlo, R Triebel, W Burgard, C Baker, Z Omohundro, S Thayer, and W Whittaker. A System for Volumetric Robotic Mapping of Abandoned Mines. *IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[99] Paul Besl and Neil McKay. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[100] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155, April 1992.

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

[102] Paul Scovanner, Saad Ali, and Mubarak Shah. A 3-dimensional sift descriptor and its application to action recognition. In *International Conference on Multimedia (ACM)*, MM '07, pages 357–360, New York, NY, USA, 2007.

[103] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1848–1853, Piscataway, NJ, USA, 2009. IEEE Press.

[104] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems Conference (RSS)*, Pittsburgh, PA, July 2014.

[105] J. Zhang and S. Singh. Visual-lidar odometry and mapping: low-drift, robust, and fast. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2174–2181, May 2015.

[106] Todor Stoyanov, Martin Magnusson, Henrik Andreasson, and Achim J Lilienthal. Fast and accurate scan registration through minimization of the distance between compact 3D NDT representations. *The International Journal of Robotics Research*, 31(12):1377–1393, 2012.

[107] Mark Cummins and Paul Newman. Appearance-only slam at large scale with fab-map 2.0. *The International Journal of Robotics Research*, 30(9):1100–1123, 2011.

[108] C. K. Chow and C. N. Liu. Approximating Discrete Probability Distributions with Dependence Trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

[109] Negre Carrasco Pep Lluis, Francisco Bonin-Font, and Gabriel Oliver-Codina. Global image signature for visual loop-closure detection. *Autonomous Robots*, 40(8):1403–1417, Dec 2016.

[110] M. Labbé and F. Michaud. Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Transactions on Robotics*, 29(3):734–745, June 2013.

[111] R. Arroyo, P. F. Alcantarilla, L. M. Bergasa, J. J. Yebes, and S. Bronte. Fast and effective visual place recognition using binary codes and disparity information. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3089–3094, Sept 2014.

[112] Xin Yang and Kwang-Ting Cheng. Ldb: An ultra-fast feature for scalable augmented reality on mobile devices. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 49–57, Nov 2012.

[113] R. Mur-Artal and J. D. Tardós. Fast relocalisation and loop closing in keyframe-based slam. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 846–853, May 2014.

[114] Stephanie Lowry, Niko Sünderhauf, Paul Newman, John J. Leonard, David Cox, Peter Corke, and Michael J. Milford. Visual place recognition: A survey. *IEEE Transactions on Robotics*, 32(1):1–19, 2016.

[115] M. Bosse and R. Zlot. Place recognition using keypoint voting in large 3d lidar datasets. In *IEEE International Conference on Robotics and Automation*, pages 2677–2684, May 2013.

[116] A. Gawel, T. Cieslewski, R. Dubé, M. Bosse, R. Siegwart, and J. Nieto. Structure-based vision-laser matching. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 182–188, Oct 2016.

[117] T. Cieslewski, E. Stumm, A. Gawel, M. Bosse, S. Lynen, and R. Siegwart. Point cloud descriptors for place recognition using sparse visual information. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4830–4836, May 2016.

[118] B. Steder, M. Ruhnke, S. Grzonka, and W. Burgard. Place recognition in 3D scans using a combination of bag of words and point feature based relative pose estimation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[119] M. Magnusson, H. Andreasson, A. Nuchter, and A. J. Lilienthal. Appearance-based loop detection from 3d laser data using the normal distributions transform. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 23–28, May 2009.

[120] T. Röhling, J. Mack, and D. Schulz. A fast histogram-based similarity measure for detecting loop closures in 3-d lidar data. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 736–741, Sept 2015.

[121] Karl Granström, Thomas B Schön, Juan I Nieto, and Fabio T Ramos. Learning to close loops from range data. *The International Journal of Robotics Research*, 30(14):1728–1754, 2011.

[122] Renaud Dubé, Daniel Dugas, Elena Stumm, Juan I. Nieto, Roland Siegwart, and Cesar Cadena. Segmatch: Segment based loop-closure for 3d point clouds. *CoRR*, abs/1609.07720, 2016.

[123] Alexei A. Makarenko, Stefan B. Williams, Frederic Bourgault, and Hugh F. Durrant-Whyte. An experiment in integrated exploration. In *IEEE / RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 534–539, 2002.

[124] S. Huang, N. Kwok, G. Dissanayake, Q. Ha, and G. Fang. Multi-step look-ahead trajectory planning in slam: Possibility and necessity. In *IEEE / RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5031, 2006.

[125] Thomas Kollar and Nicholas Roy. Using reinforcement learning to improve exploration trajectories for error minimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3338–3343, 2006.

[126] Hugh F. Durrant-Whyte Eric W. Nettleton, Peter W. Gibbens. Closed form solutions to the multiple-platform simultaneous localization and map building (slam) problem. In *Sensor Fusion: Architectures, Algorithms, and Applications*, pages 4051 – 4061, 2000.

[127] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[128] Michael Kaess, Viorela Ila, Richard Roberts, and Frank Dellaert. *The Bayes Tree: An Algorithmic Foundation for Probabilistic Robot Mapping*, pages 157–173. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[129] Kruno Lenac, Andrej Kitanov, Robert Cupec, and Ivan Petrović. Fast planar surface 3d slam using lidar. *Robotics and Autonomous Systems*, 92:197 – 220, 2017.

[130] Kruno Lenac, Andrej Kitanov, Ivan Maurović, Marija Dakulović, and Ivan Petrović. Fast active slam for accurate and complete coverage mapping of unknown environments. In *International Conference on Intelligent Autonomous Systems (IAS)*, pages 415–428, 2016.

[131] Simon Julier and Jeffrey Uhlmann. Unscented Filtering and Non Linear Estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

[132] Junhao Xiaoa, Jianhua Zhangb, Benjamin Adlera, Houxiang Zhangc, and Jianwei Zhanga. Three-dimensional point cloud plane segmentation in both structured and unstructured environments. *Robotics and Autonomous Systems*, 61(12):1641–1652, 2013.

[133] J. Xiao, B. Adler, J. Zhang, and H. Zhang. Planar Segment Based Three-dimensional Point Cloud Registration in Outdoor Environments. *Journal of Field Robotics*, 30(4):552–582, 2013.

[134] Robert Cupec, Emmanuel K Nyarko, Damir Filko, and Ivan Petrović. Fast pose tracking based on ranked 3D planar patch correspondences. In *International Federation of Automatic Control*, pages 108–113, 2012.

[135] R. Cupec, E. K. Nyarko, D. Filko, A. Kitanov, and I. Petrović. Place Recognition Based on Matching of Planar Surfaces and Line Segments. *The International Journal of Robotics Research*, 34(4-5):674–704, 2015.

[136] Francis Schmitt and Xin Chen. Fast segmentation of range imagery into planar regions. *Computer Vision, Graphics and Image Processing*, 45(1):42–60, 1991.

[137] Robert Cupec, Emmanuel Karlo Nyarko, Damir Filko, Andrej Kitanov, and Ivan Petrović. Global Localization Based on 3D Planar Surface Segments Detected by a 3D Camera. *Croatian Computer Vision Workshop (CCCW)*, pages 31–36, 2013.

[138] Boost. Boost library, 2016. `http://www.boost.org`.

[139] Dorit Borrmann, Andreas Nüchter, Marija Đakulović, Ivan Maurović, Ivan Petrović, Dinko Osmanković, and Jasmin Velagić. The project thermalmapper – thermal 3d mapping of indoor environments for saving energy. *International Federation of Automatic Control Symposium on Robot Control*, 45(22):31 – 38, 2012.

[140] Marija Đakulović, Šandor Ileš, and Ivan Petrović. Exploration and mapping of unknown polygonal environments based on uncertain range data. *Automatika*, 52(2):118–131, 2011.

[141] A. Ekman, A. Torne, and D. Stromberg. Exploration of polygonal environments using range data. *IEEE Transactions on Systems, Man, and Cybernetics*, 27(2):250–255, Apr 1997.

[142] M. Seder, M. Baotić, and I. Petrović. Receding horizon control for convergent navigation of a differential drive mobile robot. *IEEE Transactions on Control Systems Technology*, 25(2):653–660, March 2017.

[143] A. Stentz. Optimal and efficient path planning for partially-known environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3310–3317, May 1994.

[144] 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 *IEEE International Conference on Robotics and Automation (ICRA)*, Kobe, Japan, May 2009.

[145] François Pomerleau, M. Liu, Francis Colas, and Roland Siegwart. Challenging data sets for point cloud registration algorithms. *The International Journal of Robotics Research*, 31(14):1705–1711, December 2012.

[146] Martin Magnusson, Narunas Vaskevicius, Todor Stoyanov, Kaustubh Pathak, and Andreas Birk. Beyond points: Evaluating recent 3d scan-matching algorithms. In *International Conference on Robotics and Automation (ICRA)*, 2015.

[147] Martin Magnusson, Achim Lilienthal, and Tom Duckett. Scan Registration for Autonomous Mining Vehicles Using 3D-NDT. *Journal of Field Robotics*, 24(10):803–827, 2007.

[148] G. Pandey, J. R. McBride, and R. M. Eustice. Ford Campus vision and lidar data set. *The International Journal of Robotics Research*, 30(13):1543–1552, 2011.

[149] I. Maurović, M. Seder, K. Lenac, and I. Petrović. Path planning for active slam based on the d* algorithm with negative edge weights. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, PP(99):1–11, 2017.

[150] K. Lenac, J. Ćesić, I. Marković, I. Cvišić, and I. Petrović. Revival of filtering based SLAM? Exactly sparse delayed state filter on Lie groups. . In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (accepted for publishing)*, (accepted) 2017.

[151] Kruno Lenac, Josip Ćesić, Ivan Marković, and Ivan Petrović. Exactly sparse delayed state filter on lie groups for long-term pose graph slam. *The International Journal of Robotics Research*, 2017 (submitted for review).

[152] Igor Gilitschenski, Gerhard Kurz, Simon J. Julier, and Uwe D. Hanebeck. Unscented orientation estimation based on the Bingham distribution. *IEEE Transactions on Automatic Control*, in press:11, 2015.

[153] J. Glover and L. P. Kaelbling. Tracking 3-D rotations with the quaternion Bingham filter. Technical report, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology (MIT), 2013.

[154] Ethan Eade. *Monocular Simultaneous Localisation and Mapping*. PhD thesis, University of Cambridge, 2008.

[155] Ethan Eade, Philip Fong, Mario E Munich, and Evolution Robotics. Monocular Graph SLAM with Complexity Reduction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3017–3024, 2010.

[156] G Bourmaud, R Mégret, A Giremus, and Y Berthoumieu. Discrete Extended Kalman Filter on Lie groups. In *European Signal Processing Conference (EUSIPCO)*, pages 1–5, 2013.

[157] Christoph Hertzberg, René Wagner, Udo Frese, and Lutz Schröder. Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds. *Information Fusion*, 14(1):57–77, 2013.

[158] Guillaume Bourmaud, Rémi Mégret, Marc Arnaudon, and Audrey Giremus. Continuous-Discrete Extended Kalman Filter on Matrix Lie Groups Using Concentrated Gaussian Distributions. *Journal of Mathematical Imaging and Vision*, 51(1):209–228, 2015.

[159] Axel Barrau and Silvere Bonnabel. Intrinsic filtering on Lie groups with applications to attitude estimation. *IEEE Transactions on Automatic Control*, 60(2):436–449, 2015.

[160] Josip Ćesić, Ivan Marković, Mario Bukal, and Ivan Petrović. Extended information filter on matrix Lie groups. *Automatica*, 82:226–234, 2017.

[161] Gregory S Chirikjian. *Stochastic Models, Information Theory, and Lie Groups, Volume 2: Analytic Methods and Modern Applications*. Springer, 2012.

[162] J M Selig. Lie Groups and Lie Algebras in Robotics. In *Computational Noncommutative Algebra and Applications*, pages 101–125. Springer Science and Business Media, 2005.

[163] W. Park, Yunfeng Wang, and G. S. Chirikjian. The Path-of-probability Algorithm for Steering and Feedback Control of Flexible Needles. *The International Journal of Robotics Research*, 29(7):813–830, 2010.

[164] Guillaume Bourmaud, Remi Megret, Audrey Giremus, and Yannick Berthoumieu. From Intrinsic Optimization to Iterated Extended Kalman Filtering on Lie Groups. *Journal of Mathematical Imaging and Vision*, 55(3):284–303, 2016.

[165] Yunfeng Wang and Gregory S Chirikjian. Nonparametric Second-Order Theory of Error Propagation on Motion Groups. *The International Journal of Robotics Research*, 27(11):1258–1273, 2008.

[166] Timothy D. Barfoot and Paul T. Furgale. Associating uncertainty with three-dimensional poses for use in estimation problems. *IEEE Transactions on Robotics*, 30(3):679–693, 2014.

[167] Henrik Kretzschmar and Cyrill Stachniss. Information-theoretic compression of pose graphs for laser-based SLAM. *The International Journal of Robotics Research*, 31(11):1219–1230, 2012.

[168] N Carlevaris-Bianco, M Kaess, and R M Eustice. Generic Node Removal for Factor-Graph SLAM. *IEEE Transactions on Robotics*, 30(6):1371–1385, 2014.

[169] Igor Cvišić and Ivan Petrović. Stereo odometry based on careful feature selection and tracking. In *European Conference on Mobile Robots (ECMR)*, pages 0–5, 2015.

[170] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[171] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The EuRoC micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.

[172] Janosch Nikolic, Joern Rehder, Michael Burri, Pascal Gohl, Stefan Leutenegger, Paul T Furgale, and Roland Siegwart. A synchronized visual-inertial sensor system with FPGA pre-processing for accurate real-time SLAM. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 431–437. IEEE, 2014.

[173] Kruno Lenac, Josip Ćesić, Ivan Marković, and Ivan Petrović. Cooperative cloud slam on matrix lie groups. In *Iberian Robotics Conference (ROBOT)*, 2017.

[174] L. Riazuelo, Javier Civera, and J.M.M. Montiel. C2tam: A cloud framework for cooperative tracking and mapping. *Robotics and Autonomous Systems*, 62(4):401 – 413, 2014.

[175] Andrew Howard. Multi-robot simultaneous localization and mapping using particle filters. *The International Journal of Robotics Research*, 25(12):1243–1256, 2006.

[176] B. Kim, M. Kaess, L. Fletcher, J. Leonard, A. Bachrach, N. Roy, and S. Teller. Multiple relative pose graphs for robust cooperative mapping. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3185–3192, 2010.

[177] Keith Y. K. Leung, Timothy D. Barfoot, and Hugh H. T. Liu. Decentralized cooperative slam for sparsely-communicating robot networks: A centralized-equivalent approach. *The International Journal of Robotics Research*, 66(3):321–342, 2012.

[178] Liam Paull, Guoquan Huang, Mae L. Seto, and John J. Leonard. Communication-constrained multi-auv cooperative slam. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 509–516, 2015.

[179] A. Birk and S. Carpin. Merging occupancy grid maps from multiple robots. *Proceedings of the IEEE*, 94(7):1384–1397, 2006.

[180] Anastasios I. Mourikis and Stergios I. Roumeliotis. Predicting the performance of cooperative simultaneous localization and mapping (c-slam). *The International Journal of Robotics Research*, 25(12):1273–1286, 2006.

[181] X. S. Zhou and S. I. Roumeliotis. Multi-robot slam with unknown initial correspondence: The robot rendezvous case. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1785–1792, 2006.

[182] L. Carlone, M. Kaouk Ng, J. Du, B. Bona, and M. Indri. Rao-blackwellized particle filters multi robot slam with unknown initial correspondences and limited communication. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 243–249, 2010.

[183] A. Cunningham, K. M. Wurm, W. Burgard, and F. Dellaert. Fully distributed scalable smoothing and mapping with robust multi-robot data association. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1093–1100, 2012.

[184] V. Indelman, E. Nelson, N. Michael, and F. Dellaert. Multi-robot pose graph localization and data association from unknown initial relative poses via expectation maximization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 593–600, 2014.

[185] Sajad Saeedi, Michael Trentini, Mae Seto, and Howard Li. Multiple-Robot Simultaneous Localization and Mapping: A Review. *Journal of Field Robotics*, 33(1):3–46, 2016.

CURRICULUM VITAE

KRUNO LENAC was born in Rijeka, Croatia in 1989. He finished mathematically oriented high school Andrije Mohorovičića in Rijeka in 2008 and received his mag. ing. degree from the University of Zagreb, Faculty of Electrical Engineering and Computing (FER) in 2013. His main area of research interest is mobile robotics with the focus on localization of autonomous mobile robots and map building of unknown environments.

After his graduation in 2013 he was employed by the Faculty of Electrical Engineering and Computing, Department of Control and Computer Engineering as a researcher in the field of autonomous and cooperative systems funded by the European FP7 project "ACROSS - Centre of Research Excellence for Advanced Cooperative Systems". Once the project ended he became a researcher on the project "FER-KIET - Advanced Technologies in Power Systems and Railway Vehicles" where he worked on algorithms that increase safety in the public transportation vehicles. In 2016 he was employed by the "cloudSLAM - Cooperative cloud based simultaneous localization and mapping in dynamic environments" project funded by the Unity through knowledge fund, where he developed localization and mapping algorithms for autonomous mobile robots. In 2016 he was a visiting researcher at the University of Wurzburg for two months and in 2017 he spent three weeks as a researcher at the Karlsruhe Institute of Technology. Besides scientific work, he has also been a teaching assistant at several undergraduate and graduate courses.

He is a graduate student member of IEEE and KoREMA and in 2014 he became a technical editor of the journal "Automatika - Journal for Control, Measurement, Electronics, Computing and Communications". During his studies, he was receiving scholarships from the city of Rijeka and from the Primorsko-Goranksa county. As a recognition for successful studying, he received the “Josip Lončar” award in the academic year 2008/2009 from FER. As a result of his research, he published two journal papers and four conference papers which are listed in the sequel.

PUBLICATIONS

JOURNAL PUBLICATIONS:

1. K. Lenac, J. Ćesić, I. Marković and I. Petrović. Exactly Sparse Delayed State Filter on Lie groups for Long-term Pose Graph SLAM. *The International Journal of Robotics Research*, submitted.

2. K. Lenac, A. Kitanov, R. Cupec and I. Petrović. Fast planar surface 3D SLAM using LIDAR. *Robotics and Autonomous Systems*, 92:197-220, 2017.

3. I. Maurović, M. Seder, K. Lenac and I. Petrović. Path Planning for Active SLAM Based on the D* Algorithm With Negative Edge Weights. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 99:1-11, 2017.

CONFERENCE PUBLICATIONS:

1. K. Lenac, J. Ćesić, I. Marković, I. Cvišić and I. Petrović. Revival of filtering based SLAM? Exactly sparse delayed state filter on Lie groups. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, Canada, 2017*.

2. K. Lenac, J. Ćesić, I. Marković and I. Petrović. Cooperative Cloud SLAM on Matrix Lie Groups. In *Third Iberian Robotics Conference (ROBOT), Seville, Spain, 2017*.

3. K. Lenac, I. Maurović, and I. Petrović. Moving Objects Detection Using a Thermal Camera and IMU on a Vehicle. In *International Conference on Electrical Drives and Power Electronics (EDPE), High Tatras, Slovakia, 2015*.

4. K. Lenac, A. Kitanov, I. Maurović, M. Đakulović and I. Petrović. Fast Active SLAM for Accurate and Complete Coverage Mapping of Unknown Environments. In *Intelligent Autonomous Systems (IAS), Padua, Italy, 2014*.

ŽIVOTOPIS

KRUNO LENAC rođen je 1989. godine u Rijeci. Matematički orijentiranu gimnaziju Andrije Mohorovičića završio je 2008. godine te je diplomirao u polju elektrotehnike na Sveučilištu u Zagrebu, Fakultetu elektrotehnike i računarstva (FER) 2013. godine. Njegovi istraživački interesi uključuju mobilnu robotiku s posebnim naglaskom na autonomnu lokalizaciju i izgradnju karte nepoznatih prostora.

Nakon diplomiranja, zaposlio se na Fakultetu elektrotehnike i računarstva, u Zavodu za automatiku i računalno inženjerstvo, kao istraživač u području autonomnih kooperativnih sustava financiran od strane europskog FP7 projekta "ACROSS – znanstveni centar izvrsnosti za napredne i kooperativne sustave". Po završetku projekta ACROSS postaje istraživač na projektu "FER-KIET - Napredne tehnologije u elektroenergetskim postrojenjima i tračničkim vozilima" gdje radi na algoritmima za povećanje sigurnosti vozila koja se koriste u javnom prijevozu. U 2016. godini zapošljava se na projektu "cloudSLAM - Cooperative cloud based simultaneous localization and mapping in dynamic environments" koji financira Fond "Jedinstvo uz pomoć znanja" gdje razvija algoritme za lokalizaciju i kartiranje nepoznatih prostora autonomnim mobilnim robotima.

Tijekom 2016. godine proveo je dva mjeseca kao gostujući istraživač na Sveučilištu u Wurzburgu, a 2017. godine proveo je tri tjedna kao istraživač na Tehnološkom institutu Karlsruhe. Uz znanstveni rad asistent je i na nekoliko dodiplomskih i diplomskih predmeta. Član je udruge IEEE i KoREMA, a 2014. godine postao je tehnički urednik časopisa „Automatika – časopis za automatiku, mjerenje, elektroniku, računarstvo i komunikacije." Tijekom studija primao je stipendiju grada Rijeke i Primorsko-goranske županije. Također, dobitnik je nagrade „Josip Lončar" Fakulteta elektrotehnike i računarstva za postignuti uspjeh tijekom 2008./2009. akademske godine. Rezultate svojih istraživanja objavio je u dva časopisna i četiri konferencijska znanstvena rada.

COLOPHON

This document was typeset and inspired by the typographical look-and-feel `classicthesis` developed by André Miede, which was based on Robert Bringhurst's book on typography "The Elements of Typographic Style", and by the `FERBook` developed by Jadranko Matuško.