Bachelorarbeit Nr. 336

3D-Mapping using Planar Surface SLAM on Mobile Robots

Henning Schlüter

Course of Study: Technische Kybernetik

Examiner: Prof. Dr. rer. nat. Marc Toussaint

Supervisor: Prof. Dr. rer. nat. Marc Toussaint

Commenced: 2016-05-12
Completed: 2016-11-11

CR-Classification: I.2.1, I.2.9, I.4.6
Abstract

A feature-based real-time mapping approach using bounded planar surfaces extracted from depth images (like those produced by 3D depth cameras) will be presented in this paper. The mapping system was devised with the intention of utilising it for active autonomous exploration employing multiple mobile robots.

The system itself is based on factor-graph optimisation using iSAM2. A representation for robot poses and plane landmarks as variables in such a graph is provided. Then constraints resulting from measurements, i.e. odometry and extracted planes, between the variables are derived as factors in the graph. This results in two different factors for planes, which are then analysed in experiments. The modular system developed to handle the data and manage the factor-graph is capable of being expanded to handle further types of input measurements and multiple robots.

Zusammenfassung

Ich zeige einen Echtzeitkartographieransatz basierend auf räumlich begrenzten Ebenen als Feature, welche aus 3D-Daten eines RGB-D-Sensors extrahiert werden. Das Kartographiersystem wurde mit der Absicht entwickelt, es für eine aktive autonome Erkundung mit mehreren mobilen Robotern nutzen zu können.

# Contents

1 Introduction .............................................. 1  
  1.1 Motivation ........................................... 1  
  1.2 Goals .................................................. 2  
  1.3 Outline ............................................... 3

2 Related Work ............................................ 5

3 Background .............................................. 9  
  3.1 SLAM – Simultaneous Localization and Mapping .... 9  
        3.1.1 Filtering ...................................... 10  
        3.1.2 Smoothing .................................... 10  
  3.2 Graph SLAM .......................................... 11  
  3.3 SLAM Initialization .................................. 12  
  3.4 SLAM Data Association Problem ......................... 13

4 Theoretical Approach ................................... 15  
  4.1 Representation ....................................... 16  
        4.1.1 Robot poses ................................... 16  
        4.1.2 Plane Landmarks ............................... 16  
  4.2 Local coordinates for robot poses ..................... 19  
  4.3 Local coordinates for plane landmarks ............... 22  
        4.3.1 Lie algebra se(3) of plane transform ...... 22  
        4.3.2 Parameters of general plane equation ....... 23  
        4.3.3 Range and 2 angles .......................... 25  
  4.4 Factor for time-consecutive samples of a single robot’s poses ... 28  
  4.5 Factor for plane landmarks ........................... 30  
        4.5.1 Available input data ............................ 30  
        4.5.2 Possible measures .............................. 30  
        4.5.3 The factor using the $xy$-plane of a transformed coordinate system ......................... 34  
        4.5.4 The factor using the general form of the plane equation ......................... 35  
  4.6 Factor for prior on robot poses ........................ 37
Contents

5 Software Architecture 39
  5.1 Base components ............................................. 40
    5.1.1 MapperBase ............................................. 40
    5.1.2 Robot .................................................... 40
  5.2 Plane components .......................................... 41
  5.3 ROS components ............................................ 42

6 Experimental Results 43

7 Conclusion 49
  7.1 Future Work .................................................. 49
    7.1.1 Internalising Plane Identification ...................... 49
    7.1.2 Using inlier points as a plane-factor ................... 50
    7.1.3 Multi-agent extension .................................... 50
    7.1.4 Map Saving and Loading ................................ 51
    7.1.5 Dynamic reduction of the factor graph .................. 51
    7.1.6 Hierarchical map/factor graph .......................... 52
    7.1.7 Covariance of variables for active exploration ......... 52

Nomenclature I

List of Equations V

List of Figures IX

Bibliography XI
1 Introduction

1.1 Motivation

The application of active learning techniques to have a robot learn interactions with the physical 3D world has become more and more viable in recent years. Especially with the emergence of inexpensive 3D sensors and the improvements in computing power, learning about their immediate surroundings has proved feasible for robots to an unprecedented degree.

However, the application of such techniques to larger areas, such as entire office floors with their long corridors or even whole building complexes, requires some functional, semantic map of the area in question. Since the intention of applying active learning concepts is to have the robot generate the – for its task required – knowledge by itself, any awareness of its surroundings, i.e. a map, should be generated dynamically as well.

For this purpose Simultaneous Localization And Mapping (SLAM) techniques can be used, however, traditionally they are designed for localization and navigation purposes only. Thus, only what is needed for navigation is mapped, that is whether or not some space is free for the robot to traverse. In the case of wheel-driven robots these occupancy maps are often limited to 2D, e.g. they ignore height, which could not be traversed anyway. While these types of maps are fine for their purpose, they are of limited use beyond it as they do not contain any functional information about the given surroundings.

With the release of Kinect came the advent of methods like Kinect Fusion and others that provide dense volumetric maps. These maps contain all the data that was recorded in volumetric form, data virtually useless for machine learning without further processing since lacking semantic information. Moreover, the huge amount of data is challenging to deal with in real-time applications.

So instead, a more semantic object-based paradigm can be used, where the sensor data is preprocessed, finding objects in the observed scene, which are then used for generating the SLAM map. Having objects as part of the map allows to accumulate and store knowledge about them directly. While any kind
1 Introduction

of object might be used, planar surfaces are probably the most common structure that can be found, particularly in man-made environments. In addition to their abundance, planar surfaces often describe large and structurally important objects, e.g. walls, floors, tables, etc. Thereby they make it possible to describe the general layout of the surroundings. By charting the floors they allow to some extent to sketch the traversable area for ground-based robots. Furthermore, planes have a simple mathematical representation describing large area with few parameters. This low dimensionality in description does not only allow to save processing power and space, but also makes it more accessible for learning algorithms.

1.2 Goals

Hence, I believe planar surfaces are a good starting point for designing a SLAM algorithm with the intention of it being used in conjunction with (active) learning techniques. For this end, I have developed a factor-graph-based SLAM implementation using the planes as landmarks. This implementation serves as a platform for comparing the two plane-factors that have been developed as part of this thesis.

While I believe these kinds of maps to be of use in particular for the described applications, the evaluation of how such maps can actually be used in the relevant context is beyond the scope of this paper. Moreover, this paper will not cover the question of how to extract the plane from sensor input, but use an existing plane segmentation algorithm.

I will focus on describing two variants of a SLAM system capable of mapping the extent and locations of planes in the environment using a 3D depth camera and then present experiments analysing each mapping modality.
1.3 Outline

The remainder of this thesis is organised as follows.

Chapter 2 discusses some key related works, outlining the development of SLAM in general and the techniques I made use of in this paper in particular as well as some interesting alternate approaches to PlaneSLAM.

In chapter 3 I will lay the foundations for my work by providing a short compendium of the theoretical background of simultaneous localization and mapping and the use of factor graph for it.

I will set out in detail the mathematical derivation and description of my SLAM system in chapter 4. It first shows how I arrived at the representation for the robot poses and the plane landmarks. This representation is used for the variables in the factor graph of the SLAM. Since it is too complex to be used in optimisation, the next two sections 4.2 and 4.3 establish a local description for each of them that allows derivation and thus optimisation on the graph. The last three sections derive constraints, i.e. factors for the factor graph, between the various variables. These constraints incorporate the sensor measurements into the factor graph and thereby the SLAM system.

In chapter 5 I will present the implementation of this SLAM system, making use of a modular design. The implementation uses ROS and is designed to run on a TurtleBot2. The modularity, however, allows the system to be used in other environments with little to no effort, especially if one remains with ROS. I will further explain how this implementation could be expanded to accommodate further factor types beyond what has been developed in this paper by using the same system I used to incorporate my own factors.

The result of the experiments I ran in simulation to evaluate and compare my factors are detailed in chapter 6. If further explains why the experiments could not be run on actual hardware.

Chapter 7 concludes this thesis, summarising and drawing some conclusions from the previous chapters. It then provides directions for future work to build on the SLAM system presented in this paper.
2 Related Work

Today there is a wide range of different SLAM techniques and a variety of semantic mapping approaches that have been developed over the years. Only some key related works will be outlined in this section.

From the beginning the SLAM problem and solutions to it have seen rapid development. Smith and Cheeseman were the first to propose a solution to SLAM in their work [26]. Their approach used an Extended Kalman Filter (EKF) [17] with a state vector which was augmented with landmark positions. This technique was quite successful, basically set the norm in the early years of SLAM and improved variants are still in use today.

However, it is not without its flaws. EKF-SLAM methods linearise observations of the current state, which, however, means that the linearisation point will change as the state evolves. After many iterations, this inevitable leads to inconsistencies in the map as Julier and Uhlmann have first observed [13].

Many modern SLAM implementations break anyway from the traditional EKF-SLAM formulation. A detailed summary of recent developments in the field up to the year 2006 is given by Bailey and Durrant-Whyte in their articles [1, 4] for the IEEE Robotics Automation Magazine.

Of special interest for this paper is the development in graph-based SLAM techniques. These methods use continuous optimisation on graphical models of the entire robot trajectory. Keeping record of the entire trajectory leaves the landmarks uncorrelated, thus the representation of the SLAM problem stays very sparse, which allows for efficient optimisation. Furthermore, it becomes possible to repair past erroneous assignment of observations to landmarks, for example when the robot closes a larger loop returning to a previous location. Folkesson and Christensen pioneered these methods in their paper “Graphical SLAM - a self-correcting map” [8]. Using a non-linear optimisation engine they found the best assignment for the robot trajectory and the landmark positions.

Dellaert and Kaess improved their approach using sparse linear algebra to exploit the sparsity of the full SLAM problem. The least-squares solution to a linear approximation of the measurements is found repeatedly by their
related work

algorithm causing rapid convergence to the solution. Their results show that paradoxically by keeping the extra information about its history the performance actually improves. This method is expanded on by Kaess et al. [16, 14, 15] to allow for incremental updates to the factor graph making online operations, e.g. on a mobile robot, possible. In my mapping implementation, I use their Georgia Tech Smoothing and Mapping Library (GTSAM) [2] for graph optimisation.

Others focused on developing new types of features to be used in conjunction with the algorithms that were being developed. For example research went into extending the SLAM problem to account for 3 dimensional space, allowing for semantic mapping and so forth. Weingarten and Siegwart [30, 31] explored the use of (infinite) planar features within an EKF-based SLAM framework making use of the symmetries and perturbation model (SPmodel). Using a rotating 2D laser-range finder the features were computed and then used as landmarks. For solving the SLAM only the normals were used, while the extent of the features was tracked by an alpha shape which was extended incrementally as new portions of the plane became visible.

Later Trevor et al. expanded on this methodology by employing a graph-based SLAM approach using iSAM2 [15] instead as this apparently improves the accuracy of mapping the extent of the planes. In addition, they utilised different sensors as input, allowing them to use both planar and line features by combining data from a fixed 2D laser scanner and a 3D camera.

In this thesis I will use a similar approach using different kinds of factors for planes. Not using line features from a laser scanner in my implementation cause problems if the 3D camera does not detect a sufficient number of planes due to issues with its short detection range, i.e. issues that Trevor et al. managed to compensated for in their work because of the longer range of their laser scanners. However, my work could easily be extended by such a feature. This has not been done yet because I decided to focus on comparing different plane factors.

Another approach to using higher-level features in SLAM is proposed by Gee et al. In their paper [9] they expanded the SLAM problem to incorporate the detection of lines, planes and so forth. Initially only points are inserted into the map by the sensors. Next, if their SLAM algorithm is sufficiently confident that a set of points fit a line or even a plane, these higher-level features are added. The points themselves are collapsed to 1D and 2D points on the new feature respectively or even removed entirely, achieving a reduction in redundancy of the map. This has the advantage of being able to detect planes late by observing portions of them from different angles. However, this late detection cannot guarantee that the detected planes are also physical planes in reality.
All of these methods have in common that they only use the normal of the plane as factor in solving the SLAM. Other aspects of planar features have been used as well. Nguyen et al. [21] introduced a method connecting different planes with each other, in addition to the commonly used relative measurement of landmarks – the planes – by the robot. They argue that – particularly in man-made environments – there are many orthogonal surfaces. That is why they introduce orthogonality as constraint between planar features, even going so far as to disregard all planes that are not parallel or perpendicular to each other and only mapping those that align. According to them this will still cover the main structure of most indoor environments.

While studying a mapping only problem – wanting to achieve interactive 3D reconstruction by using a hand-held 3D camera – Taguchi et al. evaluated the combined use of point and plane primitives for constructing a 3D map. In their paper [27] they employ an interesting technique to compare an observation with a plane in the map. Instead of using the plane normal, they compute the distance of a randomized subset of the inlier points – points on the detected plane – from the plane in the map.

Confronted with a breakthrough in real-time dense SLAM with KinectFusion algorithm [20] that is able to generate dense volumetric surface maps, Salas-Moreno et al. argue that semantic mappers should aim for similar dense map generation explaining the world in pixel detail. To this end they expand upon the idea of Keller et al. [18] and use surfels for their map. They distinguish two type of surfels, planar and non-planar ones. The planar surfels match a planar surface, thus they can be updated as one and can share a normal among other properties. Whereas the non-planar surfels have to be updated individually, appearing especially in regions of high curvature, they just have too little in common with their neighbours. While being able to explain a lot of pixels using surfels, they have to keep them all stored in the machine’s memory and even with their compression algorithm this might constitute several megabytes of data.

In 2014 Trevor et al. tried to introduce OmniMapper [29] as a standard modular framework for GraphSLAM using iSAM2. While this paper does not make direct use of this framework, my implementation has to some extent been inspired by it. For example the way robot pose measurements are added borrows from similar concepts. However, I expanded their modular design to include robot pose measurements, too, whereby extending my method to work with multi-agent scenarios can more easily be done, which was part of the initial design idea.
3 Background

In order to generate a map of a robot’s environment, its exact location is needed, so that observations can be placed in the right point on the map.

However, the exact absolute location of a mobile robot is rarely available. While an estimate of the absolute location can be measured by using GPS or Motion Capture System for example, these systems require coverage. Particularly with regard to scenarios where plane maps are useful, such as indoor corridors, they become completely impractical to use.

On the other hand if an accurate map were available the exact location could easily be calculated. Not only is providing a static map somewhat problematic due to a changing surroundings, it also makes little sense, given that map creation is the goal. Furthermore, there are settings where the layout is inherently unknown, hence exploration is required.

A solution could be found if it were possible to tackle both problems at the same time. This poses the SLAM problem.

3.1 SLAM – Simultaneous Localization and Mapping

Initially this may appear as a chicken-and-egg problem, but it can be solved by approximation.

Since measurements introduce uncertainties, one usually uses a probabilistic definition for the SLAM problem. The goal is to compute the agents’ locations $x_i \in X$ and the landmarks $l_j \in L$ given the sensor observations $z_k \in Z$ and control outputs $u_i \in U$ over discrete time $i \in 1...t$. The resulting objective

$$X, L = \arg \max_{X, L} P(X, L | Z, U)$$  

(3.1)
3 Background

can be solved given some kind of motion model $P(x_i | x_{i-1}, u_i)$ and observation model $P(z_k | x_{i(k)}, l_{j(k)})$. There are several different types of well known algorithms:

- Unscented/Extended Kalman Filters
- Particle Filters
- Expectation-Maximization-Filters
- Graph-based approaches, see section 3.2

They can generally be split into two categories as the SLAM problem is either tackled by filtering or smoothing.

3.1.1 Filtering

*Filtering* is only concerned with the current location of the robot and the current map. Thus one optimizes for

$$x_t, L_t = \arg \max_{x_t, L_t} P(x_t, L_t | Z_t, u_t, x_{t-1}, L_{t-1}) \tag{3.2}$$

as the objective. The past is marginalized away. While this is fine in linear settings, in the more common non-linear case and its linearisation, doing so results in new problems. “[…] [T]he potentially wrong linearisation points are ‘baked into’ the covariance matrix […]” [3, p. 17]. This can even cause the solution to diverge from the true solution as shown by Julier and Uhlmann [13]. Furthermore, the matrix becomes eventually dense, thus filtering will be slow to compute after a few steps.

3.1.2 Smoothing

*Smoothing* looks at the entire robot trajectory up to the current time as well as the entire set of landmarks. Thus, one optimizes for

$$x_{1:t}, L = \arg \max_{X, L} P(x_{1:t}, L | Z, U, x_0) \tag{3.3}$$

as the objective. This is optimally estimating the maximum a posteriori (MAP) of the all unknown values. “Smoothing” is also known as “bundle adjustment” or “structure from motion” in other fields.
This has several advantages over filtering. For one, since the entire history is kept, one never commits to a linearisation point. Thus, errors can be corrected at a later point in time by re-linearising the entire graph or a part thereof. Furthermore, the information matrix stays sparse, thus inference can be implemented using sparse linear algebra (see √SAM [3]), which makes the computation in some situations even faster than filtering despite the need of compute more variables. It is even possible to solve this incrementally as done with algorithms like iSAM (see [16, 14]) and iSAM2 (see [15]), which expand upon √SAM.

3.2 Graph SLAM

The approach to SLAM used in this thesis is Graph SLAM (with smoothing). The general idea behind it is to represent the objective (3.1) as a factor graph.

Formally, a factor graph is a bipartite graph $G = (\mathcal{F}, \Theta, \mathcal{E})$ with two node types, factor nodes $f_i \in \mathcal{F}$ and variable nodes $\theta_i \in \Theta$. The edges $e_{ij} \in \mathcal{E}$ always connect a factor node and a variable node. Each factor $f_i$ is a function in the variables $\Theta_i = \{ \theta_j | \exists e_{ij} \}$ adjacent to it. The whole graph $G$ thereby represents the factorization

$$f(\Theta) = \prod_i f_i(\Theta_i)$$

(3.4)

of a function $f(\Theta)$.

The objective (3.1) can be factorized as

$$P(X, L | Z, U) \propto P(X, L, Z, U) = P(x_0) \prod_{i=1}^M P(x_i | x_{i-1}, u_i) \prod_{k=1}^K P(z_k | x_{i(k)}, l_{j(k)})$$

(3.5)

using the motion model, the observation model and a prior for the robot pose $P(x_0)$.

With the landmarks $l_j$ and robot poses $x_i$ as variables $\Theta$, one can reformulate the objective

$$\arg \max_{X, L} P(X, L | Z, U) = \arg \max_{\Theta} f(\Theta)$$

(3.6)

as factor graph and therefore has a unary prior factor and binary factors for both motion and observation model. An example of such a graph can be seen in figure 1 on page 13. This means that, at least for the optimization part, both measurements and odometry data can be treated the same, they are simply factors.
3 Background

Henceforth, Gaussian models will be assumed

$$f_i(\Theta_i) \propto \exp(-\frac{1}{2} \| h_i(\Theta_i) - z_i \|_{\Sigma_i}^2)$$

(3.7)

as is standard in the SLAM literature. Hereby $h_i(\cdot)$ denotes a non-linear function that predicts the measurement of $x_i$ given the current value of variables $\Theta_i$ connected to the factor, $z_i$ is the actual measurement and $\| e \|_{\Sigma_i}^2 = e^T \Sigma e$ is the squared Mahalanobis distance with covariance matrix $\Sigma$. With this assumption the objective (3.6) corresponds to

$$\arg \max_{\Theta} f(\Theta) = \arg \min_{\Theta} \left( -\log f(\Theta) \right) = \arg \min_{\Theta} \frac{1}{2} \sum_i \| h_i(\Theta_i) - z_i \|_{\Sigma_i}^2$$

(3.8)

which is a non-linear least-squares criterion. In practice one typically considers only linearised versions of this and uses non-linear optimization methods such as Gauss–Newton iterations or the Levenberg–Marquardt algorithm to approach the minimum with successive solving of linear approximations. Therefore, the Jacobian matrix of $h_i(\Theta_i) - z_i$ with regard to $\Theta_i$ should be readily available.

While general solvers for non-linear least-squares criteria could be used, more specialized solvers can be significantly faster and memory efficient, by utilizing additional knowledge of the problem structure. For SLAM performance is especially important, because the solution needs to be computed in real-time, at least several times a minute. Further, the number of factors and hence the dimensionality of the problem can increase significantly over time and will thereby slow down the solvers, consuming ever more memory.

In practice one robot pose is merely connected to the next pose and the few landmarks observed from there. The landmarks are generally not even directly connected, because a relation between them is not easily measured. The resulting factor graph thus has a fairly sparse covariance/information matrix, as can been seen in figure 1. This sparsity can be utilized for efficient optimization, as is done with ISAM2.

3.3 SLAM Initialization

At the beginning of the SLAM computations neither the map nor the position of the robot is known. In this “Kidnapped Robot” scenario the initial position of the robot can simply be used as the origin of the coordinate system of the map. This method however only works for a single robot, since in a cooperative
3.4 SLAM Data Association Problem

In landmark-based SLAM (e.g. with planes) the sensor data must be preprocessed before they can be used as an observation in one of the SLAM-algorithms (see section 3.1). First the features that make up a landmark must be extracted from the sensor data, for example planes extracted from 3D-pointclouds, which
are returned by depth sensors like the Kinect. This thesis will assume that such an algorithm (e.g. RANSAC [7]) exists and is given.

Second these features must be identified with the landmarks already on the map. If a feature is observed that is not on the map, it has to be added to it as a landmark. If the map contains a landmark that is not observed, but should have been according to position and sensor field of view, it should be removed at some point to reflect a change in the real world.
4 Theoretical Approach

In order to do PlaneSLAM, we obviously need:

1. Internal representations of the robots poses and the various planes.

While these representations can be used directly to compute the measurement functions, the Jacobians cannot be calculated with respect to these complex objects. Therefore, we need a coordinate system to express the local delta changes of the variables for optimisation. Furthermore, such a system has to allow for updating the internal representation with a local delta, similar to the gradient being added to the independent variable when optimising a function with gradient descent. Hence, we also need:

2. A coordinate system for the vector $\Theta_{R_1}$ for robot poses.

3. A coordinate system for the vector $\Theta_{P_1}$ for plane landmarks.

In addition, each type of factor used in the graph for PlaneSLAM will have to be defines, so that iSAM2 can optimise the values. For this, measurement functions and their respective Jacobians need to be derived, or as the case may be, defined for each type. Hereby the error occurring between prediction and measurement can be stated as $e(\cdot, z) = h(\cdot) - z$. It must be ensured that this error function can be express with a real-vector-valued codomain, e.g. $e: D \rightarrow \mathbb{R}^n$. In detail that means the following functions are needed:

4. The measurement function $h([\Theta_{R_1}, \Theta_{R_2}])$ for the movement between pose samples of a robot as well as its Jacobians in $\Theta_{R_1}$ and $\Theta_{R_2}$. This corresponds to the $\blacksquare$-shaped factors in the figure 1.

5. The measurement functions $h([\Theta_{R_1}, \Theta_{P_1}])$ for each of the planes measured from a robot pose $\Theta_{R}$ individually as well as its Jacobians in $\Theta_{R}$ and $\Theta_{P}$. This corresponds to the $\blacklozenge$-shaped factors in the figure 1.

6. The measurement function $h([\Theta_{R_0}])$ for factor representing the prior on the first robot pose and its Jacobian in $\Theta_{R_0}$. This corresponds to the $\blacklozenge$-shaped factors in the figure 1.
4 Theoretical Approach

4.1 Representation

4.1.1 Robot poses

In this thesis we assume robots to be without movable joints. Moving parts like wheels are of no consequence to the SLAM, because they do not change the observation aside from moving the robot. Therefore, the robot as a whole can be viewed as rigid body that somehow moves through space.

Hence, the entire robot pose can be represented as a single transform $T = [R, t] \in SE(3)$, with the rotation $R \in SO(3)$ and the translation $t \in \mathbb{R}^3$ from world origin to a fix point on the robot. $T$ can be expressed as

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \quad \text{with } R \in \mathbb{R}^{3 \times 3}, \det R = 1, \ t \in \mathbb{R}^3$$

(4.1)

using homogeneous coordinates, which is useful for computing the composition $\otimes$ of transformation as this can be done by simple matrix multiplication $T_1 \otimes T_2 := T_1 \cdot T_2$.

4.1.2 Plane Landmarks

In mathematics planes in 3D-space are generally represented by the general form of the equation of a plane

$$ax + by + cz + d = 0$$

(4.2)

or by the equivalent point-normal form

$$\langle n | r - r_0 \rangle = 0 \iff a(x - x_0) + b(y - y_0) + c(z - z_0) = 0$$

(4.3)

with the plane normal vector $n = [a, b, c]^T$, the minimal distance to the origin $d = -(ax_0 + by_0 + cz_0)$ and a point on the plane $r_0 = [x_0, y_0, z_0]^T$. A point $r = [x, y, z]^T$ is on the plane if the equations hold.

Neither of these forms is suited for this purpose however. Mathematically speaking a plane is a flat two-dimensional surface that extends infinitely far. In reality planes are always finite. For a map of an existing environment these finite boundaries must be modelled, thus neither (4.2) nor (4.3) can be used to model the landmarks.
4.1 Representation

The boundary of a plane can be modelled mathematically using its convex hull. Requiring the hull to be convex does not significantly impact the validity of our model, since a concave plane can be split into convex planes. But it makes computations much simpler and is also what is returned by most plane segmentation algorithms including those used in this project.

The convex hull is an ordered set of points for which a frame of reference is needed. Initially the position of the robot’s sensor observing the plane could be seen as reasonable reference frame. As a plane can be surveyed from different locations at multiple times, however, using any one of them as reference would be quite arbitrary. Moreover, using the sensor as reference point is futile when it comes to exporting the map without dependence on the observations made.

Therefore, the sensor is not suited as reference frame. The next logical candidate for a practical reference frame is the origin point of the map/world. However, the sensor location at the time of observation and therefore the location of the plane is uncertain regarding the origin. In addition, the measurement itself induces uncertainty. So the origin cannot be used either.

The best option is a local coordinate system on the plane. We know that the points will always be on the plane. Therefore, a 2D coordinate system on the surface of the plane will suffice. The benefit of such a reference frame for the points is that they move with the plane and therefore stay on the plane in the right place.

Both the general form (4.2) and the point-normal (4.3) form do not contain sufficient information to define such a coordinate system. While (4.3) offers \( r_0 \) as point of origin for the system, the direction of the axes is only restricted to be orthogonal to the normal vector \( n \) and thus one degree of freedom is still left undefined.

Another way to define planes would be parametrically

\[
    r(s, t) = r_0 + s \cdot v + t \cdot w
\]

with a point on the plane \( r_0 \) as before and two linearly independent vectors on the plane. The plane is then defined as the sum of points \( r(s, t) \forall s, t \in \mathbb{R} \). While this definition has a coordinate system with \((s, t)\)-points, the system is generally non-Cartesian, which would be desirable. Furthermore, applying a transformation \( T = (R, t) \in SE(3) \) to a plane given in this form is somewhat inconvenient when using

\[
    r'_0 = T \cdot r_0, \quad v' = R \cdot v, \quad w' = R \cdot w
\]
4 Theoretical Approach

as all vectors \( r_0, v \) and \( w \) must be transformed individually. Finding a local coordinate system for the delta changes and transforming would be complicated.

In this thesis we have therefore chosen a different approach and define the plane by the coordinate system used for the convex hull. This coordinate system is given by a transform \( T_p \in \text{SE}(3) \) applied to the map coordinate system. The plane is then defined as the \( xy \)-plane of the coordinate system with the \( z \)-axis is the normal vector. The points of the convex hull have \( z = 0 \) and thus all lie on the plane. So the tuple

\[
\left( T_p, (p_0, \ldots, p_n) \right), \quad T_p \in \text{SE}(3), \forall i : p_i \in \mathbb{R}^2, \forall j \neq k : p_j \neq p_k, (p_0, \ldots, p_n, p_0) \text{ convex } n\text{-sided polygon}
\]

defines a plane landmark. Transforming the plane can now simply be done by changing the transforms \( T_p' = T \cdot T_p \). Writing the transform \( T_p \) for each of the discussed forms in homogeneous coordinates one can see the conversions between the different forms.

\[
T_p^{(4.4)} = \begin{bmatrix}
\frac{v}{\|v\|} & \frac{w}{\|w\|} & \frac{v \times w}{\|v \times w\|} & r_0 \\
0 & 0 & 0 & 1
\end{bmatrix}, \quad T_p^{(4.3)} = \begin{bmatrix}
p & q & n & r_0 \\
0 & 0 & 0 & 1
\end{bmatrix}, \quad T_p^{(4.2)} = \begin{bmatrix}
p & q & b & y_0 \\
0 & 0 & c & z_0
\end{bmatrix}
\]

\[
\tilde{w} = w - \langle v \mid w \rangle v
\]

\[
\exists p, q \in \mathbb{R}^3 : p \times q = n \wedge \langle p \mid q \rangle = 0 \wedge \|p\| = \|q\| = 1 \leadsto 1 \text{ DoF}
\]

\[
\exists x_0, y_0, z_0 \in \mathbb{R} : ax_0 + by_0 + cz_0 = -d \leadsto 2 \text{ DoF}
\]

The transforms obtained from the forms (4.2) and (4.3) have 3 and 1 Degree of Freedom, that means they both carry less information than the other ones, as stated before. While the parametric form (4.4) contains the same information as (4.6) with regard to the transform, it is more cryptic and brings up the issue with the convex hull discussed before. So the form (4.6) will be used from here on. Although it should be possible to use the parametric form as well, if desired.

For convenience lets define \( \mathbb{PL} \) as the space of bounded planes in three dimensions, i.e. the space of plane landmarks.

\[
\mathbb{PL} = \text{SE}(3) \times \left( \mathbb{R}^2 \right)^n, \quad \left( T_p, \text{hull} = (p_0, \ldots, p_n) \right)^{(4.6)} \in \mathbb{PL}
\]

18
4.2 Local coordinates for robot poses

As defined in section 4.1.1 a robot pose is represented by a transform $T \in \mathbb{SE}(3)$. If we have a function $f : \mathbb{SE}(3) \to \mathbb{R}^m$, we cannot derive the Jacobian of it, since $\mathbb{SE}(3)$ is not a vector space. Therefore, we need to find a local representation that is a vector space, so that we can compute the Jacobian for $f(T)$. This is necessary because the measurement functions will be of similar form as $f$.

All transforms have two components, i.e. the rotation $R \in \mathbb{SO}(3)$ and the translation $t \in \mathbb{R}^3$, that can be viewed independently. The translation $t$ is already in a vector space, the standard $\mathbb{R}^3$. Therefore, it can be directly used for our local representation.

The rotation group $\mathbb{SO}(3)$ however is not a vector space, but a differentiable manifold. Therefore, we have to use the tangent space of this manifold for derivations. In fact $\mathbb{SO}(3)$ is a Lie group and hence there is a Lie algebra $\mathfrak{so}(3)$ associated with it. This Lie algebra consists of all skew-symmetric $3 \times 3$-matrices. While the elements $A \in \mathfrak{so}(3)$ themselves are no rotations, an infinitesimal rotation matrix has the form $1 + A \cdot d\phi$ with an infinitesimal $d\phi \to 0$, thus $A$ is the derivative of such an infinitesimal rotation. As these are rotations close to the identity rotation, the space of derivative $\mathfrak{so}(3)$ is the tangent space of the manifold $\mathbb{SO}(3)$ at the identity element.

Although the Lie algebra is matrix space, and consequently it is still not possible to compute the Jacobian with respect to it, we can choose a basis for it such that it becomes a vector space. Since $\mathbb{SO}(3)$ and therefore $\mathfrak{so}(3)$ has the dimension $3$, we need three basis matrices:

\[
L_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \quad L_y = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \quad L_z = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}
\] (4.9)

Because Lie algebra $\mathfrak{so}(3)$ with the matrix commutator is isomorphic to the Lie algebra $\mathbb{R}^3$ under cross product, the commutation relations of the basis elements $[L_x, L_y] = L_z$, $[L_z, L_x] = L_y$, $[L_y, L_z] = L_x$ should agree with the relation under cross product for the standard unit vector of the $\mathbb{R}^3$, which is the case for this basis.

Since $\mathbb{SO}(3)$ is a matrix Lie group, the exponential map is defined as

\[
\exp : \mathfrak{so}(3) \to \mathbb{SO}(3), \quad A \mapsto e^A = \sum_{k=0}^{\infty} \frac{1}{k!} A^k
\] (4.10)
4 Theoretical Approach

the standard matrix exponential series. The exponential map is just surjective. But still a logarithm map \( \log : \mathrm{SO}(3) \to \mathfrak{so}(3), R \mapsto A \) can be found. For example by using the Rodrigues’ formula one gets

\[
R = e^A = \left( I + \frac{1}{2} A^2 \sin \frac{1}{2} ||A|| \right) + \frac{1}{2} (A \sin ||A||)
\]

\[
= \frac{1}{2} \left( R + R^T \right) + \frac{1}{2} \left( R - R^T \right)
\]

\[
\Rightarrow \log : \mathrm{SO}(3) \to \mathfrak{so}(3), R \mapsto A = \frac{R - R^T}{||R - R^T||} \arcsin \frac{1}{2} ||R - R^T||
\]

This logarithm-map maps \( R \) to the smallest element \( A \) of \( \mathfrak{so}(3) \), which fulfils \( \exp A = R \). This choice is necessary, because \( \exp \) is subjective and therefore multiple solutions could exist.

With these equations we can define a 6-dimensional vector \([r_x, r_y, r_z, t_x, t_y, t_z]^T \in \mathbb{R}^6\) as local representation for a transform \( T \in \mathfrak{se}(3) \) to be used for deriving the Jacobian. Therefore, we get the exponential map

\[
\exp : \mathbb{R}^6 \to \mathfrak{se}(3), \begin{bmatrix} r_x \\ r_y \\ r_z \\ t_x \\ t_y \\ t_z \end{bmatrix} \mapsto \begin{bmatrix} \exp(r_x \cdot L_x + r_y \cdot L_y + r_z \cdot L_z) \\ t_x \\ t_y \\ t_z \end{bmatrix}
\]

and the logarithm map

\[
\log : \mathfrak{se}(3) \to \mathbb{R}^6, \begin{bmatrix} R \\ t \end{bmatrix} \mapsto \begin{bmatrix} r_x \\ r_y \\ r_z \\ t_x \\ t_y \\ t_z \end{bmatrix} \]

\[
\text{with } \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} := t, \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} := \log R
\]

for conversion between the two forms. Further we can make use of an infinitesimal transform in homogeneous coordinates

\[
1 + \text{Ad}T = \begin{bmatrix} 1 & -dr_z & dr_y & dt_x \\ dr_z & 1 & -dr_x & dt_y \\ -dr_y & dr_x & 1 & dt_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
\]
4.2 Local coordinates for robot poses

to derive the Jacobian of the hypothetical function $f : \SE(3) \to \mathbb{R}^m$ as

$$\mathcal{J}_f(T_0) = \frac{\partial f}{\partial T}(T_0) = \left. \frac{\partial}{\partial \delta \left[ \begin{array}{c} \delta r_x, \delta r_y, \delta r_z, \delta t_x, \delta t_y, \delta t_z \end{array} \right] \right|_{\delta T=0} \in \mathbb{R}^{m \times 6} \quad (4.15)$$

In addition to the Jacobian we also need to apply a local delta in this tangent space to update a pose, for example to apply an iteration step in the non-linear solver, which uses the Jacobian to compute the update. For this we define the retract-operation $\oplus : \SE(3) \times \se(3) \to \SE(3), (V, \Delta V) \mapsto V'$ which applies the local delta $\Delta V$ to $V$. To compute $V \oplus dV$ the transform $V$ is composed with the infinitesimal transform $dV$ (see equation (4.14)) corresponds to,

$$V \oplus dV := V \cdot (1 + AdV) \quad (4.16)$$

however this only works for an $dV$ close to zero (or to be exact infinitesimally close to zero). In order to compute $V \oplus \Delta V$ for any $\Delta V$ the corresponding transform $\in \SE(3)$ can be used. This yields a more exact version of retract as

$$V \oplus \Delta V := V \otimes \exp \Delta V \quad (4.17)$$

however, $\Delta V$ is an element of the tangent space at the identity, therefore it contains only first-order information of the update, so only this can be applied. The exponential map $\exp$ is fairly expensive to compute. Since we only do linear update in the first place, an approximation would be more appropriate. In this thesis we often use

$$V \oplus \Delta V \approx \left( R_T \otimes \exp(r_x \cdot L_x + r_y \cdot L_y + r_z \cdot L_z), t_T + R_T [t_x, t_y, t_z]^T \right) \quad (4.18)$$

as approximation with $V = (R_T, t_T)$ and $\Delta V = [r_x, r_y, r_z, t_x, t_y, t_z]^T$, this updates the rotation exactly and does a first-order approximation for the update of the translation.

Further an “inverse” of this operation is useful, that is an operation that computes the local delta between two poses. For this we define the local coordinate-operation $\ominus : \SE(3) \times \SE(3) \to \se(3), (U, V) \mapsto \Delta(U - V)$. The exact version can be easily derived from (4.17) using

$$V' = V \oplus \Delta V = V \otimes \exp \Delta V \Leftrightarrow V^{-1} \otimes V' = \exp \Delta V \Leftrightarrow \log \left( V^{-1} \otimes V' \right) = \Delta V$$

Thus we get

$$U \ominus V := \log \left( V^{-1} \otimes U \right) \quad (4.19)$$

21
4 Theoretical Approach

for the local coordinate-operation which comes at similar computation cost as retract. Because of this and for consistency, we can derive an approximation from (4.18) and get

\[ U \ominus V \approx \begin{bmatrix} r_x, r_y, r_z, R^{-1}_V (t_U - T_V) \end{bmatrix}^T \]  

(4.20)

with \[ \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} \]  

(4.9) \[ := \log (R^{-1}_V R_U) \]

as the approximate inverse. This form of local coordinate is used in this thesis whenever the approximate retract is used.

4.3 Local coordinates for plane landmarks

Using the tuple \((T, hull)\) with \(T \in SE(3)\) and the \(n\)-length point sequence \(hull \in (\mathbb{R}^2)^n\) to represent a plane landmark (as introduced in section 4.1.2) leads to problems similar to those involving robot poses. For a function \(f : PL \to \mathbb{R}^m\) the Jacobian cannot be derived, since again \(PL\) is not a linear space. In fact, we have not even defined any operation on it nor does it make much sense to do so. You cannot multiply or add planes together. \(PL\) is a topological space, but not even a manifold, since the hull prevents it from being locally Euclidean. It is therefore not a differentiable manifold nor can we attach a differentiable structure to it. Hence, we cannot do calculus on \(PL\), yet we need to, so that we can use the landmarks in the Factor Graph for SLAM with a non-linear solver. However, while the whole structure of a plane landmark eludes us, we can apply calculus to a part of it, if we ignore the hull. We can do this for optimisation, but that means we need to update the hull elsewhere.

Since there is no trivial way to assign a tangent space to \(PL\), we will look at multiple variants.

4.3.1 Lie algebra se(3) of plane transform

If we drop the hull from the 2-tuple this \(PL\), only the transform \(T_p \in SE(3)\) remains. So the simplest way to get the Jacobian would be to treat the transform just like the one used in the case of robot poses. This allows us to reuse most
findings from section 4.2 with some adjustment. So it follows that the Jacobian of the hypothetical function \( f : \mathbb{P}L \rightarrow \mathbb{R}^m \) can be defined as

\[
J^\mathfrak{se}_f(P_0) = \left. \frac{\partial f}{\partial \mathfrak{se} P} (P_0) \right|_{dT=0} = \partial \left[ f_{(1...m)} \left( \left[ (I + A_d) \otimes T_{p,0}, hull_0 \right] \right) \right]_{dT=0} \in \mathbb{R}^{m \times 6}
\]

with \( P \in \mathbb{P}L \) and \((T_{p,0}, hull_0) = P_0 \in \mathbb{P}L\). We henceforth use \( \mathfrak{se} \) to mark the use of the tangent space we get from the Lie algebra of the transform \( \in \mathbb{SE}(3) \).

The dimension of this tangent space is six, which is considerably higher than the three degrees of freedom an unbounded plane in 3D has. This is not necessarily bad, as it makes local representation more expressive, which is why the \( \mathbb{P}L \) representation has six dimensions, not counting the hull, in the first place. However, it also doubles the size of the matrices involved in computing the factor graph optima, which has a significant performance impact.

In addition to the Jacobian we also need to compute the local delta between two planes (or the plane and itself in the next iteration) in this tangent space and be able to apply a local update. For this we can adjust the \textit{retract} and the \textit{local coordinate} operation from (4.17)/(4.18) and (4.19)/(4.20). Here only the exact version is shown, the approximation can be adjusted in a similar fashion.

This yields for the update of the plane \( P = (T_{p}, hull) \in \mathbb{P}L \) with the local delta \( \Delta P \in \mathbb{R}^6 \)

\[
P \oplus \Delta P := \left( T_p \otimes \exp \Delta P , hull \right)
\]

and for the local delta between the two planes \( P = (T_{p}, hull) \in \mathbb{P}L \) and \( Q = (T_{q}, hull) \in \mathbb{P}L \)

\[
Q \ominus P := \log \left( T_{p}^{-1} \otimes T_{q} \right)
\]

with the exponential and logarithm map from (4.12) and (4.13). The small \( \mathfrak{se} \) over the operators again marks the use of this tangent space.

### 4.3.2 Parameters of general plane equation

Alternatively we could revisit the general form of the plane equation

\[
ax + by + cz + d = 0
\]

from section 4.1.2 on page 16 and use the variables \( a, b, c \) and \( d \) for our tangent space, similar to what Trevor et al. [28] did in their work. As indicator
4 Theoretical Approach

for this tangent space we will use \( abcd \). An object in this space would be \( [a, b, c, d]^T \in \mathbb{R}^4 \).

The first step is to derive a logarithm map and the exponential map. The log-map is quite simple, using (4.7) on page 18 we get

\[
\log_{abcd} : \mathbb{P}L \rightarrow \mathbb{R}^4, (T_p, hull) \mapsto [a, b, c, d]^T \tag{4.24}
\]

with

\[
\begin{bmatrix}
* & * & a & x_0 \\
* & * & b & y_0 \\
* & * & c & z_0 \\
0 & 0 & 0 & 1
\end{bmatrix} = T_p, d = -(ax_0 + by_0 + cz_0)
\]

which is a surjective map, for every plane there is just one way to represent it in \( abcd \). But all bounded planes that lie in the same the infinite plane, regardless of their position or rotation, are mapped to the same point.

This forces us to chose the “right” plane, when defining the exponential map. As stated under section 4.1.2 on page 18, there are three degrees of freedom. For each one of them you would need to make the “right” choice, which is quite impossible. The information needed to reconstruct the transform is just not there. This comes not unexpectedly as we reduce the dimensionality from six to three with the logarithm map. \( a, b, c, d \) are just three independent dimensions since the transform \( T_p \) has the property \( \| [a, b, c]^T \| = 1 \Rightarrow c = \sqrt{1 - a^2 - b^2} \).

So there is no exponential map, but it is not actually required, as long as we can define the Jacobian \( J_f^{abcd} \), the retract- and the local coordinate-operation. The local coordinate-operation is again simple, as it is defined by the log-map, thus we get

\[
Q^{abcd} \oplus P := \log_{abcd} \left( T_p^{-1} \otimes T_q \right) \tag{4.25}
\]

Assuming that we have the whole transform to compute the Jacobian we get

\[
J_f^{abcd} = \left. \frac{\partial f}{\partial abcd} \right|_{P_0} = \left. \frac{\partial}{\partial [a, b, c, d]} \left( f_{(1...m)} \left( (\tilde{T}, hull_0) \right) \right) \right|_{[a, b, c, d] = \log_{abcd} P_0} \in \mathbb{R}^{m \times 4} \tag{4.26}
\]

with

\[
\tilde{T} = \begin{bmatrix}
T_{p,(1,1)} & T_{p,(1,2)} & a & \xi \\
T_{p,(2,1)} & T_{p,(2,2)} & b & T_{p,(2,4)} \\
T_{p,(3,1)} & T_{p,(3,2)} & c & T_{p,(3,4)} \\
0 & 0 & 0 & 1
\end{bmatrix}, \xi = -\frac{bT_{p,(2,4)} + cT_{p,(3,4)} + d}{a}
\]

with \( P \in \mathbb{P}L \) and \( (T_p, hull_0) = P_0 \in \mathbb{P}L \).
4.3 Local coordinates for plane landmarks

The retract-operation $P \oplus^{abcd} \Delta P$ would be defined by the missing exponential map. However, since we know that $\Delta P$ is just a “small” change/update to $P$, we can possibly find the result as a transformation of $P$. The translation $t_P$ of the plane transform $T_P$ is best updated by moving $d$ along the normal vector of $\Delta P$. While there are infinitely many possible points we could move to, this move is the short distance, which is reasonable for a small change $\Delta P$. Therefore we get $t_P' = t_p + d \cdot [a, b, c]^T$. The rotation $R_P$ of the plane is not as straightforward.

4.3.3 Range and 2 angles

Instead of choosing a tangent space as a well-known plane representation, we can pick one that lets us make this last step of finding the retract-operation easy. The challenging part with the $abcd$-tangent space was to find the rotation induced by $a$, $b$ and $c$. So let us have two variables instead that represent the rotation directly. That is one for each degree of freedom of rotating a plane around a fix point on the plane, for example the centre point.

To derive such a representation, we start with a plane $P$ to which we will apply a local delta in our new tangent space. Its plane transform $T_P$ will become our base coordinate system, as can be seen in figure 2 on the following page. In this coordinate system the normal vector of $P$ is a priori $[0, 0, 1]^T$, as it is defined to be the $z$-axis. This unit-length normal vector can be rotated to match any other unit-length vector by one rotation around the $x$-axis and one rotation around the $y$-axis. Therefore, the rotation part of plane movement can be handled with the two angles $\varphi$ and $\theta$. Translations can be handled just like before with a single distance variable $r$ that represents the distance traversed along the normal vector towards the new plane.

Combining the standard rotation matrices around the $x$- and $y$-axis with a translation along the normal vector, e.g. translation along the $z$-axis of the
Figure 2: Illustrates the application of the local update $\Delta P = [r, \theta, \varphi]$ to the $xy$-plane, which yields the $x'y'$-plane with the $z'$ axis as normal. In other words the computation of $P' = P \oplus^{r\theta\varphi} \Delta P$. The $r\theta\varphi$-values in the figure are $r = 1$, $\theta = 30^\circ$ and $\varphi = -30^\circ$.

initial coordinate system, as visualized in figure 2, we get

$$T_{P \rightarrow P'} = \left[ \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) & 0 \\ 0 & \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] \cdot \left[ \begin{array}{cccc} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] = \left[ \begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{array} \right]$$

(4.27)

as transform to the new plane $P'$ in the reference frame of $P$.

With this we can define a new tangent space. Henceforth, $r\theta\varphi$ will be used to indicated the use of this new tangent space. The simplest definition is the retract-operation, as this operation is already shown in figure 2 as it is part of the derivation. In terms of equations we get

$$P \oplus^{r\theta\varphi} \Delta P = (T_{Map \rightarrow P'}, hull) = (T_{Map \rightarrow P} \otimes T_{P \rightarrow P'}(\Delta P), hull) = (T_p \otimes T_{P \rightarrow P'}(\Delta P), hull)$$

(4.28)
4.3 Local coordinates for plane landmarks

which – in turn – by comparison with the initial definition of \( \text{retract} \) in the equation (4.17) on page 21 gets us the exponential map of \( r \theta \varphi \):

\[
\exp : \mathbb{R}^3 \rightarrow \mathbb{P}L, \begin{bmatrix} r \\ \varphi \\ \theta \end{bmatrix} \mapsto \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ \sin(\varphi) \cdot \sin(\theta) & \cos(\varphi) & -\sin(\varphi) \cdot \cos(\theta) \\ -\cos(\varphi) \cdot \sin(\theta) & \sin(\varphi) & \cos(\varphi) \cdot \cos(\theta) \\ 0 & 0 & 0 \\ r \cdot \sin(\theta) & 1 \end{bmatrix}, (4.29)
\]

with the transform as derived before and an empty or unassigned hull as there is no sensible way to reconstruct it, after all information about it has been dropped. Keep in mind that while the map is not surjective with respect to the whole \( \mathbb{P}L \), its image contains all planes without a hull, as they indistinguishable when rotated around the normal.

The logarithm map can be extracted from figure 2 using simple geometry as

\[
\log : \mathbb{P}L \rightarrow [0, \infty) \times (-\pi, \pi) \times \left( -\frac{\pi}{2}, \frac{\pi}{2} \right), \quad (4.30)
\]

\[
T_p = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ 1 \end{bmatrix}, \quad hull \mapsto \begin{bmatrix} r \\ \varphi \\ \theta \end{bmatrix} = \begin{bmatrix} \arg(-z_2 + iz_3) \\ \arcsin z_1 \end{bmatrix}
\]

Since the exponential map is not injective, the logarithm map it not uniquely specified. Here the solution closest to zero is chosen for the logarithm map. While the exponential map is not surjective either, this is of no consequence because the hull is dropped and without it – as stated before – there exists only one pre-image for all plane landmarks. Consequently, the logarithm is well-defined.

With exponential and logarithm map defined we can reuse the old definition of the \( \text{retract} \)- and \( \text{local coordinate-operation for the} \ r \theta \varphi \)-tangent space

\[
P^r \theta \varphi \Delta P := T_p \otimes \exp \Delta P, \ hull \] (4.31)

\[
Q^r \theta \varphi P := \log(T_{p^{-1}} \otimes T_q) \] (4.32)

The Jacobian matrix using this tangent space becomes

\[
\mathcal{J}_{f}^{r \theta \varphi} = \frac{\partial f}{\partial r \theta \varphi} (P_0) = \left[ \frac{\partial f_{(1 \ldots m)}}{\partial [r, \varphi, \theta]} \left( \exp^{r \theta \varphi} \left[ \begin{bmatrix} r \\ \varphi \\ \theta \end{bmatrix}, \ hull_0 \right] \right) \right]_{[r, \varphi, \theta] = \log^{r \theta \varphi} P_0} \in \mathbb{R}^{m \times 3} \] (4.33)
4 Theoretical Approach

As the dimension of this tangent space is only three, we save half of the space the $se(3)$ used for Jacobian. As half-size matrices lead to half as many computations, we should expect a performance gain, using $r\theta\varphi$.

4.4 Factor for time-consecutive samples of a single robot’s poses

Considering a single robot we can take a robot pose measurement whenever required. Since in graph slam we do not directly model time, we have to sample the continuous data and create a new variable for each point in time a pose is sampled. In between the samples we aggregate all odometry data in a single transform from the latest sampled pose.

Therefore, it makes no sense to add a factor between any pose samples but two consecutive ones, as longer distances factors would be nothing but an aggregate of shorter ones and additional factors slow down optimisation. Moreover, it makes no sense to take samples, when there are no other measurements taken at the same time. Pose sample variables that are only connected to odometry factors do nothing for the SLAM but to slow it down. In the end they only combine the transforms before and after the sample, which can be achieved much more efficiently by direct composition as done by the aggregation in between the samples.

Hence, we only need a measurement function that predicts the transform between two poses. Each pose is represented by a single transform (see section 4.1.1 on page 16), so we can use the well-known formula to compute the transform between them, which yields the measurement function

$$h : \mathbb{SE}(3) \times \mathbb{SE}(3) \rightarrow \mathbb{SE}(3), \left( \Theta_{R(i-1)}, \Theta_{R_i} \right) \mapsto \Theta_{R(i-1)}^{-1} \otimes \Theta_{R_i}$$

with $\Theta_{R_i}$ being the current estimate of the transform representing the robot pose at each of the points of time, i.e. the variable for the sample in the factor graph.

The function is not vector-valued nor is the measurement itself, they are both transforms in $\mathbb{SE}(3)$. This is fine since both prediction and observation have the same type which is required, but we need to have a vector-valued error $e = h(\cdot) - z$. Furthermore, there is not even a subtraction define on the $\mathbb{SE}(3)$ that could be used here.
4.4 Factor for time-consecutive samples of a single robot’s poses

For this we can again use the tangent space defined for transforms from section 4.2 on page 19. The error is the same as local delta between prediction and measurement which is equal to the position of the prediction in the tangent space of the measurement. For computing this we have defined the local coordinate-operator. So the error function is

\[ e : \mathbb{SE}(3) \times \mathbb{SE}(3) \times \mathbb{SE}(3) \to \mathbb{R}^6, \]

\[ \left( \Theta_{R_{i-1}}, \Theta_{R_i}, z \right) \mapsto h\left( \Theta_{R_{i-1}}, \Theta_{R_i} \right) \odot z \]

\[ = \left( \Theta_{R_{i-1}}^{-1} \odot \Theta_{R_i} \right) \odot z \]  

(4.35)

with the factor graph variables as before used for the prediction and the measurement \( z \), i.e. the transform from aggregated odometry.

Finally, we need for the factor the Jacobian matrices which are derived using the equation (4.15) on page 21.

\[ \frac{\partial h}{\partial \Theta_{R_{i-1}}} = -\text{Adj} \left\{ \left( \Theta_{R_{i-1}}^{-1} \odot \Theta_{R_i} \right)^{-1} \right\} \]

\[ \frac{\partial h}{\partial \Theta_{R_i}} = \mathbb{I}_6 \]

(4.36)

Take note of the fact that these are the Jacobians for the measurement function, not the ones for the error, which would actually correspond with the ones needed for the factor. In the context of optimisation, however, they are a sufficient approximation for the Jacobians of the factor. The issue with deriving the exact Jacobians is the local coordinate-operation in the error function which is more than a simple subtraction \( h(\cdot) - z \). We have defined it to be

\[ U \odot V := \log \left( V^{-1} \otimes U \right) \]

(4.19 revisited)

where the \( V \) is the constant \( z \) and the \( U \) the measurement function. Consequently, we are interested in the derivation with respect to the first operand \( U \). The inner derivative, e.g. the one of \( V^{-1} \otimes U \) is simple, it is merely \( V^{-1} \).

The outer derivative of the logarithm map however is not simple, for an exact derivation of it see [5]. For our propose the influence of these is limited, while it may cost a few iterations more the optimisation will still converge sufficiently quick. Hence, let us pretend the derivative of the local coordinate with respect to the first operand is \( \mathbb{I} \), which is a coarse oversimplification and far from correct, but works as an approximation. As a result we approximate \( \frac{\partial e}{\partial \Theta_{R_i}} \approx \frac{\partial h}{\partial \Theta_{R_i}} \), ergo the Jacobians for the measurement function can be used as the ones for the factor.

29
4 Theoretical Approach

4.5 Factor for plane landmarks

Whereas it was fairly straightforward to get a measurement function for robot poses, the matter is less clear with bound plane landmarks. Here we do not have a unique obvious way to measure them. For robot movement it is quite clear that we measure the movement, e.g. the transform between the poses. Also, this is what the sensors actually measure. We do not observe the planes directly, they are just clusters of points that fit a planar surface in a point cloud. The point cloud is what is actually generated by the sensors.

Therefore, there is some freedom in choosing what the plane measurement is, in the sense of what will contribute to the factor and what not. In this paper we will define two different factors to explore this freedom (see sections 4.5.3 and 4.5.4), but let us first consider some of the possibilities.

4.5.1 Available input data

The representation of the plane in the map (compare section 4.1.2) is a transform giving us a coordinate system on the plane, the normal vector and a (convex) hull that defines the bounds of the plane, according to the map. The plane “observation”, as generated by the plane segmentation algorithm, is constituted of the plane normal vector and something describing the shape of the planar surface detected. This description can be the set of inliers, that is the points the algorithm perceives to be on the plane, or the accumulation of these as a convex hull.

The segmentation algorithm used in this paper delivers a convex hull. This has the advantage of being a more compact description, thus it saves space when stored in memory and transmitting it over the network is more efficient. However, being a more abstract description some additional information contained in the set of inliers is lost. For the thesis this is without consequence since the information would not be used by any of the factors described below anyway.

4.5.2 Possible measures

Knowing the data available, the quality and usefulness of it as a measurement is to be considered before deriving factors.
4.5 Factor for plane landmarks

Plane normal direction

The direction of the plane normal vector being calculated by fitting a plane on many measurements, e.g. the inlier points. While the position of the points in a point cloud measured with a single depth sensor cannot be assumed to be truly independent, one can expect the noise affecting the direction of the normal vector to be generally much less than the noise in the position of the individual points. For a larger plane a small change in the normal direction would lead to a significant change in the position of points on the boundary assuming the centre remains fixed.

Being a fairly precise measure and being a distinctive characteristic of planes makes the normal direction a good measure to be used by the factor. By its nature measuring the normal only yields information on the orientation of the plane. Thus factors using only this measure would only constrain some of the dimensions of the local representations, for the $r\theta_\phi$-representation only 2 of 3 dimensions and for the $se$-representation 3 of 6 dimensions (compare sections 4.3.1 and 4.3.3 respectively). Yet iSAM2 – like most solvers – requires that at least some constraint is placed on all dimensions of the local representations. Therefore, further measures are required.

Convex hull

The largest portion of the plane observation is the convex hull (at least memory-wise), however, while being an important and very descriptive property of a bounded plane, it is of very limited use as a measure for a factor, not only because we have excluded it from the local representation. This is because the observation of the hull as boundary of the plane is the result of several interfering effects. Initially there is the actual boundary of the real planar surface, which by itself would be of use. However, it is reduced by the sensor’s limited field of view and occlusion by other objects. With that even a perfect sensor and plane segmentation could only observe partial planes. Thus, the convex hull might change unpredictably with each change in the location of the observation.

Any measurement function for plane landmarks is a predictor for the plane observation that would have been made if the robot had been in a given position at a given orientation and the real plane had been the same as the one in the map. However, any measure based on the convex hull inherits its unpredictability and therefore no predictor, no measurement function could possibly exist for it.
4 Theoretical Approach

Nevertheless, the lack of a measurement function does not mean we cannot define some generalised version of a measurement factor, the iSAM2-implementation actually allow for arbitrary non-linear factors, as long as they can be linearised.

For example, we know that we did observe the plane at the time when we have a factor, therefore for any robot pose and plane combination given to the factor we know that the plane must be in the field of view and is not completely occluded by any object. Here the occlusion calculations would not only be extremely expensive, but also would require the factor to depend on all objects in the map, including future ones, which cannot be done with iSAM2. This would be further complicated by a non-static observed world when, of course, the real world is dynamic. Thus, while occlusion cannot be taken into account, field of view computation though still somewhat expensive would be possible.

Let us assume there is a function $f: \mathbb{S} \mathbf{E}(3) \times \mathbb{P} \mathbb{L} \rightarrow \{0, 1\}$ that computes whether the plane is in view. This is not a continuous function, which is problematic when it comes to the required linearisation. However, in this case $f$ is piecewise-constant, either 0 or 1. Therefore, the Jacobian is zero everywhere except at the discontinuities, which makes it utterly useless for optimisation. Trying to alter the function to make it smooth with a non-zero Jacobian, at least when the plane is not in view, would not only insanely complicate the computation of the function itself, but also the Jacobian which then would still have to be derived.

Going back to a more direct use of the convex hull leads to the idea of using the intersection of the hull from the map and the observation of the hull based on current, transient solution for the robot pose computed by the SLAM. Using a binary factor by just evaluating whether the hulls intersect, leads to the same problems that occurred in the previous example.

Having said that one could maximize the area of the intersection, which would be a continuous function, for the factor. However, computing the intersection of two many-sided polygons, e.g. the hulls, is quite complex and expensive even if they are convex as it is the case here. Computing the area of the intersection polygon, which may have many sides as well, is not trivial either.

All that only constitutes the function, the Jacobian of it is then to be derived and computed, which will not be simple or cheap either. And whether a linearised version works well for optimisation is to be questioned, the whole computation seem quite non-linear. Furthermore, in real-time applications the function would need to be evaluated several hundreds times a second at the very least, which seems quite infeasible.
4.5 Factor for plane landmarks

Even if one put special effort into this to make it work, the factor would still only take effect, if the observed convex hull does not lie within the one from the map. This is however quite unlikely, as the convex hull from the map is updated every time the plane is observed in order to include the observation, i.e. the observed hull promptly merged into the hull in map. In a nutshell, maximizing the area of the intersection would be a factor that influences the graph hardly ever, but is extremely expensive to compute. Therefore, we do not consider this as an adequate idea for a factor.

Moreover, the convex hull does not appear to provide for an admissible factor. Hence, here again it will be ignored for the computation. Keep in mind we are only discussing the plane boundaries described by a convex hulls, as the same cannot be said about the set of inlier points – e.g. the alternative description of the shape of the plane – that could be returned by plane segmentation. Their use as measure is beyond the scope of this thesis, however, it will be briefly discussed in section 7.1.2 as part of the section on Future Work.

Distance

Having only determined the plane normal as measure for use by a potential factor, we are still short of a complete measurement factor by at least one constraint. The obvious choice and the only thing remaining as a result provided by the plane segmentation used in this thesis is the distance between the plane and the observer, e.g. the sensor on the robot.

There are two different distances to be considered: the short distance and the relative distance of the centre of the convex hull from the observer. Using the shortest distance leads to a short correlation between the normal and the location, whereas using the centre might seem free of that bias.

However, the independence of orientation and position when using the centre comes at the price of depending on the convex hull. Hence, the centre might move/jump around a bit, but all the while remain on the plane. This is not quite as bad as what has been described in relation to the convex hull, it can be compensated. The centre can be adjusted before creating the factor, and the translational measurement noise in any direction tangential to the plane can be increased to the point of being very large. This compensation again leads to a dependence of the distance on the normal vector.

Both of these types of distance measures find use in one of the two factors defined in this paper, each time in combination with the measure of normal direction.
4.5.3 The factor using the $xy$-plane of a transformed coordinate system

The perhaps most straightforward way to “measure” a plane – in the context of creating a measurement function – might be to assess it as it would be represented internally, e.g. by applying the same form used for the representation of plane landmarks (see section 4.1.2 on page 16). So let us recall how a plane landmark is represented as an element of $\mathbb{P}L$

$$\mathbb{P}L = \mathbb{SE}(3) \times (\mathbb{R}^2)^n \quad (T_p, \text{hull} = (p_0, \ldots, p_n)) \quad \in \mathbb{P}L \quad (4.6) \quad (4.8 \text{ revisited})$$

Previously (see section 4.5.2) it was concluded that the convex hull does not lend itself to contribute to the factor. Ignoring the hull leaves the transform $T_p \in \mathbb{SE}(3)$ as the quantity that the measurement function “measures”. Such a transform consists of a rotation and a translation part, i.e. the two of the previously discussed measures that were considered for use.

The measurement function, which is to be designed, takes a robot pose, i.e. a transform, and a plane, i.e. another transform, to predict from them the “measurement” of the plane from the pose, which again is a transform. So basically we start with a global transform and compute the relative transform to another global transform, or in other words compute the relative transform between the two global transforms of pose and plane. Ultimately this computation is almost identical to the measurement function for consecutive robot poses (see section 4.4).

Here are just the equations for this factor, which are little more than just slight variants of the ones for the consecutive robot poses with $T_{\Theta_P}$ as the transform of $\Theta_P$. 

$$h : \mathbb{SE}(3) \times \mathbb{P}L \rightarrow \mathbb{SE}(3), (\Theta_R, \Theta_P) \mapsto \Theta_R^{-1} \otimes T_{\Theta_P} \quad (4.37)$$

$$e : \mathbb{SE}(3) \times \mathbb{P}L \times \mathbb{SE}(3) \rightarrow \mathbb{R}^6, \quad (\Theta_R, \Theta_P, z) \mapsto h(\Theta_R, \Theta_P) \ominus z \quad (4.38)$$

$$= (\Theta_R^{-1} \otimes T_{\Theta_P}) \ominus z$$

As for the Jacobians we need to remember that we have defined two distinct local coordinate system, which each come with their respective version for the Jacobians. For this the $\mathfrak{se}$-coordinate are best suited, since they are designed for transforms – they are the Lie algebra for the Lie group of transforms $\mathbb{SE}(3)$. 

34
4.5 Factor for plane landmarks

So using these coordinates we can reuse our findings from before (see equation (4.36)) and get

\[
\frac{\partial h}{\partial \Theta_R} = -\text{Adj}\left(\left(\Theta_R^{-1} \otimes T_{\Theta_P}\right)^{-1}\right)
\]

\[
\frac{\partial h}{\partial \Theta_P} = \mathbb{I}_6
\]

(4.39)

Obviously the same approximations regarding the Jacobians apply here as well, so that it suffices that they consider the measurement function instead of the error function.

4.5.4 The factor using the general form of the plane equation

The plane-factor derived above is six dimensional, as explained before a lower dimensional factor is preferable for speed as long as accuracy does not suffer too much. This is why further local coordinates systems have been derived after the six dimensional se-system. As the only other coordinate system that is complete is \(r\theta\varphi\), the new factor should use it for the Jacobians. But why limit it to the Jacobians? Since the internal representation was used as measurement before, a local coordinate system might be used just as well, which would further reduce the dimensionality of the resulting problem.

\(r\theta\varphi\) is, however, not well suited for that propose. The angles \(\theta\) and \(\varphi\) are defined in the range \((-\frac{\pi}{2}, \frac{\pi}{2})\) and \((-\pi, \pi]\) respectively. The issue now is that they warp around at the limits of these intervals, which would make any function, which uses \(r\theta\varphi\) as codomain and might cross these limits, discontinuous, which disqualifies it for use as an error function.

Furthermore, when minimizing the difference \(\alpha - \beta\) between the two angles \(\alpha\) and \(\beta\) for the purpose of aligning two objects, for example lines, it would make no sense just to head for \(\alpha - \beta = 0\), but one should also consider \(\exists k : \alpha - \beta = 2k\pi\) as minimal, as any two objects at these angles would still be aligned. Hence, there is a region in which the direction in which we need to head towards the minima is ill-defined, as both – whether increasing and decreasing the angle – are valid.

We can avoid the entire problem with angles in the error function, by revisiting the general form of the plane equation again

\[
ax + by + cz + d = 0
\]

(4.2 revisited)
4 Theoretical Approach

and while $abcd$ failed as local coordinate system, $a$, $b$, $c$ and $d$ are all Euclidean coordinates. Thus, we are fine to subtract them without too much further consideration in an error function

$$e(\Theta_R, \Theta_P, z) = \begin{bmatrix} h_a & h_b & h_c & h_d \end{bmatrix}^T - \begin{bmatrix} z_a & z_b & z_c & z_d \end{bmatrix}^T$$

(4.40)

The required measurement function is derived by transforming the plane into the robot coordinate system. To make the derivation of the Jacobians simpler later on, we choose to compute the $abcd$-coordinates before the transformation, instead of the other way round, which should be otherwise equivalent. This yields the measurement function

$$h : \mathbb{SE}(3) \times \mathbb{P} \rightarrow \mathbb{R}^4, (\Theta_R, \Theta_P) \mapsto \begin{bmatrix} R_{\Theta_R}^T \cdot \vec{n}_{\Theta_P} \\ \langle \vec{n}_{\Theta_P} | t_{\Theta_R} \rangle + d_{\Theta_P} \end{bmatrix}$$

(4.41)

with the rotation $R_{\Theta_R}$ and the translation $t_{\Theta_R}$ of the robot pose $\Theta_R$ and the normal vector $\vec{n}_{\Theta_P}$ and the d-value $d_{\Theta_P}$ of the plane $\Theta_P$.

The error function then becomes

$$e : \mathbb{SE}(3) \times \mathbb{P} \times \mathbb{R}^4 \rightarrow \mathbb{R}^4,$$

$$(\Theta_R, \Theta_P, z) \mapsto h(\Theta_R, \Theta_P) - z$$

$$= \begin{bmatrix} R_{\Theta_R}^T \cdot \vec{n}_{\Theta_P} \\ \langle \vec{n}_{\Theta_P} | t_{\Theta_R} \rangle + d_{\Theta_P} \end{bmatrix} - \begin{bmatrix} z_a \\ z_b \\ z_c \\ z_d \end{bmatrix}$$

(4.42)

which is a plain subtraction, so $\frac{\partial e}{\partial \Theta_R} = \frac{\partial h}{\partial \Theta_R}$ can be used for the Jacobians without approximating.

This leads to the Jacobians of the measurement function in $r\theta\varphi$-coordinates

$$\frac{\partial h}{\partial r\theta\varphi \Theta_R} = \begin{bmatrix} 0 & -c_{\Theta_P} & b_{\Theta_P} & 0 & 0 & 0 \\ c_{\Theta_P} & 0 & -a_{\Theta_P} & 0 & 0 & 0 \\ -b_{\Theta_P} & a_{\Theta_P} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a_{\Theta_P} & b_{\Theta_P} & c_{\Theta_P} \end{bmatrix}$$

(4.43)

$$\frac{\partial h}{\partial r\theta\varphi \Theta_P} = \begin{bmatrix} 0 & 0 & \cos \theta_{\Theta_P} \\ 0 & -c_{\Theta_P} & a_{\Theta_P} \sin \varphi_{\Theta_P} \\ 0 & b_{\Theta_P} & -a_{\Theta_P} \cos \varphi_{\Theta_P} \\ -1 & 0 & 0 \end{bmatrix}$$

with $a_{\Theta_P}$, $b_{\Theta_P}$, $c_{\Theta_P}$, $d_{\Theta_P}$, $\theta_{\Theta_P}$ and $\varphi_{\Theta_P}$ as the respective values of the plane $\Theta_P$. 36
4.6 Factor for prior on robot poses

The factors introduced thus far describe the relative relationship between consecutive pose variables and between pose and plane variables. Therefore, the relations between all variables relative to each other in the SLAM factor graph are described by just the two types of factors we already have. However, there is no absolute factor, that connects the graph to a reference frame. This can be done using a prior on the first pose. For example commonly the map frame is set to match the pose of the robot, when the SLAM is initialised. That is origin of the map, e.g. the identity transform from $\mathbb{SE}(3)$, is used as the prior on the first pose. (See section 3.3 on SLAM Initialization which discuss how to choose the prior in more detail.) Here we define a factor that states that a particular (pose) variable has “exactly” a specific value.

The measurement function $h(\Theta_R)$ is trivial, as we “measure” the value itself, thus it is

$$h : \mathbb{SE}(3) \rightarrow \mathbb{SE}(3), \Theta_R \mapsto \Theta_R$$

(4.44)

the identity function for pose, e.g. in $\mathbb{SE}(3)$-transform space. For the error function the local coordinate-operation is used, just like before with the between-factor in section 4.4. So the error function is

$$e : \mathbb{SE}(3) \times \mathbb{SE}(3) \rightarrow \mathfrak{se}(3) \equiv \mathbb{R}^6,$$

$$(\Theta_R, z) \mapsto h(\Theta_R) \odot z = \Theta_R \odot z$$

(4.45)

with the measurement $z$ as the a priori value for the pose.

The Jacobian matrix for the measurement function is trivially defined as

$$\frac{\partial h}{\partial \Theta_R} = \mathbb{I}_6$$

(4.46)

Take note of the fact that – just like before with the factor for consecutive poses – this is not actually the Jacobian of the correct function, but again the same approximations can applied, i.e. this Jacobian can be used for our purposes (see the last paragraph of section 4.4).

With the error function and the Jacobian we have a unary factor that places an absolute constraint on the graph and thereby makes the optimisation problem well-defined.
5 Software Architecture

The mapping approach described in the previous section has been implemented in C++ building upon several open-source software projects, including ROS [22] and GTSAM [2]. This section outlines some details of the implementation, how to use it and to expand upon it. For a quick overview see figure 3 on page 42.

For the back-end of my GraphSLAM implementation I utilise the iSAM2 solver from the Georgia Tech Smoothing and Mapping Library (GTSAM) [2] in order to optimise the values of the factor-graph. Consequently, I use their NonlinearFactor and DerivedValue classes as the base class for my custom factors and values.

The Robot Operating System (ROS) [22] is used to communicate with the sensors and as platform on which to provide the result of the SLAM to other applications.

The implementation itself is divided into three parts:

- **base** is in charge of the factor graph, including the optimisation of it with iSAM2. It maintains a list of MapObjects which can add factors and values to the graph. In that regard it includes a Robot-MapObject which supports robots without joints that provide continuous pose information.

- **plane** is in charge of everything related to planes, that is it implements a MapObject for plane landmarks. For this a DerivedValue for bound planes and a NonlinearFactor for plane observations is provided.

- **ros** is in charge of piecing everything together and exposing it to ROS network. This is also the only part that depends on ROS library, hence it has to act as a means to convert and translated between the internal GTSAM value types and the ROS messages.
5 Software Architecture

5.1 Base components

5.1.1 MapperBase

The MapperBase-class is the core of the mapping system. This class constructs the factor graph used to solve the Smoothing and Mapping problem which derives from the actual SLAM problem. The class comes with a spin function that uses iSAM2 to optimise the graph. This function is to be called on a separate thread.

The use of iSAM2 places some restrictions on the graph. The entire graph must be connected and the factors must constrain the intermediate linear problem sufficiently. Neither factors nor values can be removed after they have been added. Furthermore, the factors must be time-invariant, that is they may not depend on anything not-constant but the connected values.

The main functionality is provided by subclasses of MapObject. Conceptually they each represent a type of “real world objects”, such as planes or a robot’s poses at a certain fixed point in time. These objects are solely responsible for adding values to the graph using the function `addValue`. For each value an initial estimate must be provided. They must also ensure that the new values are connected to the existing values via factors. To this end they have to add factors to the graph via the `addFactor` function of the Base. In concept this is the primary source for factors to this type of value.

Instances of these objects can be added to an internal list maintained by MapperBase. Thereby a unique subsection of the available symbols is reserved for the object. This ensures that the symbol (e.g. the key) for each value is unique, while each MapObject can operate mostly independent, maybe even in parallel without interfering with others.

Lastly different scenarios maybe require different methods of acquiring the current time. For example the use time might be the time of the system the SLAM is run on, the local time of the robot, or even simulated time in the case of simulation or playback of recorded data. The MapperBase therefore provides a function to obtain the actual time used by the system.

5.1.2 Robot

Robot is a MapObject for a robot’s poses at certain fixed points in time. The object expects a source for the pose change between two arbitrary points in
time. A robot pose is represented by a 3D-transform $T_R \in \text{SE}(3)$ as described in section 4.1.1 on page 16. The mathematics behind the factors for this object is described in sections 4.2, 4.4 and 4.6.

As stated in section 4.4, we only add a value node on request. Such a request can be done by calling `getSymbolAtTime` which will return a symbol for the value so that factors to it can be constructed. By virtue of the inability of the implementation to add a value node at an earlier time than the last value that has been added to the factor graph, a new value is not added immediately but only after a `commitWindow` has passed. This allows for slower measurements to be added to the right pose. Because of this delayed addition, everything connected to the pose symbol must be added with delay as well. For this propose the class provides `addFutureFactor` and `addFutureValue`.

An instance of `Robot` represents only a single robot in the SLAM. However multiple instances can be created and added to the `MapperBase` in order to represent a multi-agent scenario. To facilitate this, one can use `setInitialPose` to adjust the pose value that is added with corresponding prior factor on initialisation as initial pose of the robot.

In addition, in such a scenario the relative positions of the robot to each other should be taken into account as measurement and therefore something should add factors between the poses of the different robots, when such relative data it available, e.g. one robot sees the another one. At least on initialisation factors connecting the robots’ pose values must be added to ensure connectivity in the graph. Unfortunately, due to lack of time it has not yet been possible to implement such functionality.

## 5.2 Plane components

`PlaneLandmark` is a MapObject for using bounded planes as landmarks. Pre-identified observations of planes are added to the SLAM with `addObservation` along with the robot that made them. For this the needed `PlaneFactors` are created and if required a new variable for the new `Planes` is added. It handles the transformations of the planes and the timing issues that arise from the concurrency of the SLAM implementation.

The `Plane` and `PlaneFactor` classes derive from GTSAM’s `DerivedValue` and `NoiseModelFactor2` respectively. The plane is internally store as described in section 4.1.2. `Plane` implements the `retract` and `local coordinate`
5 Software Architecture

operations and PlaneFactor implements the error function with its Jacobians in accordance with sections 4.3 and 4.5. As I use two different local representation, there are two versions for each of these classes.

5.3 ROS components

This part is where all the above comes together in one single ROS-node. The main class MapperROS creates the instances of MapperBase, Robot and PlaneLandmark, sets them up with ROS providers and listeners, initializes them and starts the necessary execution threads. The resulting interactions between these components is illustrated in figure 3.

For the robot poses the ROS’s tf-system is used and the corrections computed with the SLAM are published as a tf-frame. The bounded planes are detected with TableTop [12] and the Point Cloud Library (PCL) [23] and the resulting map is published using the messages from TableTop.

For evaluation purposes the observations of the bounded planes are simulated with a TestPlaneProvider which alleviates the reliance on data association for the identification of the planes as landmarks.
6 Experimental Results

Our GraphSLAM implementation was originally intended to be tested on two TurtleBot2s [11]. Unfortunately, it was not possible to test the code on the actual hardware, because of technical difficulties with the preprocessing chain for the planes so that they could not be fed into the mapper. These issues can be circumvented in simulation. So in order to get some experimental results, the entire preprocessing chain aimed at obtaining the planes from the point cloud provided by the 3D depth camera is replaced with a simulated 3D plane sensor. Since the preprocessing chain would have mostly consisted of existing tools, it is still possible to test most of the software developed.

While simulation is not the best test environment for SLAM, it allows adjusting the noise generated by the sensors. Hence, the performance under different amounts of noise can be more easily tested than with actual hardware. Additionally, a simulated world is unchanging over the test runs, which is good for a comparison between different set-ups.

The simulated robot is modelled after the TurtleBot2. The original has two actuated wheels on the diameter of its circular base footprint, which is further supported by two passive ball wheels in the front and back. This allows for car-like motion as well as rotation on the spot. Robot movement is simulated accordingly and reflected in the tf-transform-tree. As for sensors, the actual robot would provide odometry supported by a gyroscope and an Asus Xtion PRO LIVE 3D depth camera. By adding simulated noise to ground truth the simulation provides odometry data with comparable characteristic as the real robot.

As stated before, the 3D depth camera is replaced with a simulated 3D plane sensor. This sensor uses the ground truth and the simulated world already consisting of planar surfaces, to generate noisy plane observations for the ground true position and orientation of the simulated sensor. The TestPlaneProvider mentioned before takes care of this implementation-wise (see section 5.3).

Our simulated worlds are mostly maze-worlds (for example see figure 4) that expand only in the $x$ and $y$ directions with walls of constant height. Since the robot cannot leave the ground, complex structures in $z$ directions would offer
Figure 4: Top-down view on one of the test worlds. There are 36 planes in total as each of the walls has two sides but no simulated thickness. An extra challenge is added by the fact that the sensor vision range of 3.5 m is sometimes insufficient to see a wall.

little extra challenge for the tests. Such maze-like environments are optimal for Plane SLAM, since there is an abundance of planar surfaces. They are also very similar to other real world environments a robot might operate in such as office corridors for example. Unlike in reality there is a lack of clutter which might pose a challenge in plane detection.

At least three plane must be detect for a localization with Plane SLAM. Worlds with fewer walls, wide spaces or lack of planar surfaces would therefore provide a challenge pure Plane SLAM is not designed for. Wide spaces are especially challenging since typical depth cameras have a short range with a limited field of view. For example the Asus Xtion PRO LIVE has an operational distance of between 0.8 m and 3.5 m with a 58° horizontal and 45° vertical field of view. Within these sensor limits the number of detectable planes might be a very small, if there are not many around to start with. In such case the algorithm would have to be enhanced by using further types of sensor measurements, for example lines from laser scanners, which have a longer range. For this reason the tests are run in worlds where these issues should not impede the algorithms from functioning successfully.

In this paper two factor types for plane landmarks, the pose-factor in sec-
Figure 5: Plane SLAM with the abcd-based factor from section 4.5.4. The current hypothesis (in blue) for the planes according to the SLAM, needs more time to converge to exact hide the lines displaying the ground truth for the planes (in green). For some reason one of the planes in the back has started to diverge.

In all of the following experiments the PlaneSLAM-implementation using $r\theta \varphi$-factors showed some strange behaviour. At some point, fairly soon after being detected for the first time, one or two planes started to drift to infinity in the same curved paths. This never affected more than one or two of the planes and in repeated runs only ever the plane, that were affect the first time, were misbehaved. All the other unaffected planes converge to their ground true counterpart and then remained there, which is the correct and expected behaviour. This can for example be seen in back of figure 5. This show a fairly early phase in the execution, the back plane, which is only shown as ground truth, has not been detected. But the blue plane of the SLAM map has already moved across the map from out of view.

While the bug has not been found, it can be assumed that the error lies in the implementation and does not stem from the mathematics behind it. As most of the plane are well-behaved, it will be assumed that the implementation works correctly for them. However, all conclusion about the factor in its mathematical
formulation have to be taken with a grain of salt, as it cannot be said for certain that the bug does not affect them in any way whatsoever. It will find no further mention and is to be assumed for all discussion about the \( r\theta\phi \)-factor.

The first experiment uses low artificial noise, similar if not higher than the levels of noise on the real sensor, and places no “vision” limits on the plane sensor. In this optimal scenario both factor types preformed quite well. Although it was noticeable that the \( r\theta\phi \)-factor was slightly faster in converging to the truth, while not exactly hitting the mark, always staying a little bit off. With the pose-factors, on the other hand there were problems when turning on the same spot for an extended time, showing ever greater lag. This was mostly due to processing time, as rotating generates a lot of factors.

After 10 to 20 minutes the buffer for recording the robot movement, which stores only the last minute, became too small to provide a between-factor for the poses and the program terminated. Using the \( r\theta\phi \)-factor it took about twice as long to reach this point. Which makes sense as the dimension of this factor is half that of the pose-factor (3 instead of 6), thus the dimension of the information matrix in the iSAM2 increases half as fast for the same number of factors. Towards the end of the execution the matrix inversions and other iSAM2 step were responsible for almost all of the processing time used.

Next the plane sensor was limited to the aforementioned 3.5 m vision range of the depth camera on the physical robot, thus the number of visible plane dropped significantly. Depending on its location on the map the view port of the sensor did not even detect the required minimum of three detected planes. This has a significant impact on the quality of the SLAM. Especially if the robot stayed in a region with too few detections for some time, the accuracy dropped. In a few instances the SLAM stopped working entirely as the drift due to noise had thrown it completely off course.

For the last experiment the vision limit was widened while the artificial noise was increase significantly at the same time. For the pose-factor one of the runs can be seen in figure 6. Most of the observations deviate considerably from the actual true plane, but with sufficient time and observation the map can be reconstructed with amazing precision. If one compares this result with figure 5 which shows the use of the other type of factor in a low noise run, one sees the accuracy the pose-factor-type can achieve. However, when turning rapidly the map lags behind significantly and only catches up, when the robot stops. In comparison, the \( r\theta\phi \)-factor in this extreme scenario is often not even able to construct any map whatsoever.
Figure 6: Plane SLAM with the transform-based factor from section 4.5.3. The recently detected planes (in red) are used to compute the current hypothesis (in blue) for the planes according to the SLAM. The ground truth for the planes (in green) is mostly hidden behind the map planes, which illustrates the accuracy of the SLAM map.

In conclusion, neither of the factors is better than the other in all situations. The high noise scenario in which the $r \theta \varphi$-factor fails is barely relevant in practice, as the plane segmentation with a well-calibrated depth camera providing the input usually provides fairly precise measurements for the distance and normal of a plane. Settings where the noise is higher, such as a cloud of dust or fog, make plane detection very difficult and reduce the number of potentially detectable planes. Therefore, Plane SLAM would not be the optimal choice here in the first place. In all other tests, while there were differences, no clear winner emerged.
7 Conclusion

This thesis presents a real-time mapping system enabling a robot to chart planar surfaces in its environment. Different solutions to integrate plane observations into the factor-graph optimisation as constraints have been explored. While none was clearly superior, it has become clear that the choice of a factor for planes has serious impact on the performance and accuracy of the SLAM. Regrettably, I did not manage to test the system on the physical robot, but simulation shows that the principle works as expected in most cases. Nevertheless, there is still room for improvement, the system is far from bug-free and would greatly profit from additional features. Particularly in the long run, optimisation slows down to a crawl showing unexplained phenomena, like the diverging planes discussed in chapter 6.

7.1 Future Work

7.1.1 Internalising Plane Identification

The main reason why the system could not be tested on the robot itself was that the interface with the software doing the identification did not work as expected. To avoid such problems and to improve performance, it would be beneficial to develop an identification algorithm that uses similar formulas to identify the plane as those used by the SLAM as the factor for optimisation of the known planes. Thereby identification and SLAM would become more consistent. Furthermore, the data association problem is tightly connected to the SLAM problem so solving them together makes sense. This paper misses out on the association part, this should be rectified in future work. This could give PlaneLandmark-class discussed in section 5.2 a more meaningful role.
7 Conclusion

7.1.2 Using inlier points as a plane-factor

As mentioned in section 4.5.2 the inlier points that matched the plane during segmentation have not been considered as measure for use by a plane-factor. This was mostly based on practical consideration as the plane segmentation software used does not make them available via ROS. If one were to use the plane segmentation algorithm from the Point Cloud Library (PCL) [23] and the method of Rusu et al. [24] directly instead of utilizing TableTop [12] as an intermediary, one would obtain access to the set of inlier points that the segmentation process used to fit the plane. These inliers provide for a few interesting uses in a plane-factor. For example, Taguchi et al. [27] used a random subset sampled from the set of inlier points for plane registration.

Such a technique might be interesting to use in a SLAM system. In the context of my SLAM system this could be done by computing the minimal Euclidean distance between each point in the sample and the plane. Each of these distance measures would become an additional dimension in the measurement function.

This higher dimensionality might pose a challenge, yet the potential to have the extent of the plane contribute to the optimisation might be worth it. Using additional knowledge about the plane should always lead to an improvement in accuracy at the cost of speed. On the other hand higher accuracy means we require fewer observations thus it might be even faster to make use of the inlier points.

Even if ultimately using them in the optimisation itself proves too slow, they will definitely improve the plane identification. Besides, this is surely not the only way to incorporate the inlier points into the SLAM.

7.1.3 Multi-agent extension

As already mentioned in section 5.1.2, the SLAM system I devised should be able to handle multiple robots. Once it were able to run successfully on one real robot, it would be worthwhile to explore how the plane-factor fares in a multi-agent SLAM. With this end in view factors constraining the robots’ locations on observation by another robot would need to be devised. This could be done in a manner similar to that chosen by Kim et al. [19] and Huang et al. [10]. The PlaneLandmark as-is should be able to handle observations from multiple robots. By adding plane-factors from multiple robots one would also add indirect constraints between them whenever they observe the same plane.
landmarks. Identifying the planes as the same from the perspective of various robots, however, might require an adjustment of the algorithms.

7.1.4 Map Saving and Loading

In the current implementation the map only exits as solution of the factor graph optimisation. The map will be lost whenever the execution ends and rebuild from scratch when it is resumed. The objective of map creation is to end up with a map. Therefore, we should save the map to disk before we end the map creation process and thereby delete it again.

While storing the latest solution of the factor graph optimisation to file is trivial, loading it is not. Loading a map into SLAM requires some thought on how to initialize the factor graph, i.e. the SLAM. Furthermore, the question is whether just saving the solution, i.e. all the values of the variables, is what has to be done. The entire solution would contain the path the robot has travelled. Does it make sense at all to kept that data? Or could it just be dropped as the old path is irrelevant to any (new) robot loading the map? In contrast, we might even want to store the factors as they contain our knowledge on how accurate each part of our map is. Also, this information could be stored as covariance on a prior factor, we could use to initialize the plane landmarks after loading the map.

Hence, the saving and loading of the map is far from trivial and needs some careful consideration, but it is absolutely necessary for a successful mapping solution to be able to do so.

7.1.5 Dynamic reduction of the factor graph

In the experiments I observed as described in chapter 6 that after some time, i.e. after too many variables and factors have been added, the optimisation slows down to a crawl. Therefore, some method needs to be devised to reduce the number of factor that are involved in the optimisation. I can think of two ways this problem could be approached.

One could remove the oldest robot poses from the graph and thereby also the connected factors, reducing their number. Classically this would be done by marginalising. This, however, does not work well with the idea behind iSAM2 and keeping the entire trajectory, which is that by not marginalising the history of the poses the matrices involved in the optimisation will stay sparse.
7 Conclusion

So marginalising the oldest pose would make the problem more dense, thus slowing optimisation down instead of speeding it up.

In the previous section I proposed that the factors could be accumulated in a prior factor for planes on saving/loading the map. Something similar could be done here, dropping old factors from the graph and replacing them by a unary prior-like factor as it seems safe to assume that a factor effect will have sufficiently taken hold after several optimisation runs. Although one would need to be careful as this breaks constraints placed upon the plane landmarks and indirectly on their relative locations as well, as long as one leaves sufficiently many factors in place one might be fine to do so. Furthermore, care needs to be taken that no variable is left completely unconstrained or unconnected as we are removing constraints. Obviously this would require a solid theory behind it in order to do it properly. Also, one should remember that it is somewhat tricky to remove a factor from an iSAM2-factor-graph-optimisation.

7.1.6 Hierarchical map/factor graph

Taking it one step further, one could consider removing distant parts of the map from the optimisation as has been done in various hierarchical SLAM designs like the one from Estrada et al. [6]. However, Estrada et al. use EKF-based SLAM in their work. So with graph-based SLAM it would require some new theory building. Whether it is possible with an unmodified version of iSAM2 as solver is an open question. Combining the benefits from graph-based full trajectory SLAM with the hierarchical approach could certainly produce some interesting results and make them it applicable to even wider areas.

7.1.7 Covariance of variables for active exploration

On a completely different note and returning to the motivation of this paper, extending this SLAM system to allow for active exploration should be considered. iSAM2 provides a covariance for each variable in its respective local space. This covariance could be utilised in an active exploration algorithm as it represents the confidence towards a specific plane landmark. Thereby one could derive an algorithm driving the robot to explore its surroundings optimally, i.e. reduce the covariance on the plane landmark variables.
Nomenclature

⟨·|·⟩ the dot product of two vectors

\( a \times b \) the cross product of two 3D-vectors \( a \) and \( b \)

\( M^T \) the transpose of a matrix \( M \)

\( \|a\| \) the 2-norm of a vector \( a \): \( \|a\| = \sqrt{\langle a|a \rangle} \)

\( \|A\| \) the matrix norm of \( A \)

\( \|a\|_\Sigma \) the Mahalanobis distance with covariance matrix \( \Sigma \): \( \|a\|_\Sigma = \sqrt{a^T \Sigma a} \)

\( P(A,B) \) joint probability over \( A \) and \( B \)

\( P(A|B) \) conditional probability, \( A \) given \( B \)

\( \text{sinc } a \) unnormalized cardinal sine function \( \text{sinc } : a \mapsto \begin{cases} \frac{\sin a}{a}, & a \neq 0 \\ 1, & a = 0 \end{cases} \)

\( V \otimes U \) The composition of two elements \( U,V \) of a Lie group. In the case of transforms \( U,V \in SE(3) \) given in homogeneous coordinates, the composition is the matrix multiplication: \( V \otimes U = V \cdot U \)

\( V \oplus \Delta V \) Applies \( \Delta V \) to \( V \) and returns the result. Both \( V \) and the result are part of the same space, if this space is a Lie group then \( \Delta V \) must be an element of the associated Lie algebra. Given the exp-map is well-defined, the update can be computed as \( V \oplus \Delta V := V \otimes \exp \Delta V \)

Identity: \( V \oplus (V \ominus U) = U \)

\( V \ominus U \) Computes local coordinates of \( U \) given in \( V \). \( V \) and \( U \) are element of the same space, if this space is a Lie group the retract returns an element of the associated Lie algebra. Given the log-map is well-defined, the local coordinates can be defines as \( V \ominus U := \log^{abcd} (V^{-1} \otimes U) \)

Identity: \( V \ominus (V \oplus \Delta V) = \Delta V \)

\( \mathbb{N} \) the natural numbers: \( 1, 2, 3, \ldots \)

\( \mathbb{R} \) the real numbers
**Nomenclature**

\( \mathbb{SO}(n) \) the special orthogonal group, also called rotation group. Its elements are the usual rotations in n-dimensional space. For \( n = 3 \) the elements/rotations an be given as \( 3 \times 3 \)-matrix with \( \det = 1 \).

\( \mathfrak{so}(n) \) the Lie algebra of \( \mathbb{SO}(n) \). It consists of all skew-symmetric matrices. The Lie bracket, given by the matrix commutator \( [A_1, A_2] = A_1A_2 - A_2A_1 \) with \( A_1, A_2 \in \mathfrak{so}(n) \) is again a skew-symmetric matrices.

When \( l \in \mathfrak{so}(3) \) is represented as a 3-dimensional vector \([l_x, l_y, l_z]\) the following commonly-used basis is used throughout the thesis:

\[
L_x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \quad L_y = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \quad L_z = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}
\]

The commutation relations of these basis elements \([L_x, L_y] = L_z, [L_z, L_x] = L_y, [L_y, L_z] = L_x\) agree with the relation under cross product for the standard unit vector of the \( \mathbb{R}^3 \), which makes this basis quite suitable.

\( \mathbb{SE}(n) \) the special Euclidean group. It is a group of the direct isometries, that is to say isometries preserving orientation, also called rigid motions. These are the moves of a rigid body in n-dimensional space. The elements can be given as tuples of a rotation \( R \in \mathbb{SO}(n) \) and a translation \( t \in \mathbb{R}^n \). For \( n = 3 \) it is also possibly express them as transform in homogeneous coordinates \( T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in \mathbb{SE}(3) \) with \( R \in \mathbb{SO}(3) \) and \( t \in \mathbb{R}^3 \).

\( \mathfrak{se}(n) \) the Lie algebra of \( \mathbb{SE}(n) \). The Lie bracket is given by the matrix commutator \([A_1, A_2] = A_1A_2 - A_2A_1\) with \( A_1, A_2 \in \mathfrak{se}(n) \).

\( \mathbb{PL} \) the space of bounded planes in 3D.

\( \mathbb{PL} = \mathbb{SE}(3) \times (\mathbb{R}^2)^n \) is the space of all 2-tuples of \( T_p \in \mathbb{SE}(3) \) and \( n \)-length point sequence hull, that is \( \left(T_p, \text{hull} = (p_0, \ldots , p_n)\right) \in \mathbb{PL} \) with \( T_p \in \mathbb{SE}(3), \forall i : p_i \in \mathbb{R}^3 \land \forall j \neq k : p_j \neq p_k \) and \((p_0, \ldots , p_n, p_0)\) being a convex \( n \)-sided polygon. For more detail see subsection 4.1.2, especially (4.6) and (4.8).

\( i, j, k \) index variables \( i, j, k \in \mathbb{N} \)

\( x, y, z \) the coordinates of some point in a Cartesian coordinate system \( x, y, z \in \mathbb{R} \)

\( T \) a coordinate transform \( \in \mathbb{SE}(3) \)
\( R \) a rotation \( \in \mathbb{SO}(3) \)
\( t \) a translation \( \in \mathbb{R}^3 \)
\( \Theta \) a variable in the factor graph \( \in \mathbb{R}^n \)
\( h(\cdot) \) a measurement function over some variables \( \Theta_i \). Often as part of the measurement error \( h(\Theta_i) - z_i \) a measurement \( z_i \)
List of Equations

3 Background
(3.1) SLAM Objective ................................................. 9
(3.2) SLAM Objective for Filtering ................................. 10
(3.3) SLAM Objective for Smoothing ......................... 10
(3.4) Function factorization by a factor graph ................ 11
(3.5) Factorization of SLAM objective ....................... 11
(3.6) SLAM objective factorized as factor graph ............ 11
(3.7) Gaussian factor ............................................... 12
(3.8) Factorized SLAM objective as nonlinear least-squares criterion ... 12

4 Theoretical Approach

4.1 Representation
(4.1) Transform in homogeneous coordinates ................. 16
(4.2) Plane equation in general form ............................. 16
(4.3) Plane equation in point-normal form ..................... 16
(4.4) Plane equation in parametric form ....................... 17
(4.5) Coordinate transform of a plane in parametric form .... 17
(4.6) Representation for the plane landmarks .................. 18
(4.7) Conversion between the different types of plane equations ...... 18
(4.8) Definition of the $PL$ ........................................... 18
List of Equations

4.2 Local coordinates for robot poses

(4.9) Basis for the $\mathfrak{so}(3)$ ........................................... 19
(4.10) Exponential map for the $\mathfrak{so}(3)$ ................................ 19
(4.11) An logarithm map for the $\mathfrak{so}(3)$ as derived from Rodrigues’ formula 20
(4.12) Exponential map for the $\text{SE}(3)$ .............................. 20
(4.13) Logarithm map for the $\text{SE}(3)$ ................................ 20
(4.14) Infinitesimal transform in homogeneous coordinates ........ 20
(4.15) Jacobian of $f: \text{SE}(3) \to \mathbb{R}^m$ .............................. 21
(4.16) retract-operation for $\text{SE}(3)$ with infinitesimal transform .... 21
(4.17) Definition of retract-operation for $\text{SE}(3)$: $V \oplus \Delta V$ .......... 21
(4.18) Approximation of retract-operation for $\text{SE}(3)$................. 21
(4.19) Definition of local coordinate-operation for $\text{SE}(3)$ .......... 21
(4.20) Approximation of local coordinate-operation for $\text{SE}(3)$ .... 22

4.3 Local coordinates for plane landmarks

(4.21) Jacobian of $f: \text{PL} \to \mathbb{R}^m$ using $\mathfrak{se}(3)$-tangent space ......... 23
(4.22) Definition of retract-operation for $\text{PL}$ using $\mathfrak{se}(3)$-tangent space .. 23
(4.23) Definition of local coordinate-operation for $\text{PL}$ using $\mathfrak{se}(3)$-tangent space .......................................................... 23
(4.24) Definition of logarithm map for the $abcd$-tangent space of $\text{PL}$ .... 24
(4.25) Definition of local coordinate-operation for $\text{PL}$ using $abcd$-tangent space .......................................................... 24
(4.26) Jacobian of $f: \text{PL} \to \mathbb{R}^m$ using $abcd$-tangent space ............ 24
(4.27) Transform of a plane in $r\theta\varphi$-tangent space ...................... 26
(4.28) retract-operation for $\text{PL}$ using plane transform in $r\theta\varphi$-tangent space 26
(4.29) Exponential map for the $\text{SE}(3)$ .............................. 27
(4.30) Logarithm map for the $\text{SE}(3)$ ................................ 27
(4.31) Definition of retract-operation for $r\theta\varphi$ ...................... 27
List of Equations

(4.32) Definition of local coordinate-operation for $r\theta\varphi$ .......................... 27
(4.33) Jacobian of $f : \mathbb{P} \rightarrow \mathbb{R}^m$ using $r\theta\varphi$-tangent space .............. 27

4.4 Factor for time-consecutive samples of a single robot’s poses
(4.34) Measurement function for time-consecutive samples of a single robot’s poses ......................................................... 28
(4.35) Error function for time-consecutive samples of a single robot’s poses 29
(4.36) Jacobians of the measurement function for time-consecutive samples of a single robot’s poses ............................... 29

4.5 Factor for plane landmarks
(4.37) Measurement function for plane landmarks using $xy$-plane of a transformed coordinate system ............................... 34
(4.38) Error function for plane landmarks using $xy$-plane of a transformed coordinate system ............................................. 34
(4.39) se-Jacobians of the measurement function for plane landmarks using $xy$-plane of a transformed coordinate system ............ 35
(4.40) Concept for the error function using $abcd$ plane coordinates .... 36
(4.41) Measurement function for plane landmarks using the general form of the plane equation ........................................... 36
(4.42) Error function for plane landmarks using the general form of the plane equation .......................................................... 36
(4.43) $r\theta\varphi$-Jacobians of the measurement function for plane landmarks using the general form of the plane equation ............... 36

4.6 Factor for prior on robot poses
(4.44) Measurement function for priors on robot poses ......................... 37
(4.45) Error function for priors on robot poses ................................. 37
(4.46) Jacobians of the measurement function for priors on robot poses 37
List of Figures

3 Background
1 Factor Graph with robot poses and plane landmarks ............... 13

4 Theoretical Approach
2 A plane in $r\theta\phi$-tangent space .................................. 26

5 Software Architecture
3 Interactions between the components in the implementation ... 42

6 Experimental Results
4 Top-down view on one of the test worlds ......................... 44
5 Plane SLAM with the abcd-based factor ......................... 45
6 Plane SLAM with the transform-based factor .................... 47
Bibliography


Bibliography


XII


Bibliography


Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature