Development and Evaluation of a Robocentric SLAM Algorithm

Max Holmberg
Abstract

Development and Evaluation of a Robocentric SLAM Algorithm

Max Holmberg

In this master thesis the front end of a Simultaneous Localization And Mapping (SLAM) system is developed in the programming language C++ and the meta-operating system ROS (Robot Operating System). The algorithms are based on previous work done at the Swedish defense research agency (FOI) and are a part of a GPS free positioning system developed for military use. The parts that have been implemented during the project includes: feature extraction from LIDAR data, feature association and a Robocentric Extended Kalman Filter. The sensors used in the SLAM system are a Velodyne LIDAR (LIght Detection And Ranging) unit and an IMU (Inertial Measurement Unit). During the master thesis, data collection has been done in different types of outdoor environments. The resulting front end SLAM with a Kalman filter is evaluated in the different types of environments and compared with both accurate RTK (Real Time Kinetic) GPS and a version of the filter that uses data filtered with a GPS. The GPS free SLAM algorithm in urban and forest environments gives position estimates that drifts less than 2% compared with the SLAM algorithm that has help from a GPS. In open field terrain the GPS free SLAM algorithm has trouble estimating its position due to a lack of features, which results in significant drift over time. When the SLAM algorithm with GPS filtered data is compared with an accurate Real Time Kinetic GPS in an urban environment the average drift is less than 1%.
Populärvetenskaplig sammanfattning

Acronyms

**EKF** Extended Kalman Filter. 1–3, 6, 8, 9, 17, 28, 29

**JCBBB** Joint Compatibility Branch and Bound. 29

**LIDAR** Light Detection And Ranging. 1–4, 9, 12, 15–17, 28–30

**ROS** Robot Operating System. 2, 3, 17, 28, 30

**RTK** Real Time Kinetic. 2, 3, 16, 18, 22, 23, 27, 30

**SLAM** Simultaneous Localization And Mapping. 1–5, 8, 9, 11, 12, 14, 17, 19–23, 27–30
## Contents

1 Introduction ................................. 1
   1.1 Background .................................. 1
   1.2 Problem Definition .......................... 1
   1.3 Sensors and Platforms ....................... 2
   1.4 Related Work ............................... 2

2 Theory ....................................... 4
   2.1 Simultaneous Localization And Mapping .... 4
   2.2 Quaternions ................................ 5
   2.3 Composition and Transformation Inversion ... 6
   2.4 The Kalman Filter ........................... 7
     2.4.1 Extended Kalman Filter ................. 8
   2.5 Robocentric EKF-SLAM ....................... 9
     2.5.1 The Prediction Step .................... 9
     2.5.2 The Update Step ........................ 9
     2.5.3 The Composition Step .................. 10
   2.6 $k$-d Tree .................................. 11
   2.7 Feature Extraction and Association ......... 12
     2.7.1 Segmentation ............................ 12
     2.7.2 Plane Estimation ........................ 12
     2.7.3 Plane Comparison ....................... 13
     2.7.4 Feature Association ..................... 14
     2.7.5 Excluding Untrustworthy Features ...... 14
     2.7.6 Circular Features ....................... 14
   2.8 $\chi^2$-test ................................ 14
   2.9 Odometry Update ............................. 15

3 Method ...................................... 16
   3.1 Sensors and Data Acquisition ............... 16
   3.2 Software and Libraries ...................... 17
   3.3 The Data .................................... 17
   3.4 The ROS Network ............................ 17

4 Results ...................................... 18
   4.1 Calibration Measurement ..................... 18
   4.2 Urban ...................................... 19
   4.3 Forest ...................................... 20
   4.4 Field .................................... 21
   4.5 Comparison with RTK-GPS .................... 22
   4.6 Quantitative Comparison ..................... 23
   4.7 Analyses of Covariance Matrix ............... 23
     4.7.1 Covariance as a Function of Feature Quantity ... 25

5 Discussion ................................. 27
   5.1 Discussion of Results ....................... 27
     5.1.1 Positional Differences between the Environments ... 27
     5.1.2 Covariance Matrix Analyses .............. 27
   5.2 Discussion of Method ....................... 28
   5.3 Future Work ................................. 29

6 Conclusions ............................... 30
1 Introduction

1.1 Background

Robotics is the science of perceiving and manipulating the physical world through computer-controlled devices [1]. This degree project will deal with perceiving the physical world, specifically relating the robot position in relation to the environment. The concept of Simultaneous Localization And Mapping (SLAM) consist, as the name suggests, of two main parts, the positioning of the robot in relation to a map, and the construction of that map from sensor data. The map generally consist of a number of landmarks that are extracted from sensor data, and observing these landmarks more than once will reduce the localization error of the robot.

SLAM and related techniques are being deployed in many different real-world applications, from self-driving cars to mobile devices. In the coming years many different types of technology will increasingly rely on positioning from SLAM algorithms, particularly in situations where GPS or similar positioning systems are not available, e.g. indoors, in situations where a higher precision is needed or in military applications.

A universal solution to the problem of SLAM has been a long time goal within the robot community. Probobalistic SLAM algorithms started to be developed in the 1980s, by among others Peter Cheeseman, Jim Crowley and Hugh Durrant-Whyte. It was also during this time period that a statistical basis for describing relationships between landmarks and manipulating geometric uncertainty was developed [2]. The classical approach to solving the SLAM problem is using an Extended Kalman Filter (EKF) for estimating the pose of the robot and the position of landmarks. EKF SLAM algorithm uses a map that is represented by a collection of features extracted from sensor data. At every time point \( t \) the robot observes a vector of bearings and ranges to nearby features. Odometry measurements along with these feature measurements are then used for estimating the pose of the robot [1]. EKF SLAM could also be featureless, but in this project the system is feature based. A variant of EKF SLAM is to use a robocentric frame of reference, this will reduced the linearization errors introduced by the EKF whilst still maintaining the same level of accuracy [3]. A more detailed description of the EKF-algorithm will be presented in the theory part.

Since the SLAM system developed in this master thesis are to be used mainly for military purposes a somewhat different approach is needed compared with for example an industrial robot. The first thing is that the SLAM system will work as a complement to the already existing positioning techniques for military vehicles. SLAM is often used for fully autonomous systems whilst it will in this implementation be used as an additional tool for the soldiers. Furthermore, the idea is that the system will be placed on more than one vehicle driving for example in a convoy. If the SLAM system can be synchronized between many different vehicles it will be a considerable advantage. Many robots and autonomous systems are meant to function in an indoors environment, but that is irrelevant for military vehicles and this SLAM system need to be able to work in many different types of outdoor environments.

1.2 Problem Definition

This degree project will consist of further development, evaluation and improvement of a SLAM-algorithm used at FOI, the Swedish Defense Research Agency. The SLAM-algorithm is a part of the development of a positioning system that is independent of GPS and that should work for military applications. The goal is to use the SLAM algorithm on a number of vehicles that work together to estimate their environment and their position in it [4]. The sensors that are used are LIDAR (Light Detection and Ranging) data and an IMU (Inertial Measurement Unit). The algorithm is robocentric [3] and employ a backend graph based optimization (see [5]) to fuse SLAM output between multiple platforms. The SLAM system has been developed in MATLAB previously to construct a working demonstration and prove that the techniques used are viable for the given situation. As a part of the further development that is to be executed during
this master thesis, the algorithm will firstly be taken from the two dimensional case to three dimensions and the working environment will be C++ instead of MATLAB for optimization purposes. The collection of software frameworks called ROS (Robot Operating System) [6] is also used in conjunction with C++ to establish the framework on which the software will run. The previous version of the SLAM-algorithm runs on data that has been previously collected whilst the new version will be able to run fully online with incoming data. This degree project will aim to implement the front end (without using the graph based optimizations) part of the algorithm for one vehicle in three dimensions in the programming language C++. The algorithm will then be evaluated and compared with accurate GPS.

The questions that will be answered in this thesis project are,

- How well the front end LIDAR feature based EKF-SLAM handles different outdoor environments.
- How much difference odometry updates with GPS assistance does in comparison to fully GPS free updates.
- How does this type of algorithm compare with an RTK GPS.
- How well does the estimated covariance matrix represent the error in position and how is the covariance matrix affected by feature scarcity.

### 1.3 Sensors and Platforms

The long term goal of the FOI project is to have a SLAM-algorithm fully independent of GPS, thus the only resources available for the positioning should be the LIDAR and the IMU.

Using LIDAR as the main sensor for perceiving the environment is a common way of providing accurate data for SLAM algorithms as [7], [3], [8] are examples of. LIDAR is a remote sensing method that uses light in the form of a pulsed laser to measure ranges. LIDAR is used for a number of different applications for example geodesy, archeology, laser guidance, seismology and control and navigation of autonomous cars etc. [9]. Other methods for perceiving the environment in a SLAM-algorithm includes monocular or binocular visual imaging, [10] and sonar [11]. LIDAR has the advantage of providing not only the bearing to a landmark but also accurate range measurements which can be a major issue when using for example monocular visual images. Some of the LIDAR data used for the SLAM algorithm have been collected previously but some have been collected as a part of this thesis.

As a first step in constructing a fully GPS free outdoor positioning system the SLAM-algorithm can run with GPS filtered velocity as training wheels. This is due to the fact that the SLAM has to be very good in order to be able to run with non processed accelerometer data. In the results a comparison with and without the GPS filtered velocity is done. For the evaluation part of this project an RTK GPS is used as a "ground-truth" to be able to compare how accurate the SLAM algorithm is, this GPS is however not used in the actual SLAM algorithm.

The platform on which these sensors are placed is a car. The sensors are placed on a setup on the roof of the car such that the sensors are far enough above the roof so that the car does not disturb the measurement.

### 1.4 Related Work

The problem of simultaneous localization and mapping is not a new issue, it has been an active research field since the 1980s [2]. It has been a field in which great progress has been made during the last 30 years. In a 2D indoor environment with a robot with wheel encoders and a laser sensor the problem can be considered as solved if an accuracy of 10 cm is considered enough, see for example the commercial KUKA navigation solution [12]. SLAM is however a very broad subject and many robots, environments and performance criteria demands more
fundamental research in order to function with a satisfactory accuracy and robustness. Since this master thesis revolves around a SLAM algorithm that is supposed to function outdoors in a large variety of environments, without GPS with relatively limited sensors the task is not an easy one and today’s commercial solutions are not adequate. There are a vast amount of examples for achieving accurate EKF-based SLAM. In [10] a monocular visual-inertial odometry algorithm is used on a UAV. Whilst using a robocentric EKF-filter they do not use any LIDAR for sensing the environment. Instead they rely on pixel intensity errors of image patches.

In [7] an algorithm for estimating position from just LIDAR data is presented. A great deal of inspiration for this project is drawn from this paper, for example to use the kd-trees in the association step. Also in [7] an interesting way of extracting features based on curvature in the point cloud is used with good results. One major difference between the FOI project and [7] is that an IMU is used in the FOI SLAM in addition to LIDAR data as a complement.

Another paper from which inspiration on choice of method was drawn is [3]. In this paper the consistency of EKF-SLAM is analyzed. One of the conclusions was that a robocentric world-view will reduce the linearization errors produced in a global reference frame EKF which is why this approach is used in the FOI SLAM system.

FOI has also previously conducted research on outdoors positioning, for example in [13] where a system for automatic estimation of tree position and tree stem diameter was developed. Also in [14] the benefits and capabilities of 3D LIDAR on unmanned aerial vehicles were examined.
2 Theory

2.1 Simultaneous Localization And Mapping

A naive way of performing position estimates of a robot or vehicle is to use raw accelerometer data and integrate it twice to get the position estimate. While this is theoretically correct the resulting system is an unstable dynamic system which means that even the smallest of errors will grow over time resulting in very large errors. This is why the IMU functions only as a complement to the observations of the environment. As explained earlier, a feature based SLAM revolves around observing the same feature more than once in order to estimate position. In figure 1 a vehicle with a sensor can be seen, this vehicle observes a feature A at time step k. In the next time step k+1 the same feature is again observed by the sensor, however the vehicle has moved since the last time step resulting in different range and bearing to the feature A. From these measurements the movement of the vehicle can be estimated. This methodology demands a static world, if features are relocated during a run, errors will occur. However, as long as the movement of features are limited, the dynamics of the surrounding world can be treated as noise.

A modern SLAM system includes two main components, the front end and the back end [15]. The front end part of the system deals with feature extraction and other preprocessing of the sensor data whilst the back end performs inference on the data provided by the front end. An example of a SLAM back end algorithm is to formulate SLAM as a maximum a posteriori estimation problem expressed through factor graphs, which is what will be used as the back end filter in the SLAM system developed at FOI. This degree projects focus is on the front end part of a SLAM algorithm and thus will not venture further in to the back end component.

In order for the algorithm to estimate the vehicles position and its surrounding environment the front end component of the SLAM system needs to extract relevant features from sensor data. Regardless if the sensor data are visual images, LIDAR, Sonar or any other type of sensor data the SLAM needs distinguishable points in the environment to be able to give a position estimate. It also deals with associating the extracted features. The association process consists of finding where the new measured features belong in the map of landmarks. Furthermore, it also gives a first estimate of the map and position to be passed on to the back end part.

In this project the front end will consist of feature extraction based on planes in LIDAR data, feature association will be done with kd-trees and an Extended Kalman Filter which will do a first estimate of the map and vehicle position.

![Figure 1: The vehicle (black square) observes the range $r_1$ and bearing $\phi_1$ to the feature A at two different locations. The movement between position $p_1$ and $p_2$ can then be calculated.](image-url)
2.2 Quaternions

The standard method for handling rotations in 3D-space is to use 3x3 rotation matrices. However, since rotation matrices can suffer from computational and data handling issues an alternative to rotation matrices are quaternions which provide a more useful way of representing rotations [16].

Quaternions were first introduced by William Rowan Hamilton in 1843 [16], and are defined as

$$q = w + ix + jy + kz$$  \hspace{1cm} (1)

where \(w\) is the scalar part and \(ix + jy + kz\) is the vector part of the quaternion. A quaternion is a 4-tuple, thus it defines an element in \(\mathbb{R}^4\) and is an expansion of the regular complex number,

$$c = a + ib$$  \hspace{1cm} (2)

and behave in similar ways. \(i, j, k\) have the following properties,

$$i^2 = j^2 = k^2 = ijk = -1$$  \hspace{1cm} (3)

which are similar to the properties of the imaginary number \(i\). \(w, x, y, z\) are all real numbers or scalars and should not be confused with the normal notation for Cartesian coordinates. From equation 3 all quaternion arithmetic follows, except multiplication, quaternion multiplication is similar to matrix multiplication in that it is not commutative.

The most important use for quaternions is indeed to perform rotations of vectors and reference systems. A quaternion rotation can in general be expressed as,

$$v' = qvq^{-1}$$  \hspace{1cm} (4)

where \(q\) is a quaternion representing a rotation and \(v\) is a position described in a Cartesian system by \(v = xi + yj + zk\) [16]. For example, let the position vector of a point be \(v = [1, 0, 3]\), and let this point be rotated \(\theta = 180^\circ\) around an axis defined by the unit vector \(u = 0i + 0j + 1k\) (which happens to be the z axis in a Cartesian system). Through an extension of Euler’s formula we get that the quaternion \(q\) that performs this rotation through equation 4 is defined as,

$$q = \cos(\frac{\theta}{2}) + u * \sin(\frac{\theta}{2})$$  \hspace{1cm} (5)

which leads us to the quaternion \(q = 0 + 0i + 0j + 1k\). With the help of equation 4 and the quaternion arithmetic defined in equation 3 we now get the new position,

$$v' = qvq^{-1} = (0 + 0i + 0j + 1k) * (1i + 0j + 3k) * (0 + 0i + 0j − 1k) = −1i + 0j + 3k$$  \hspace{1cm} (6)

In figure 2 this rotation is illustrated. In the SLAM algorithm used and developed in this thesis quaternions are used for all rotation operations.
Figure 2: Illustration of a rotation of 180° around the z axis of a point \( v = [1, 0, 3] \).

2.3 Composition and Transformation Inversion

Two operators that are essential when dealing with translations and rotations of poses are transformation inversion and composition represented by \( \ominus \) and \( \oplus \) respectively [17]. They are especially important when reference systems are moving. A pose is a combination of position and rotation and is in this case a 7D vector \( p = [x, y, z, q_w, q_x, q_y, q_z] \). The behavior of these operators is described by,

\[
\hat{x}_B^A = \ominus \hat{x}_B^A
\]

\[
\hat{x}_C^A = \hat{x}_B^A \oplus \hat{x}_C^B
\]

where \( A, B \) and \( C \) are reference frames. As an example let \( p_1 \) and \( p_2 \) be two different poses as shown in figure 3a and 3b. The resulting pose from the transformation composition operation is shown in figure 3c. In other words, the operation \( p = p_1 \oplus p_2 \) means that the second poses transformation is concatenated to the reference system already transformed by the first pose. The inversion operator \( \ominus \) can for example describe \( p_2 \) in figure 3b as \( p_2 = p_1 \ominus p_1 \), which means that the pose \( p \) (in global coordinates) is seen as \( p_2 \) with respect to the reference frame \( p_1 \) [18]. These operators are nonlinear and if they are to be used in the measurement model in the EKF their Jacobians are needed in order to do a first order linearization. These jacobians are defined as,

\[
J_{\ominus}(\hat{x}_B^A) = \frac{\partial(\ominus \hat{x}_B^A)}{\partial \hat{x}_B^A} \bigg|_{\hat{x}_B^A}
\]

\[
J_{1\oplus}(\hat{x}_B^A, \hat{x}_C^B) = \frac{\partial(\hat{x}_B^A \oplus \hat{x}_C^B)}{\partial \hat{x}_B^A} \bigg|_{\hat{x}_B^A, \hat{x}_C^B}
\]

\[
J_{2\oplus}(\hat{x}_B^A, \hat{x}_C^B) = \frac{\partial(\hat{x}_B^A \oplus \hat{x}_C^B)}{\partial \hat{x}_C^B} \bigg|_{\hat{x}_B^A, \hat{x}_C^B}
\]

In Appendix section A, these Jacobians are explicitly expressed in 3 dimensions with quaternion representation (as described in [18]).
2.4 The Kalman Filter

The Kalman filter is in general terms used as a technique for filtering and prediction in linear Gaussian systems and was first introduced by Swerling (1958) and Kalman (1960). In the Kalman filter a state vector and its covariance matrix are updated with the help of the difference between a measurement and a calculated expected measurement [1]. The following section will describe the Kalman filter in more detail.

Let the state vector to be estimated be expressed by the following linear function with added Gaussian noise,

\[ x_k = F_k x_{k-1} + B_k u_k + w_k \]  \hspace{1cm} (12)
where \( x_k \) is the state vector of size \( n \times 1 \), \( u_k \) is the control signal of size \( m \times 1 \), \( B_k \) and \( A_k \) are matrices of size \( n \times n \) and \( m \times m \) respectively. \( w_k \) is added Gaussian noise that models the uncertainty introduced by the state transition. The covariance of \( w_k \) is \( Q_k \). A measurement of the state \( x_k \) is described as,

\[
z_k = H_k x_k + v_k
\]  

where \( H_k \) is the observation model and \( v_k \) is the observation noise with covariance \( R_k \).

The Kalman filter consist of two distinct steps, the prediction and the update steps. The prediction is expressed as follows,

\[
\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + B_k u_k
\]

\[
P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k
\]

When the prediction step is done the update step follows as,

\[
K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1}
\]

\[
\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(z_k - H_k \hat{x}_{k|k-1})
\]

\[
P_{k|k} = (I - K_k H_k) P_{k|k-1}
\]

### 2.4.1 Extended Kalman Filter

The equations 12 to 18 govern the Kalman filter if the system to be filtered is linear, that is if the measurement can be described by 13 and the state vector by 12. However for many systems this relationship is not linear, for example when using the Kalman filter for SLAM. Take for example a robot with constant translational and rotational velocity, it will move on a circular arc which cannot be represented by linear state transitions. In general, the non-linear state transitions in robotics render the normal linear Kalman Filter useless in most robotics cases [1]. In a nonlinear system the state vector and measurement can be described by the following equations,

\[
x_k = f(x_{k-1}, u_k) + w_k
\]

\[
z_k = h(x_k) + v_k
\]

where \( f \) and \( h \) are non-linear functions. In order to cope with the non-linearity the matrices \( F_k \) and \( H_k \) in equations 14, 15, 16, 17, and 18 are instead defined by the Jacobians,

\[
F_k = \frac{\partial f}{\partial x} \bigg|_{\hat{x}_{k-1|k-1}, u_k}
\]

\[
H_k = \frac{\partial h}{\partial x} \bigg|_{\hat{x}_{k|k-1}}
\]

This version of the Kalman filter is known as the Extended Kalman Filter(EKF). Since SLAM is a non-linear process it is important to use the extended Kalman filter. However it is also important to remember that the EKF is a linearization of a non-linear problem and as such it will introduce linearization errors. In many cases the linearization errors are negligible but in some cases methods for reducing these errors are required [3].
2.5 Robocentric EKF-SLAM

The goal with the EKF-SLAM algorithm is to estimate not only the position of the robot in relation to its environment but also its environment, in this case represented by landmarks identified from the LiDAR point cloud. The state vector will thus consist of both the position of the robot and the position of landmarks. Inconsistency problems is an issue that arises with EKF-SLAM due to linearization errors. This issue can be reduced with a robocentric reference frame as shown in [3]. The linearization errors are reduced with the reduction of uncertainty provided by a robot centered representation. The general EKF-SLAM algorithm must be slightly altered to be able to incorporate a robocentric world view. The robocentric EKF-SLAM consist of three distinct sections: prediction, update and composition.

The state vector in Robocentric EKF-SLAM will have the following structure, as described by [3],

\[
\hat{x}_k^B = [\hat{x}_k^R, \hat{x}_{F_1}^R, \hat{x}_{F_2}^R, \ldots \hat{x}_{F_n}^R]^T \tag{23}
\]

where \(\hat{x}_k^R\) is the robots absolute position in reference to a base frame, this state variable is non-observable and is only needed if an absolute map is to be recovered. \(\hat{x}_{F_i}^R\) are the landmarks position in the robot frame of reference. The uncertainty matrix will have the following structure,

\[
P^R = \begin{bmatrix}
P_{B}^R & \cdots & P_{BF}^R \\
\vdots & \ddots & \vdots \\
P_{FB}^R & \cdots & P_{F_n}^R
\end{bmatrix} \tag{24}
\]

2.5.1 The Prediction Step

In a robocentric world view it is not the robot that moves, instead it is all the features that changes position between time steps. A reasonable first step in the algorithm would be to perform this through the following location update,

\[
x_{R_k}^{F_k|k-1} = \otimes x_{R_{k-1}}^{k-1} \oplus x_{F_{k-1}}^{R_{k-1}} \tag{25}
\]

This means that the features in the map \(x_{F_{k-1}}^{R_{k-1}}\) at time \(k - 1\) are translated and rotated in accordance with how the new odometry measurement suggest the vehicle has moved, predicting where the features will be at time step \(k\) in relation to the robot at time \(k\). The different pose transformations are displayed in figure 4. However in order to reduce the errors introduced by composing the landmark positions with imprecise odometry measurements, this step is performed after the Kalman Filter update. This means that during the prediction step the new measured position and its covariance are simply added to the state in the following way,

\[
\hat{x}_{k|k-1}^R = [\hat{x}_{R_{k-1}}^R, \hat{x}_{k-1}^R]^T \tag{26}
\]

\[
P_{k|k-1}^{R_{k-1}} = \begin{bmatrix}
Q_k & 0 \\
0 & P_{k-1}^{R_{k-1}}
\end{bmatrix} \tag{27}
\]

2.5.2 The Update Step

The update step begins with relating the measurement \(z_k\) with the expected measurement as in 20. However with the robocentric algorithm we get,

\[
z_k \simeq h_k(\hat{x}_{k|k-1}^R) + H_k(x_{R_{k-1}}^R - \hat{x}_{k|k-1}^R) \tag{28}
\]
Figure 4: Illustrates the prediction step described by equation 25. The black squares are the vehicle at time k and time k-1, the red arrows are the pose transforms as seen from reference frame $R_{k-1}$ and the blue arrows are the pose transforms as seen from $R_k$.

where,

$$H_k = \frac{\partial h_k}{\partial x_{F_k}} \bigg|_{\hat{x}_{R_k|k-1}} = [H_{R_k}, 0, \ldots, 0, H_{F_k}]$$

(29)

The update step then follows the standard procedure for the Kalman filter depicted in equations 16, 17 and 18.

2.5.3 The Composition Step

When the filter has been updated the features now need to change position based on the filtered robot position change. So the step that was skipped in section 2.5.1 must now be performed. The features positions will be affected in the following way,

$$\hat{x}_{R_k} = \begin{bmatrix} \ominus\hat{x}_{R_k} \ominus \hat{x}_{0} \\ \ominus\hat{x}_{R_k} \ominus \hat{x}_{F_1} \\ \vdots \\ \ominus\hat{x}_{R_k} \ominus \hat{x}_{F_n} \end{bmatrix}$$

(30)

with covariance matrix,

$$P_{R_k} \simeq [J_{J_1}, J_{J_2}]P_{R_{k|k-1}}[J_{J_1}, J_{J_2}]^T$$

(31)

where,

$$J_{J_1} = \begin{bmatrix} J_1 \ominus (\ominus \hat{x}_{R_k}, \hat{x}_{R_0}) \\ \vdots \\ J_1 \ominus (\ominus \hat{x}_{R_k}, \hat{x}_{F_n}) \end{bmatrix}$$

(32)

$$J_{J_2} = \begin{bmatrix} J_2 \ominus (\ominus \hat{x}_{R_k}, \hat{x}_{R_0}) & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & J_2 \ominus (\ominus \hat{x}_{R_k}, \hat{x}_{F_n}) \end{bmatrix}$$

(33)

where $J_1$ and $J_2$ are the jacobians related to the composition and transform inversion operations described in section 2.3.
2.6 k-d Tree

In the feature association part of the SLAM algorithm a k-d tree is used to quickly be able to search through the points. In this section the structure and function of k-d trees will be explained in short.

A k-d tree is a data structure for storage of information to be retrieved by associative searches. In more specificity it is a multidimensional binary search tree [19]. A binary search tree has the advantage of having the complexity $O(\log(n))$ for an average search, and $O(n)$ for a worst case search [20]. A multidimensional binary search tree or a k-d tree, where the $k$ stands for the number of dimensions, work in a similar way. As the name suggest a multidimensional tree stores $k$-dimensional data, but still has the complexity $O(\log(n))$ for nearest neighbor queries, which is the task we want to use the k-d tree for in the SLAM algorithm. In figure 5 a k-d tree of dimension two is illustrated. When a new node is inserted in a k-d tree a discriminator (an integer between 0 and k-1) determines along which axis the tree is to be split. In the two dimensional case there is only 0 and 1, or x and y for illustrative purposes. The new node is then placed as the left or right child depending on the vector value at the discriminator index.

Figure 5: Illustration of a k-d tree.

(a) A two dimensional k-d tree of size 5.

(b) A two dimensional k-d tree of size 5 illustrated in the x-y plane.
2.7 Feature Extraction and Association

The process of extracting features from a point cloud is no easy task and requires careful consideration. The main type of features that are used in this SLAM algorithm are edge features, e.g., a corner of a house or similar. In figure 6 a top down view of a point cloud from which features are to be extracted can be seen. Different methods can be used for extracting edge features, for example by measuring the curvature of the point cloud as in [7], or by using methods from the world of image processing as in [21]. However for the algorithm in this thesis project a feature extractor based on estimating and comparing planes between different scan lines is used. The main idea is to first segment a subsection of the 16 scan discs from the the 16 laser sensors in the Velodyne unit, then estimate planes of those segments. If the corners of the planes from different scan discs are matching, a vertical feature has been found. The following section describe the method in more detail.

![Figure 6: A top down view of the total point cloud that is generated by the LIDAR.](image)

2.7.1 Segmentation

The segmentation of a scan disc is done recursively. The algorithm starts with estimating a line for all points in a scan disc. If there are points whose distance to that line is greater than a threshold \( \text{maxError} \), the point that is furthest away is marked and the scan disc is split into a left and right scan disc and a line is again estimated from the sub scan discs. The process is repeated until no lines have outliers further away than \( \text{maxError} \). In figure 7a the segmentation is illustrated. Note that on the left side of the figure three small segments are created due to uneven data points.

2.7.2 Plane Estimation

The segmentation has provided a number of quasi edge points, the points in between these edge points can now be estimated as a plane. Sections consisting of 4 or less points are estimated as
(a) A subsection of the point cloud in figure 6. The segmentation is illustrated with different colors.

(b) After the planes are estimated from the segmentation the end of each plane is set as a quasi edge. These quasi edges are then compared between the scan discs, and if they match up an edge feature has been found.

Figure 7: Illustration of two different steps in the feature extraction process.

planes perpendicular to the bearing of the points. Larger sections are estimated with a range, and orientation of the plane relative to the measurement unit.

2.7.3 Plane Comparison

Now when each scan disc has a number of estimated planes, a comparison between scan lines is done. If the edges of planes from a number of different scan discs are at the same position (or in close vicinity to each other), a vertical feature has been found. In figure 7b the quasi edges (in red) that are created at the end of each plane estimate can be seen, along with the edges (in green) that are the same in multiple scan disc, creating a feature.
2.7.4 Feature Association

The feature association is the process of matching new observed features with previously registered features. The matching is done by creating a search tree, a kdtree, with all the previously registered features, and then doing a nearest neighbor search to see if i) The distance from the observation is further than a given distance $\delta$ or ii) The distance is less than $\delta$. If case i) the observation is of a new feature that has not previously been seen and a new registered feature will be created, if case ii) the observation is of a previously seen feature and the observation is packaged in to a message to be passed to the Kalman filter. In this SLAM algorithm two $k$-d trees are used for association. The first one, called a clutter tree, makes sure that a feature is consistent by counting how many initial associations a feature has, if it is higher than a threshold it is passed on to the second association tree as a consistent feature.

2.7.5 Excluding Untrustworthy Features

The feature extraction described above produces several spurious features. Features that changes when the vehicle moves are useless and are to be avoided. To avoid these features a number of tests are done to see if they can be trusted. These criteria are describe in [7] and consist of the following,

- Exclude points whose surrounded points are features
- Exclude points on a surface that is parallel to the laser
- Exclude points on the boundary of an occluded region

These criteria removes certain types of spurious features but there is no guarantee that untrustworthy features can be found regardless.

Another criteria that the features need to fulfill are a $\chi^2$ test, that are explained further in section 2.8.

2.7.6 Circular Features

As a complement to the edge features other features can also be extracted and used. One of the most common shapes in an outdoor environment are circular shapes e.g. trees and lampposts. For circular objects, the segmentation algorithm previously described will produce many small segments from one circular object. The first step in the circular feature extraction is thus to cluster together small segments that are next to each other. When the clusters have been assembled, a circle fit is performed on the points in the clusters. The circle fitting algorithm used is a Taubin circle fit, see [22]. If the fitting process converges and the fitting error is below a certain threshold a circular feature has been found. The circles found in different scan discs are then compared in a similar way as with the edge features.

2.8 $\chi^2$-test

When the new observations have been associated with previously observed features a $\chi^2$ test is done in order to filter out bad associations. A $\chi^2$ test is a statistical hypothesis test that provides a value for how likely a certain event is. In the case of finding the most likely associations each new measurement is given a $\chi^2$ value based on the following expression,

$$\chi^2 = (z - \hat{z})^T S^{-1} (z - \hat{z})$$  \hspace{1cm} (34)$$

where $S$ is defined as $S = P_{\text{feature}} + R$ and where $P_{\text{feature}}$ is the covariance of the feature in question and $R$ is the Sensor noise. $z$ is here the new measurements Cartesian coordinates and $\hat{z}$ is the expected measurement. The resulting $\chi^2$ value is in other words based on how far from
the feature the measurement is \((z - \hat{z})\) and how certain the filter is of the features position with added sensor noise \((S)\). The new feature observation will be used in the update if its \(\chi^2\) value is below a threshold and if it is has the lowest \(\chi^2\) value of all the observations of a feature.

2.9 Odometry Update

Since we know that the vehicle, on which the LIDAR sensor is placed, is a four wheeled ground vehicle we can use that knowledge to our advantage. Such a vehicle will always move with a curved path, i.e. we do not allow any pure y movements. This concept is displayed in figure 8.

The equations that describe the movement update are as follows if \(\omega_z = 0\),

\[
x = v_x \cdot dt \cdot \cos \left( \frac{\omega_z \cdot dt}{2} \right)
\]

\[
y = v_x \cdot dt \cdot \sin \left( \frac{\omega_z \cdot dt}{2} \right)
\]

\[
z = v_z \cdot dt
\]

where \([x, y, z]\) is the positional vector to be calculated, \(v_i\) is the current velocity in the \(i:\)th direction, \(\omega_j\) is the current rotation around the \(j:\)th axis and \(dt\) is the time during which the movement is performed. If \(\omega_z \neq 0\) the movement updates is expressed as,

\[
x = \frac{2v_x}{\omega_z} \cdot \sin \left( \frac{\omega_z \cdot \delta t}{2} \right) \cdot \cos \left( \frac{\omega_z \cdot \delta t}{2} \right)
\]

\[
y = \frac{2v_x}{\omega_z} \cdot \sin \left( \frac{\omega_z \cdot \delta t}{2} \right) \cdot \sin \left( \frac{\omega_z \cdot \delta t}{2} \right)
\]

\[
z = v_z \cdot dt
\]

Figure 8: An illustration of the movement restrictions imposed in the odometry update. The vehicle will always either move in a straight line or in a curved arc. No pure y movements are allowed.
3 Method

In this section the various measuring techniques, software tools and information about the data is presented. The software structure is also described.

3.1 Sensors and Data Acquisition

The two sensors that are used in the measurements are an Inertial Measurement Unit (IMU), and a velodyne LIDAR sensor. The IMU used is an xsens MTi-G-710 [24] capable of providing accelerometer and gyroscope data, GPS filtered position and velocity with high accuracy, and magnetometer data. The data from the accelerometer, gyroscope and the filtered GPS velocity are/can be used in the algorithm. The LIDAR sensor is a Velodyne Puck VLP 16 [25] which has 16 channels, 360° horizontal field of view and 100 m range. LIDAR (LIght Detection And Ranging) is a remote sensing method that uses light in the form of a pulsed laser to measure ranges and bearings [9]. In figure 9 a point cloud generated by the LIDAR sensor can be seen.

Figure 9: A screen shot of the point cloud generated by the LIDAR sensor placed on a car in a built up area.

The measurements were done by placing the LIDAR sensor and IMU on the roof a car. The vehicle was then driven through different types of environments, e.g. built up areas, forest and fields. The generated point cloud and IMU-data were saved to so called ”rosbags” for convenient playback at a later time. In order to have a ground truth to compare the algorithm with, an RTK GPS was used in some of the measurements. An RTK (Real-time kinetic) GPS uses measurements of the phase of the carrier wave to improve the position accuracy down to cm level. The RTK GPS used was a Topcon GRS-1 [26] capable of providing position accuracy up to 1 cm, in the measurements the estimated accuracy was below 5 cm for all data points.
3.2 Software and Libraries

All code is written in C++ compiled with the gcc compiler version 5.5.0 (20171010). The matrix and vector operations are done through the template library "Eigen" [27]. The data received from the LIDAR sensor are processed with the help of the Point Cloud Library (PCL), which is a large scale, open project for 2D/3D image and point cloud processing [28].

Robot Operating System (ROS) is an open-source, meta-operating system for robots. It provides services like hardware abstraction, low-level device control, package management among other things. It provides tools and libraries for running code on multiple computers. A running instance of ROS consist of a number of nodes, with a central node called a roscore. These nodes can easily communicate by posting messages and listening to messages sent by other nodes [29]. ROS is the backbone of the SLAM algorithm in this thesis and the ROS version that is used is "Kinetic".

3.3 The Data

The data that was collected can be put in to four different categories, calibration measurement, urban measurement, forest measurement and field measurement. These four different scenarios have very different behavior and will thus produce different positioning accuracy. In general, the fewer the features the more difficult it is to estimate the vehicles position. Since the IMU outputs both the raw accelerometer/gyroscope data and a version that has been filtered with the help of a built in GPS both of these can be used with the algorithm. It is the GPS filtered velocity data that can be used for updating the odometry, since the the goal is to be independent of GPS no positional data is used. In the result section, data is presented where the GPS filtered velocity has been used and a comparison is made with no GPS used.

3.4 The ROS Network

The ROS-network for the SLAM algorithm has the structure depicted in figure 10. As can be seen from figure 10 the inputs to the network are a point cloud from the LIDAR sensor and IMU samples. The input data is then processed in the input node to make sure the IMU and LIDAR data are synchronized. The input node can either receive the raw accelerometer data or use the GPS filtered velocity data. The input node also extracts features from the point cloud as described in section 2.7. The output from the feature extraction is a point cloud of features. The features are then associated by the FeatureAssociation node as described in section 2.7.4. The EKF-node is an extended Kalman filter algorithm as described in section 2.5. Its outputs are an estimated pose along with an estimated feature map. Observe that the EKF node also publishes an estimated speed based on the filtered position for the input node to use for adjusting the incoming accelerometer data. The IMU-bias mentioned in figure 10 is not currently estimated with the EKF.
Figure 10: An illustration of a typical iteration in the front end filter, starting with the odometry being updated and ending with updating the vehicle pose.

4 Results

The measurements in this section are divided into four different categories depending on the type of environment in which they were performed. In order of difficulty the categories are, calibration, Urban, Forest and Field. A factor that seems to affect the accuracy of the positioning significantly is the speed of the vehicle which is why it is stated in each section. The start and end positions that are displayed in the figures are the start and end positions of the best estimate in that figure, so either RTK GPS or SLAM with GPS filtered velocity.

4.1 Calibration Measurement

This measurement was performed on the FOI grounds. The vehicle moved in as much of a straight line as possible in an environment with plenty of good features. The distance was relatively short and the velocity was slow, below 10 km/h. Due to the relatively kind circumstances this measurement was used as a calibration measurement when tuning the various parameters (see appendix B) in the algorithm. As can be seen from figure 11 the algorithm without GPS help
and with GPS help are almost identical.

Figure 11: The estimated positions when the SLAM is used in the calibration environment.

4.2 Urban

The urban measurements were performed in the type of environment that can be seen in figure 12. The abundance of houses, trees and lampposts among other things increases the number of features that are found and should thus provide a relatively easy environment for positioning. This measurement was done at a low speed 10-15 km/h which in this case results in inaccurate IMU measurements since the signal to noise ratio in the accelerometer decreases at lower speeds. In figure 13 the positioning result can be seen.

Figure 12: The type of environment that classifies as Urban.
Figure 13: The estimated positions when the SLAM is used in the urban environment.

4.3 Forest

The forest measurement was performed in the type of environment that can be seen in figure 14. This measurement was performed at a speed of around 20 km/h and was not a loop. The positional estimates can be seen in figure 15.

Figure 14: The type of environment that classifies as forest.
Figure 15: The estimated positions when the SLAM algorithm is used in a forest environment.

4.4 Field

This measurement starts in an urban environment similar to figure 12 but the second half of the measurement are in more open terrain as can be seen in figure 16. This resulted in a measurement with little or no features during the second half of the run. The lack of features meant that positioning was poor as is evident by mainly the "Filtered position without GPS" line in figure 17. This measurement was also performed at up to 20 km/h.

Figure 16: The type of environment that classifies as field.
Figure 17: The estimated positions when the SLAM algorithm is used in an urban/field environment.

4.5 Comparison with RTK-GPS

In order to estimate how far from the true position the best version (with GPS filtered velocity) is, an RTK GPS was used to get very accurate position data. The RTK GPS was only used in the urban environment and the result can be seen in figure 18.

Figure 18: The estimated positions with GPS filtered velocity along with the RTK-GPS positions and the SLAM without any GPS velocity.
4.6 Quantitative Comparison

In this section some quantitative results are presented, in table 1 the SLAM algorithm using no GPS is compared to the SLAM algorithm using filtered GPS velocity. In table 2 a comparison between the RTK GPS and the SLAM algorithm using filtered GPS velocity is made and in table 3 the SLAM with no GPS is compared with RTK GPS. The "Mean difference" in tables 1, 2 and 3 is the average position difference divided by the total distance traveled. The "Drift per m" is the difference between the end positions divided by the total distance traveled. The values in parentheses are standard deviation.

<table>
<thead>
<tr>
<th></th>
<th>Calibration</th>
<th>Urban</th>
<th>Forest</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean difference per m in $\hat{x}$</td>
<td>8.2e-05</td>
<td>0.0048</td>
<td>0.0087</td>
<td>0.0603</td>
</tr>
<tr>
<td>Mean difference per m in $\hat{y}$</td>
<td>0.6e-05</td>
<td>0.0068</td>
<td>0.0072</td>
<td>0.0086</td>
</tr>
<tr>
<td>Drift per m in $\hat{x}$</td>
<td>1.9e-05</td>
<td>0.0129</td>
<td>0.0081</td>
<td>0.1176</td>
</tr>
<tr>
<td>Drift per m in $\hat{y}$</td>
<td>1.8e-05</td>
<td>0.0202</td>
<td>0.0098</td>
<td>0.0142</td>
</tr>
</tbody>
</table>

Table 1: Comparison between the GPS free SLAM and the version with GPS filtered velocity.

<table>
<thead>
<tr>
<th></th>
<th>Calibration</th>
<th>Urban (0.088)</th>
<th>Forest</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean difference per m in $\hat{x}$</td>
<td>-</td>
<td>0.0221</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Mean difference per m in $\hat{y}$</td>
<td>-</td>
<td>0.0248 (4.27e-06)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Drift per m in $\hat{x}$</td>
<td>-</td>
<td>0.0008</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Drift per m in $\hat{y}$</td>
<td>-</td>
<td>0.0081</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2: Comparison between the RTK GPS and the SLAM algorithm with GPS filtered velocity. The standard deviation divided by travel distance is presented in parenthesis.

<table>
<thead>
<tr>
<th></th>
<th>Calibration</th>
<th>Urban (0.0092)</th>
<th>Forest</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean difference per m in $\hat{x}$</td>
<td>-</td>
<td>0.0259</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Mean difference per m in $\hat{y}$</td>
<td>-</td>
<td>0.0242 (3.77e-07)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Drift per m in $\hat{x}$</td>
<td>-</td>
<td>0.0202</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Drift per m in $\hat{y}$</td>
<td>-</td>
<td>0.0182</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3: Comparison between the RTK GPS and the SLAM algorithm with no GPS. The standard deviation divided by travel distance is presented in parenthesis.

4.7 Analyses of Covariance Matrix

In this section a closer look at the relationship between the estimated covariance of the position and the number of features that are found in the environment is taken. This section only covers the fully GPS-free SLAM algorithm. In figures 19 to 22 both the number of features that are found during a measurement and the 2-norm of the cumulative covariance matrix are shown. The 2-norm of the covariance matrix is used here to get an easy overview of the size of the covariance matrix. In the calibration measurement and the forest measurement (figures 19 and 21) there are never a lack of features generating low values in the covariance matrices. The field and Urban measurements (figures 22 and 20) both generate higher estimates of the covariance matrix. Since these figures display the cumulative 2-norm of the covariance matrix it is the slope of the curve that shows how well the estimate is at a certain point. A steeper slope means that the covariance increases more at each time step generating more uncertain estimates.
Figure 19: Number of associated features at each update step and the cumulative 2-norm of the covariance matrix of the position estimate in the calibration measurement.

Figure 20: Number of associated features at each update step and the cumulative 2-norm of the covariance matrix of the position estimate in the urban measurement.
Figure 21: Number of associated features at each update step and the cumulative 2-norm of the covariance matrix of the position estimate in the forest measurement.

Figure 22: Number of associated features at each update step and the cumulative 2-norm of the covariance matrix of the position estimate in the field measurement.

4.7.1 Covariance as a Function of Feature Quantity

In figure 23 the average over 5 measurements of the 2-norm of the covariance matrix is plotted against the number of associated features. From figure 23 it is evident that there is a limit below
which the covariance matrix never reaches. This limit is reached approximately when more than 50 different features can be used for the positional estimate and the value is, for the parameters in appendix B, $2.18 \times 10^{-4}$. Since the relationship between number of associated features and the 2-norm of the covariance matrix seem to be exponential a numerical fit is also shown in figure 23. The numerical curve fit is surprisingly good with an $R^2$ value of 0.99 and a summed square error of $1.25 \times 10^{-6}$.

\[ f(x) = 0.0067219 \exp(-0.56186x) + 0.001498 \exp(-0.074543x) \]

$R^2 = 0.98902$

$\text{SSE} = 1.2523 \times 10^{-6}$

Figure 23: The average 2-norm of the position covariance matrix over five measurements as a function of the number of features that are associated.
5 Discussion

5.1 Discussion of Results

The most obvious thing to comment regarding the results is the fact that in the cases where the RTK GPS was not used, the comparison is made between two filters that use partially the same data to estimate the position, which of course should be kept in mind. However the goal was to achieve a GPS free SLAM and as such it may be considered fair to compare the difference between the two versions. Furthermore, the version with GPS filtered velocity should be the best estimate for all cases since we base the position estimate with the help of more information. Also, as can be seen in table 2 the error between the ground truth, the RTK GPS in this case, and the SLAM algorithm with GPS velocity is relatively small with a drift of less than 1% which makes the SLAM algorithm with GPS filtered velocity viable as a quasi ground truth. Another important thing to consider is that the measurements and quantitative data presented here are based on one single example of urban/forest/field environment. The data is not necessarily completely representative of all types of environments in all cases.

5.1.1 Positional Differences between the Environments

As expected the measurement that was partly done in a field environment was clearly the most difficult and with no help from GPS measurements the SLAM algorithm cannot position the vehicle correctly. In figure 17 it is clear that in the first part, from the start position to the second corner, the difference between the two filter versions are in the same order of magnitude as the urban or forest measurements. However, as soon as the vehicle enters the more open terrain during and after the second corner it loses features and thus has to rely on the accelerometer data which in itself is very inaccurate, which results in a poor estimate.

In the forest and in the urban environment however the filter versions are more similar, and a drift between 1 and 2 percent is achieved. The average difference between the two filter versions are in these environments less than 1 percent. In the calibration environment the estimates are so close to each other they can almost be considered identical. If we allow ourself to use the filter version with GPS filtered velocity and allow ourself to consider the RTK GPS as a ground truth the positions estimated with the SLAM algorithm are around 2% wrong on average but has a drift far below one percent in an urban environment. It is worth noting that also the fully GPS free SLAM performs well in the Urban environment compared with RTK, almost as well as with GPS velocity. The drift without GPS velocity is slightly higher but during most of the measurement the average error is low.

From these numbers we can say that in both the urban and the forest environment the GPS free SLAM performs well, especially when considering the fact that no back-end optimization is used. The military vehicles on which this type of system is to be implemented on are expected to traverse many different types of environment. As the data suggest, the front-end SLAM is not robust in field terrain and might also lack accuracy in other types of environment. The back-end optimization will help but might not be enough if the algorithm is expected to work in all types of terrain.

5.1.2 Covariance Matrix Analyses

It is relatively evident from figures 19 to 22 that a higher number of features generate a lower uncertainty. The covariance matrix seems from these figures to increase relatively linear for a given number of features. In the urban measurement in figure 20 the 2-norm of the covariance matrix at the end of the measurement is around the same magnitude as the field measurement in figure 22. However, from table 1 it is clear that the urban position estimate is significantly better than the field estimate. In other words, solely looking at the uncertainty does not necessarily give an estimate of how accurate the position is.
Note that in the field measurement in figure 22 there is a large patch in the middle with little or no features found. Also the urban measurement have patches with few or no features found, these patches are shorter than in the field measurement and in between these patches good features are found, which generates a much better position estimate. Since the odometry has more time to drift, one long period of no features is worse than many small periods with no features but the covariance matrix solely is not enough to draw this conclusion. Even so, the covariance matrix does give an idea of how well the algorithm performs in different scenarios which is clear when comparing table 1 and figures 19 to 22 since a lower estimated covariance matrix correlates with a better position estimate.

Figure 23 suggests that there is an exponential relationship between number of features and the 2-norm of the covariance matrix. This relationship might be interesting when creating feature extraction algorithms for a SLAM system. Whilst it is true that more features generate a better estimate, it seems as this is only true up to a limit. The limit in this case is around 30 features after which there is little improvement. What is also interesting is that when the features are few there is a large difference between finding for example three features compared to finding one feature. This suggest that a feature extractor that is adaptive could be significantly more efficient than a static one. If the feature extractor extends its search area in feature scarce environments and restricts it in feature abundant areas, providing a more constant feature output, it might provide a considerable improvement. The features in the extended search will of course be of worse quality and might not provide as good material for the position update.

5.2 Discussion of Method

Using LIDAR and an IMU as the sole sensors works well in many situations and since bearing and ranges can directly be extracted from the point clouds it is a convenient solution, but it also has its limitations. The velodyne LIDAR has a relative short range, 100 m according to the specifications, but in reality it is almost impossible to find recognizable features at such a distance with the sensor. As long as the road is close to vegetation or buildings, the sensors provide good data but they may need to be complemented in some way in order for the SLAM algorithm to work in more difficult environments.

The ROS-network in combination with C++ were great tools for this type of algorithm and was a large part in making the algorithm fast enough to run on live data. The extended Kalman filter that was used has a number of advantages, key among them is the fact that it is a well documented filter for this type of implementations and as such it is very convenient to use. In its simplicity the EKF works well as a front end filter. There are many different types of approaches for feature extraction from LIDAR data, the variant of estimating planes in the data and then extracting edges from that, was previously used in the MATLAB version with satisfactory results, which is why it was decided to be used also in the C++ version. The kd-tree approach to feature association was another component that contributed to the speed of the algorithm in comparison with the previous MATLAB version.

The strategy for feature extraction in this SLAM version was creating quantity rather than quality. This is simply due to the fact that the results for the environments with a high amount of features were much more accurate with this strategy. Currently the feature extractor passes on basically all points that can even remotely be considered features and then lets the $\chi^2$ filter sort out which features that matches the already constructed map the best. This strategy is viable as long as there are many features in the environment. The SLAM algorithm runs in to trouble when the features are scarce however. If the number of features are not only few but also has a high uncertainty, the filter will either adjust the filter incorrectly or rely more on inaccurate accelerometer data. This type of behavior can be seen in the field environment, see figure 17. The alternative is to impose higher restrictions on what can be considered a feature, providing better but fewer features. This strategy may work in some cases and is less likely to match different features to each other and thus embedding errors in to the filter. The issue is however that with
fewer features the algorithm must rely more on accelerometer data which we want to avoid as much as possible. There is in other words a trade off between finding good features and finding many features, and in this version of the SLAM algorithm there is plenty of room for more fine tuning of the feature extraction parameters.

As previously mentioned the speed of the vehicle on which the sensor is placed affects the accuracy of the positioning. Naturally, the higher the speed of the vehicle the less LIDAR points per distance traversed which means less information for the filter to use. However, an interesting aspect is that at lower speeds, or rather lower accelerations, the accelerometer signal to noise ratio will be significantly smaller, which in turn leads to more inaccurate measurements from the IMU. The speed of the vehicle in the measurements was mostly between 10 and 20 km/h. The rather slow speeds provided high detail in the LIDAR points, but an interesting aspect would be to use the algorithm also at higher speeds.

5.3 Future Work

In order to truly get a GPS free SLAM solution more work is certainly required. The EKF-SLAM is of course just a part of the whole positioning system, the final system is meant to be run with a graph based optimization node as the back end filter. However the scope of this degree project is to focus solely on the front end EKF-SLAM on one vehicle and as such there are a number of improvements that should be implemented in the future to get a better more complete SLAM. The first thing that should be implemented is the 3D aspect. The aim of this master thesis was to create a fully 3D SLAM but due to lack of time this version, whilst operating in a fully 3D world estimating full 3D rotations and 3D movement, cannot find the height of features and thus cannot give a good estimate of the height or orientation around the horizontal axises. Since the only thing that is required is to estimate the height of the features, the algorithm could with relative ease start to work in full 3D in the future however.

As previously mentioned the more features the filter has, the better the estimate is. A future improvement would be to include more types of features, for example by adding the type of edge features based on curvature as in [7]. Also the nearest neighbor search that is done in the association step could be replaced with a Joint Compatibility Branch and Bound (JCBBB) test. As described in [30] the JCBBB test uses joint compatibility of pairings to reject spurious matchings, creating more robust feature associations.

One further way of improving the filter is by not only include the position in the state vector but also update the velocity in the Kalman filter, instead of adjusting the velocity with an average based on the filtered positions as in the current version. This would decrease the error that builds up in the IMU over time which would especially be helpful for the fully GPS-free version.

Trying different types of filters for the SLAM algorithm was outside the scope of this thesis, it is worth mentioning though that other possibly better suited filters than the Extended Kalman Filter could be used. A very similar Gaussian filter that potentially could produce more accurate estimates is the Unscented Kalman Filter which uses deterministic sampling techniques in order to reduce the error produced by non-linearities.

One aspect that has not been explored in this SLAM system is feature descriptors. A feature descriptor differentiates a feature from other features by assigning a ”numerical fingerprint” to it. A sort of descriptor is used in the algorithm when circular features and edge features are differentiated, but the subject could be explored further. A feature description system could reduce the number of incorrect associations that are made.
6 Conclusions

In this degree project, the front end component of a GPS-free SLAM system was implemented in C++ and ROS. The SLAM algorithm is based around a velodyne LIDAR sensor and an IMU. The algorithm can run fully online, operates in a 3D world and uses binary kd-trees for fast feature associations. The feature extraction is based around estimating planes in the point clouds and from there extracting edge features, along with circular features for tree trunks and lampposts etc. Through a fully robocentric representation of the filter state, major consistency errors was avoided.

In the urban and forest environments the drift and average error are a few percent compared with the SLAM algorithm using GPS filtered velocity. Also when compared with highly accurate RTK GPS the algorithm drift and average error is low. The SLAM algorithm has difficulties in more feature sparse open terrain. Due to the inherent inaccuracy of accelerometer data the algorithm drifts significantly when the features are few. The covariance matrix is shown to give an estimate of position error but can not always be trusted. Figure 23 shows that the relation between number of features in an update step and the estimated covariance matrix in that step is exponential, suggesting that more features only gives better estimates in feature scarce environments. Implementing a graph-based back-end optimization algorithm will improve the results and is the next big step in the development of the system. Due to lack of time during the project, the front end algorithm cannot fully estimate the position in three dimensions since the height of the features cannot be determined, this will be the most urgent improvement in further development of the algorithm. Further improvement of the front end can also be done by for example implementing an unscented Kalman filter, complementing the feature extraction to also use other methods or by including feature descriptors.
7 Acknowledgements

First of all, I would like to thank my supervisor Jonas Nygård at FOI for guidance and support in this project. I would also like to thank Jonas Nordlöf for great help with all practical things, measurements and fruitful discussions regarding the project. As a whole I would like to thank FOI for the possibility of doing this interesting master thesis at the department of sensor informatics. I would also like to thank my subject reader Mikael Sternad at Uppsala University.

Furthermore, I would like to thank my fellow thesis workers for providing a very enjoyable and fun working environment here at FOI.
8 References


[17] Randall Smith, Matthew Self, Peter Cheeseman, A stochastic map for uncertain spatial relationships, *Proceedings of the 4th international symposium on Robotics Research*, p. 467-474, Univ. of California, Santa Clara, California, USA, 1988
[18] José Luis Blanco Claraco, A tutorial on SE(3) transformation parameterizations and on-manifold optimization, *Technical report from University of Malaga*, Univ. of Malaga, Nov 2017


[28] [http://pointclouds.org/](http://pointclouds.org/), June 2018

[29] Jason M. O’Kane, A gentle introduction to ROS (version 2.1.3), University of South Caroline, Department of Computer science and Engineering, Columbia April 2016, ISBN 978-14-92143-23-9

Appendix

A  Transformation Jacobians in 3D with quaternions

The corresponding Jacobians for the composition and inversion operators are described here in 3 dimensions with rotation represented with quaternions.

### Composition

The inversion Jacobian is defined as,

\[
\frac{\partial f_{qw}[p_1,(x_2,y_2,z_2)]}{\partial p_1} = \begin{bmatrix}
\frac{\partial f_{qw}[p_1,(x_2,y_2,z_2)]}{\partial q_{w2}} \\
\frac{\partial f_{qw}[p_1,(x_2,y_2,z_2)]}{\partial q_{x2}} \\
\frac{\partial f_{qw}[p_1,(x_2,y_2,z_2)]}{\partial q_{y2}} \\
\frac{\partial f_{qw}[p_1,(x_2,y_2,z_2)]}{\partial q_{z2}}
\end{bmatrix}
\]

(41)

with \( \frac{\partial f_{qw}[p_1,(x_2,y_2,z_2)]}{\partial p_1} \) defined by,

\[
\frac{\partial f_{qw}[p_1,a]}{\partial (qw,qx,qy,qz)} = 2 \begin{bmatrix}
-q_2 a_y + q_y a_z \\
-q_2 a_y + q_y a_z \\
-q_2 a_y + q_y a_z \\
-q_2 a_y + q_y a_z
\end{bmatrix}
\]

(42)

\[
\frac{\partial q'[qw,qx,qy,qz]}{\partial (qw,qx,qy,qz)} = \frac{1}{(q_w^2 + q_x^2 + q_y^2 + q_z^2)^{3/2}} \begin{bmatrix}
q_x^2 + q_y^2 + q_z^2 \\
-q_x q_w + q_w q_x \\
-q_y q_w + q_w q_y \\
-q_z q_w + q_w q_z
\end{bmatrix}
\]

(43)

The second Jacobian related to the composition is defined as follows,

\[
J_{2\oplus}(p_1,p_2) = \begin{bmatrix}
\frac{\partial f_{qw}[p_1,(x_2,y_2,z_2)]}{\partial [x_2,y_2,z_2]} \\
0_{4x3}
\end{bmatrix}
\]

(44)

with \( \frac{\partial f_{qw}[p_1,(x_2,y_2,z_2)]}{\partial [x_2,y_2,z_2]} \) is defined as,

\[
\frac{\partial f_{qw}[p_1,(x_2,y_2,z_2)]}{\partial [x_2,y_2,z_2]} = 2 \begin{bmatrix}
\frac{1}{2} - q_y^2 - q_z^2 \\
\frac{1}{2} - q_x^2 - q_z^2 \\
\frac{1}{2} - q_x^2 - q_y^2 \\
\frac{1}{2} - q_x^2 - q_y^2
\end{bmatrix}
\]

(45)

### Inversion

The inversion Jacobian is defined as,
\[ \mathbf{J}_{\mathcal{E}}(p_1, p_2) = \begin{bmatrix} \partial f_{qwi}(\{0,0,0\}^T, q) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \]  

(46)

where the last sub-Jacobian is defined by,

\[ \frac{\partial f_{qwi}(a,q)}{\partial q} = \begin{bmatrix} 2q_y^2 + 2q_z^2 - 1 & -2q_wq_z - 2q_xq_y & 2q_wq_y - 2q_xq_z & 2q_wq_x - 2q_yq_z \\ 2q_wq_z - 2q_xq_y & 2q_x^2 + 2q_z^2 - 1 & -2q_wq_x - 2q_yq_z & 2q_wq_y - 2q_xq_z \\ -2q_wq_y - 2q_xq_z & 2q_wq_x - 2q_yq_z & 2q_x^2 + 2q_y^2 - 1 & 2q_wq_x - 2q_yq_z \\ 2q_wq_z - 2q_xq_y & 2q_wq_x - 2q_yq_z & 2q_wq_x - 2q_yq_z & 2q_x^2 + 2q_y^2 - 1 \end{bmatrix} \]  

\[ \cdot \left( \frac{\partial f_{qwiw}(a,q)}{\partial q} \right) \]  

(47)

with,

\[ \frac{\partial f_{qwiw}(a,q)}{\partial q} = \begin{bmatrix} 2q_y\Delta z - 2q_z\Delta y & 2q_y\Delta y + 2q_z\Delta z & 2q_x\Delta y - 4q_y\Delta x + 2q_w\Delta z & 2q_z\Delta z - 2q_w\Delta y - 4q_z\Delta x \\ 2q_z\Delta x - 2q_x\Delta z & 2q_y\Delta x - 4q_z\Delta y - 2q_w\Delta z & 2q_x\Delta x + 2q_z\Delta z & 2q_w\Delta x - 4q_z\Delta y + 2q_y\Delta z \\ 2q_z\Delta y - 2q_y\Delta x & 2q_z\Delta x + 2q_w\Delta y - 4q_z\Delta z & 2q - z\Delta y - 2q_w\Delta x + 4q_z\Delta z & 2q_z\Delta x + 2q_y\Delta y \\ -2q_wq_z - 2q_xq_y & 2q_wq_x - 2q_yq_z & 2q_wq_x - 2q_yq_z & 2q_x^2 + 2q_y^2 - 1 \end{bmatrix} \]  

\[ \cdot \left( \frac{\partial (qw', qx', qy', qz')(qw, qx, qy, qz)}{\partial (qw, qx, qy, qz)} \right) \]  

(48)

where the second term is the quaternion normalization defined in equation 43.
B Parameter settings

B.1 FeatureExtraction

- Scan discs used for extraction: 4 - 12 (of the 16 in total)
- Number of scan discs that a quasi edge need to be found in for it to be called an edge: 2
- Maximum error when segmenting: 0.1
- Maximum error when fitting lines to segments: 9
- Maximum fitting error with Newton-Taubin: 0.005
- Maximum diameter of the circular objects: 0.9

B.2 Feature Association

- Search radius for clutter tree: 1 m
- Search radius for association tree: 1 m
- Number of clutter hits before considered consistent: 4

B.3 Uncertainty matrices in the EKF

Sensor noise:

\[
R = \begin{bmatrix}
0.0064 & 0 & 0 \\
0 & 0.0064 & 0 \\
0 & 0 & 10000
\end{bmatrix}
\]

Process noise:

\[
Q = \begin{bmatrix}
0.0176 & 0 & 0 \\
0 & 7.0448e-08 & 0 \\
0 & 0 & 10000
\end{bmatrix}
\]