Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Vebjørn Fossheim Eklo

SLAM-Driven Localization and Registration

Master's thesis in Cybernetics and Robotics
Supervisor: Thor Inge Fossen
June 2019

NTNU
Norwegian University of
Science and Technology

Vebjørn Fossheim Eklo

SLAM-Driven Localization and Registration

Master's thesis in Cybernetics and Robotics
Supervisor: Thor Inge Fossen
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Faculty of Information Technology
and Electrical Engineering
Department of Engineering Cybernetics

MASTER'S THESIS PROJECT DESCRIPTION

Name: Vebjørn Fossheim Eklo
Department: Engineering Cybernetics
Thesis title: SLAM-Driven Localization and Registration

Thesis Description:

The goal of the project is to test software using open source algorithms for Simultaneous Location and Mapping (SLAM) using simulated and experimental data generated by a Lidar. In addition, localization by sensor fusion of inertial measurements and GPS should be addressed.

The following items should be considered in more detail:

1. Literature study and review of SLAM algorithms.
2. Implementation of the Ouster Lidar system in ROS (Robotic Operating System) and testing of open-source SLAM algorithms.
3. Simulation of open source Lidar data sets for testing of registration and mapping using different matching algorithms in Matlab.
4. Tuning of parameters for point-cloud registration using the HDL Graph SLAM algorithm.
5. INS localization by IMU and GPS sensor fusion using a Kalman filter.
6. Conclude your findings in a report.

Start date: 2019-01-07
Due date: 2019-06-03

Thesis performed at: Department of Engineering Cybernetics, NTNU
Supervisor: Professor Thor I. Fossen, Dept. of Eng. Cybernetics, NTNU

Abstract

Simultaneous Localization and Mapping is the problem of creating a map of the surroundings and simultaneously localize the robot/vehicle within this map. In mobile robotics and self-driving applications, this is one of the essential challenges that need to be solved. The use of laser range scanners is a common way of mapping the surroundings, and a LIDAR was supplemented by NTNU for real-world testing.

This thesis presents a comparison of different methods of registering point clouds. The Iterative Closest Point and Normal Distribution Transform method differ in speed and accuracy, where the first proved best in simulations, but the second best when registering point clouds from the supplied LIDAR.

Sensor fusion of an IMU and GPS is included as part of an Inertial Navigation System that utilizes an Extended Kalman filter to update the filter states and covariance. The result is a pose estimate that tracks a ground truth trajectory well but suffers from drift over longer periods because of sensor biases and the lacking of a model for the vehicle dynamics. The INS solution proves less computationally demanding than tested SLAM solutions for pose estimation. The ambit of this thesis is a building block for further development in the field of Simultaneous Localization and Mapping.

Sammendrag

SLAM er et problem som omhandler prosessen med å opprette et kart over omgivelsene og samtidig lokalisere roboten/kjøretøyet i dette kartet. I robotikken og selvkjørende applikasjoner er dette en av de viktigste utfordringene som må løses. Bruk av laser skannere er en vanlig måte å kartlegge omgivelsene på, så en LIDAR ble supplert av NTNU for sanntidstesting.

Denne oppgaven presenterer en sammenligning av ulike metoder for registrering av punktskyer. ICP og NDT er to metoder som varierer i hastighet og nøyaktighet, hvor den første viste seg best i simuleringer, mens det andre viste seg best når man registrerer punktskyer fra LIDARen.

En sensorfusjon av en IMU og GPS er inkludert som en del av et navigasjonssystem som bruker et Extended Kalman filter for å oppdatere filtertilstandene og kovariansen. Resultatet er et posisjonsestimat som sporer en forhåndsmålt bane. Estimatet lider av drift over lengre perioder på grunn av sensor bias og mangelen på en skikkelig modell for kjøretøyet. Navigasjonsløsningen viser seg å være mindre krevende enn SLAM-løsninger for posisjonsestimering. Omfanget av denne avhandlingen er en byggestein for videreutvikling innen SLAM.

Preface

This Master thesis in its finished product is the culmination of my work at the Norwegian University of Science and Technology. The choice of project is supervised by Thor Inge Fossen, Professor of Guidance, Navigation and Control at the NTNU Department of Engineering Cybernetics. The thesis is a summary of my findings, and the methods used to run Simultaneously Localization and Mapping using a laser range scanner. The study and completion of this master thesis were done during the spring of 2019.

This thesis is based on the report from the coursework TTK4551 - Engineering Cybernetics, Specialization Project [34], presenting a literature and method review of Simultaneously Localization and Mapping. Aspects I found important from that report is included in this master thesis. This especially apply to chapter 3 and 4 where general background theory for the SLAM problem is described. This thesis aims to further develop the subjects touched upon in that report, and test the suggested solutions using a LIDAR provided by the Department of Engineering Cybernetics at NTNU.

As I look back at the last few months I would like to give a special thanks to my supervisor Thor Inge Fossen, for always replying to any questions I might have. Even during weekends. A special gratitude should also be given to my fellow students during this period for all the backup, conversations and fun times we had.

Contents

List of Figures

1 Introduction

This introductory chapter will briefly provide context for the material/results presented in this report and give motivation and a description of this master thesis contribution along with an outline of the report structure.

1.1 Motivation

Many applications of robotics, such as industrial automation, the aerial industry, and the car industry are in a revolutionary transform when it comes to autonomous systems [26]. Self-driving vehicles are researched by many manufacturers, but there are still some challenges to this promising future. To have an effective control system, a map of the dynamic environment and reliable position estimates is essential. The problem is that in order to construct a map, accurate robot position estimates are needed, and to accurately measure the position estimate, the robot needs a map.

Simultaneously Localization and Mapping or SLAM is probably the most common method to solve this problem. But SLAM is not easy to implement as it is a complex system, especially when it needs to operate in dynamic environments. The motivation behind this thesis is, therefore, to test the different parts of SLAM on a laser range scanner, or LIDAR, to best be able to tell which methods and solutions that work best with the LIDAR. Apart from this, the thesis aims to answer some of the following research questions:

- *R1: How can 3D SLAM using LIDARs be a sustainable solution to the SLAM problem?*
- *R2: What is more important, speed or accuracy when it comes to mapping?*
- *R3: How can SLAM be used to improve the ability to estimate the vehicles pose?*

1.2 Background and Contributions

This thesis aims to describe and make the reader understand what the SLAM problem is. It aims to discover the essential parts of registration of a map and localization within this map. It describes how to implement a laser range scanner in ROS and how to connect the sensor to a SLAM algorithm. Registration and localization methods are simulated, and the simulated results are used as an indicator of how to tune some of the most important parameters of SLAM. The results of the simulation are tested on a data-set recorded in an indoor and outdoor environment.

The Norwegian University of Science and Technology have provided the project with the Ouster 1, 16 beam 3D LIDAR. It has been used to record data samples, and have been integrated with a SLAM system. The Matlab simulations used for registration and localization use an open-source data-set containing high-quality measurements. How they were included and used is well described and cited in the thesis.

1.3 Outline

The thesis is organized as follows. In Chapter 2 brief background theory of filtering, registration, and localization is presented to help the reader understand the later chapters. Chapters 3 and 4 thoroughly explains the SLAM problem and the most important paradigms to solve this problem. Following these chapters is the main contribution of this thesis. Firstly chapter 5 presents the used data-sets and hardware, before chapter 6, 7 and 8 presents the main implementations, the results, and an evaluation of these results. Chapter 9 tries to answer some of the research questions, and lastly, chapter 10 presents possible future work areas of the thesis.

Figure 1 illustrates a block diagram of the most important topics of SLAM, and these are repetitiously brought up as sections in the different chapters throughout the thesis. The colored fields in the figure represent the parts that were researched and implemented.

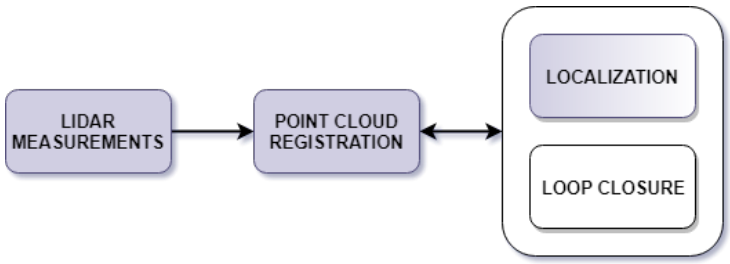


Figure 1: System Outline

Localization was not fully tested, and loop closing was a subject not touch upon at all. The intention was to implement all the blocks in the figure, but limited time, priorities, knowledge, and resources limited the outline of the report to the ones mentioned.

2 Background Theory

This chapter mentions the most important aspects of the background theory that is necessary to understand the terms and solutions presented later in the report. Filtering, registration and localization theory is presented, and a chapter on ROS and grid theory is also included.

2.1 Recursive Bayes estimation

Algorithm 1 : **Bayes filter(**$bel(x_{t-1}), u_t, z_t$**) :**

1: **for all** x_t do
2: $\bar{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) \cdot bel(x_{t-1})dx$
3: $bel(x_t) = \eta \cdot p(z_t|x_t) \cdot bel(x_t)$
4: **end for**
5: **return** $bel(x_t)$

The predominant approach for state estimation in probabilistic robotics is the Bayes filters. The method estimates the probability distribution $bel(x_t)$ over the state x_t recursively based on the most recent control inputs u_t, measurements z_t and the previous probabilistic estimate. In the Bayes filter algorithm[1] this is represented through a prediction step in line 2 and an update step in line 3. The previous probability distribution is extrapolated based on the control input u_t and the previous state x_{t-1}, in the update step the measurement is used to improve the estimate. Because of this, the Bayes filters do not just "guess" the state, they calculate the probability that any state is correct [31].

2.2 Gaussian filters

2.2.1 Kalman Filter

The Kalman filter is probably the most used implementation of the Bayes filter and has been regarded as the best solution to many tracking and data prediction solutions. The filter is constructed to minimize the mean square likelihood, with the purpose of extracting the required information from a signal, ignoring everything else. A cost or loss function is used to measure how well the filter performs.

Algorithm 2 : **Kalman filter**$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$ **:**

1: $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
2: $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
3: $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
4: $\mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t)$
5: $\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$
6: **return** μ_t, Σ_t

In the Kalman filter, the probability distribution is represented by the mean μ_t and the covariance Σ_t. Unlike the Bayesian filter, the Kalman filter needs the probability distributions to be Gaussian. The predicted distributions in the algorithm[2] is performed in the first two steps, updating the new best estimate $\bar{\mu}_t$ by incorporating the control input u_t and the previous best estimate μ_{t-1}, and updating the new uncertainty $\bar{\Sigma}_t$ from the previous uncertainty Σ_t and some additional uncertainty from the environment R_t. In step 3 through 5, the estimates are refined using the measurements z_t. K_t is called the Kalman gain and decides to what the degree the measurements should be incorporated into the new state estimates. The mean in line 4 is adjusted in proportion with the Kalman gain and the difference between the measurements z_t and the expected measurements $C_t \bar{\mu}_t$. In the last step, the covariance is adjusted with regards to the Kalman gain. Using the Kalman filter any linear system can be modeled pretty accurately. For non-linear system, an Extended Kalman filter can be used, which works by linearizing the predictions and measurements about their mean.

2.2.2 Information Filter

As for Kalman filters, Information filters represents the belief by a Gaussian Distribution. But the key difference between the two is in the way the belief is represented. Whereas the Gaussian is represented by the mean and covariance in the Kalman filter, the distribution is represented by a canonical form comprised of an information matrix and an information vector. The difference in representation leads to different update equations. What is complex in one representation is simple in the other, and vice versa. Canonical and moments (mean and covariance) are therefore often considered dual to each other, and thus are the Information and Kalman filter [31].

Algorithm 3 : **Information filter**($\xi_{t-1}, \Omega_{t-1}, u_t, z_t$) **:**

1: $\bar{\Omega}_t = (A_t \Omega_{t-1}^T A_t^T + R_t)^{-1}$
2: $\bar{\xi}_t = A_t \Omega_{t-1}^T \xi_{t-1} + B_t u_t$
3: $\Omega_t = C_t^T Q_t^{-1} C_t + \bar{\Omega}_t$
4: $\xi_t = C_t^T Q_t^{-1} z_t + \bar{\xi}_t$
5: **return** ξ_t, Ω_t

Like the Kalman filter, the information filter updates in the same two steps. The difference is that the input is a Gaussian in a canonical form ξ_{t-1} and Ω_{t-1} representing the distribution. As all Bayesian recursive filters, the input includes control and measurements. The outputs are the updated parameters ξ_t and Ω_t of the updated Gaussian. These parameters in the canonical form is a quadratic function of the negative logarithm of the Gaussian. This is e.g. used in the calculation of the Mahalanobis distance.

2.3 Measurements

Probabilistic robotics explicitly models the noise in sensor measurements, this to account for the inherent uncertainty in the sensors. The measurement model is usually defined as a probability distribution $p(z_t|x_t, m)$, where z_t is the measurement at time t given the robots pose x_t and the map m of the environment. Sensor models can be

very complicated to model as the state variables may be unknown. By modeling the measurement process as a conditional probability density instead of a deterministic function, the uncertainties in the models can be accommodated. This is probably the biggest advantage of probabilistic robotics.

2.3.1 Mapping

The map of the environment is a list of objects in the environment and their locations. The environment may be comprised of landmarks, objects, surfaces, etc. $\boldsymbol{m}$ describes the location of these. Put together, the landmark vectors create the map.

$$\boldsymbol{m} = \{m_1, m_2, ..., m_N\} \tag{1}$$

In robotics, maps are usually seen as either feature-based or location-based maps. In feature-based maps, the properties of m_n consist of the feature properties and the cartesian location of the features. For location-based maps, the mapping matrix consists of specific locations in the environment, e.g. a point cloud.

2.3.2 Beam measurement

Laser range finders measure range along a beam. The sensor shines a light at a surface and measures the times it takes for the light beam to return to the source. Since light moves at constant speed, the distance can be measured with high accuracy but there is still measurement errors that need modeling including small measurement noise, errors due to unexpected objects, detection failures errors, and random noise. The desired model of distribution $p(z_t|x_t, m)$ is therefore a mixture of densities corresponding to different measurement errors [6].

Algorithm 4 : **Beam range finder model**(z_t, x_t, m) **:**

1: $q = 1$
2: **for** $k = 1$ to K **do**
3: compute z_t^{k*} for the measurement z_t^k using ray casting
4: $p = z_{hit} \cdot p_{hit}(z_t^k | x_t, m) + z_{rand} \cdot p_{short}(z_t^k | x_t, m) + z_{max} \cdot p_{max}(z_t^k | x_t, m) + z_{rand} \cdot p_{rand}(z_t^k | x_t, m)$
5: $q = q \cdot p$
6: **return** q

2.4 Localization

In mobile robotics, localization is the problem of determining the pose of the robot relative to a given map of the environment, often referred to as position estimation. In robotics, this is one of the most important aspects of the modeling as almost all robotic systems require knowledge of the location of the robot and the objects that are handled. Localization can be seen as a problem of coordinate transformation. Maps are described in global coordinates, independent from the robots pose. Pose estimation is the process of establishing a relation between the global coordinates of the map, and the local coordinates of the robot system. If the local coordinates of the robot are known in the global coordinates of the map, it is possible to express the measured surroundings in the same coordinate system. The problem with localization is that most sensors used for pose estimation have noisy measurements. This means that the system has to integrate measured data over time to get a sufficient estimation of the robots local coordinates.

2.4.1 Markov localization

The straightforward application of Bayes filters to the localization problem is the Markov localization. The algorithm for Markov localization is very similar to the Bayes filter algorithm, but it requires a map m as input. The map is used in both the measurement model $p(z_t | x_t, m)$ and in the motion model $p(z_t | u_t, x_{t-1}, m)$. The Markov localization algorithm manages to estimate the pose in a global environment where

the initial pose $bel(x_0)$ of the robot is unknown by initializing the belief with a uniform distribution over the space of all legal poses in the map [10].

Algorithm 5 : **Markov localization**($bel(x_{t-1}, u_t, z_t, m)$) **:**

1: **for all** x_t **do**
2: $\bar{bel}(x_t) = \int p(x_t|u_t, x_{t-1}, m) \cdot bel(x_{t-1})dx$
3: $bel(x_t) = \eta \cdot p(z_t|x_t, m) \cdot \bar{bel}(x_t)$
4: **end for**
5: **return** $bel(x_t)$

As the robot moves through the environment, the belief gets updated in line 4 of the algorithm[5] for every new measurement observation the sensor makes, and the motion model in line 3 updates the belief with regards to the continuum of recent poses. As with the Bayes filter, the Markov algorithm calculates the probability that any state is correct. After a sufficient number of observations, the belief has a distribution represented close to the robots real pose.

2.4.2 EKF Localization

The Extended Kalman filter localization is a Landmark-based localization algorithm with a prediction and a correction step represented by the mean μ_t and the covariance Σ_t. The algorithm [6] is derived from the Extended Kalman filter. At initialization the required inputs are the estimate of the robots pose with mean μ_{t-1}, covariance Σ_{t-1}, control input u_t, a map m, a set of observed features z_t and the correspondence variables c_t.

The first three lines of the algorithm represent the motion model, where the first line calculates the predicted pose of the robot, while $\bar{\Sigma}_t$ is an ellipse representing the uncertainty of the predicted pose. This uncertainty increases as the robot progress. The rest of the algorithm represents the measurement update sequence, where the algorithm loops through all possible i observed at time t. The Kalman gain K_t^i is calculated from the predicted measurements $\hat{z}_t^i$ and Jacobian H_t^i of the measurement model. Lastly, the sum of all updates is applied to obtain the new pose estimate in

Algorithm 6 : **EKF localization(** $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, c_t, m$ **) :**

1: $\bar{\mu}_t = \mu_{t-1} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\delta t) \\ \omega_t\Delta t \end{pmatrix}$

2: $G_t = \begin{pmatrix} 1 & 0 & \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 1 & \frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t\Delta t) \\ 0 & 0 & 1 \end{pmatrix}$

3: $\bar{\Sigma}_t = G_t\Sigma_{t-1}G_t^T + R_t$

4: $Q_t = \begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix}$

5: for all observed features $z_t^i = (r_t^i \quad \phi_t^i \quad s_t^i)^T do$

6: $j = c_t^i$

7: $\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} m_{j,x} - \bar{\mu}_{t,x} \\ m_{j,y} - \bar{\mu}_{t,y} \end{pmatrix}$

8: $q = \delta^T\delta$

9: $\hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ atan2(\delta_y, \delta_x) - \bar{\mu}_{t,\theta} \\ m_{j,s} \end{pmatrix}$

10: $H_t^i = \frac{1}{q}\begin{pmatrix} \sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 \\ \delta_y & \delta_x & -1 \\ 0 & 0 & 0 \end{pmatrix}$

11: $K_t^i = \bar{\Sigma}_t H_t^{i,T}(H_t^i\bar{\Sigma}_t H_t^{i,T} + Q_t)^{-1}$

12: end for

13: $\mu_t = \bar{\mu}_t + \sum_i K_t^i(z_t^i - \hat{z}_t^i)$

14: $\Sigma_t = (I - \sum_i K_t^i H_t^i)\bar{\Sigma}_t$

15: return μ_t, Σ_t

line 13 and 14. As more landmarks with known positions are observed, the ellipse $\bar{\Sigma}_t$ representing uncertainty in the predicted pose shrinks, making the estimated pose better.

2.4.3 Inertial Navigation System(INS)

Inertial navigation is a self-contained navigation technique in which measurements provided by accelerometers and gyroscopes are used to track the position, velocity and orientation of an object relative to a known start position, velocity and orientation[12]. INS provides a full 6DOF navigation solution. The navigation state vector and inertial measurement input vector can be defined in equation (2) and (3).

$$x_t = \begin{bmatrix} p_t \\ v_t \\ q_t \end{bmatrix} \in R^{10} \tag{2}$$

$$u_t = \begin{bmatrix} s_t \\ \omega_t \end{bmatrix} \in R^{6} \tag{3}$$

where
p_k is position in [m], v_k is velocities in $[m/s]$, and q_k is the quaternion representation of position, velocity and orientation of the navigation system at time t. S_t denotes the difference between the inertial and gravitational acceleration $[m/s^2]$, and ω_t is angular velocity $[rad/s]$. By neglecting discretization and quantization errors in the navigation equations, the INS will track position, velocity and orientation perfectly if there is no error in the IMU readings. In real life scenarios, this is unfortunately not the case, and the measurement errors will cause the position and velocity estimates to grow without bounds. One way to improve the performance is to fuse the INS with a position estimate usually from a GNSS/GPS. The fused INS solution may then look something like figure 2. What type of filtering algorithm used to estimate the errors in

the measurements and navigation parameters can differ, but the Kalman Filter(2.2.1) is a well-used solution in INS filters.

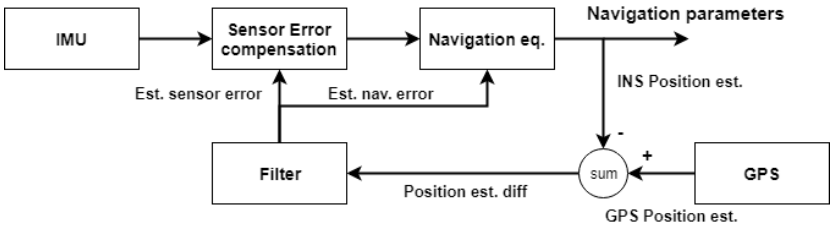


Figure 2: INS fusion closed loop

2.5 Registration

3D Registration is the process of consistently aligning two or more sets of three-dimensional points. In most applications, these sets, often called point clouds, are acquired by 3D scanners from different viewpoints. The registration process leads to the relative pose (position and orientation) between scans in the same coordinate frame, such that overlapping areas between the point clouds match as well as possible[27]. Once aligned, the individual point clouds are merged into a single one, so that techniques for extracting information can be applied.

2.5.1 Coherent Point Drift (CPD)

A robust probabilistic multidimensional point set registration algorithm that considers the alignment of two point sets as a probability density estimation problem. CPD uses a Gaussian Mixture Model(GMM) to represent one of the point clouds. GMM is a method much used in object tracking where tracking of multiple objects is needed. It creates a Gaussian probability distribution combining several clusters of data[11]. The CPD methods use the GMM-model to create centroids, figure3, which is the mean position of a set of points in the cloud. The other point cloud in the CPD-algorithm is the current

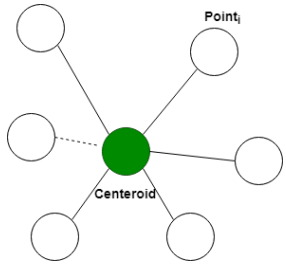


Figure 3: Centroid

observed data points. The GMM-centroids is found by iteratively maximizing the likelihood, finding a posterior probability of centroids, which provides the algorithm with a correspondence probability to the data set. The key to the algorithm is to make all the centroids move coherently as a group, preserving their topological structure, hence the name Coherent Point Drift[3].

2.5.2 Iterative Closest Point (ICP)

Probably the most popular registration algorithm is the ICP-algorithm. It minimizes the difference between two sets, or point clouds, by iteratively finding correspondences between the two sets of points [4]. The two different point clouds are called the *Target* and the *Source*. In each iteration, a point in the source point cloud is matched with its closest neighbour of the target point cloud using a search algorithm. A point-to-point distance metric minimization is performed to find the estimate of the transformation between the two points, this to best be able to align them. This step may also involve weighting points and rejecting outliers prior to the alignment, for better performance and quick calculation. The points in the source point cloud are then transformed using the obtained transformation before the method iterates a new set of neighbour points into the sequence.

A_i and B_i from algorithm[7] represents the target and source point cloud respectively. M_0 is the initial transformation between the two sets. If no initial transformation

Algorithm 7 : **Iterative Closest Point**(A_i, B_i, M_0) :

1: $M = M_0$
2: **while** not converged **do**
3: **for** $i = 1$ to N **do**
4: c_i = FindClosestPointInA($M \cdot B_i$)
5: **if** $||c_i - M \cdot b_i|| \leq d_{max}$ **then**
6: $\omega_i = 1$
7: **else**
8: $\omega_i = 0$
9: **end if**
10: **end for**
11: $M = \underset{M}{\operatorname{argmin}}(\sum_i \omega_i (M \cdot b_i - c_i)^2)$
12: **end while**
13: **return** M

information is obtainable M_0 is set to identity. Line 4 of the algorithm is the search algorithm to find the set of points which are closest to the points in the source. d_{max} is the weighting parameter mentioned, deciding the trade-off between accuracy and convergence, removing outliers. The point-to-point minimization is lastly performed to estimate the transformation M between the points. If the resolution of the point clouds is sufficient enough a point-to-plane error metric minimization may be used, where a tangent plane is constructed for every point in the target point cloud, producing a normal vector connecting the two clouds [19].

2.5.3 Normal-Distributions Transform(NDT)

In comparison to the CDP, the Normal-Distribution Transform approach applies a normal distribution to subdivided cells of the point cloud, hereby modeling the probability of a measured point[5]. The result of the transformation is a continuously differentiable probability density, that can be used to match another point cloud scan by Newton's method. The major advantage is that there is no need to establish an explicit correspondence between points or features to produce an accurate probability

distribution. It can establish a piecewise continuous and differentiable probability from a single scan. See section 6.2.2 for details of the algorithm.

2.5.4 Voxel and Occupancy grids

A voxel is a volumetric pixel, or the 3D equivalent of a pixel. It is a volume element that represents a specific grid value in 3D space. E.g. a Rubik's cube can be seen as a squared grid consisting of 3^3 voxels. Voxels do not contain information about their axis coordinates, but rather they keep some information about their relative location in relation to nearby voxels and are considered single points in the 3D space.

Many robot applications require a 3D model of the environment, along with a probabilistic representation, modeling of free, occupied, and unmapped areas, with the addition of runtime efficiency [14]. A voxelgrid is a great way of representing these areas as the environment can be mapped by voxels storing relative information about its neighbours being able to determine free, occupied and unmapped areas. It is usually possible to adjust the resolution in voxelgrids by deciding the length of the sides in the voxel cube. This way the 3D model of the environment can be adjusted accordingly.

Occupancy grid maps are similar to the voxelgrid, but usually, represent the 2D environment as binary random variables occupying some space. An obstacle in the environment is represented in a grid by occupying cells that try to recreate the shape of the obstacles. E.g. a chess board can be seen as an occupancy grid, where each square represents a binary random variable representing the percentage of an obstacle at this location. As for voxelgrids, the resolution of the occupancy grid map can be adjusted, hereby being able to represent obstacles more accurate.

2.6 ROS

Robot Operating System (ROS) is an open-source, meta-operating system. ROS was made to support the reuse of code in robotics research and development. The point is to have an environment where many modules can run concurrently, and communicate without being aware of each other. Meta-operating system means that ROS is built and based on Ubuntu Linux, and shares it's process management system, file system, user interface and programming utilities. In addition, it also provides tools and libraries for obtaining, building, writing and running code across multiple computers. In other words, instead of redefining and changing the programming vocabulary and grammar, ROS only adds features and libraries to the traditional C++ program. You can therefore simply use some function calls and classes instead of rewrite major parts of code.

In ROS, programs are called nodes. Nodes can communicate with each other by sending messages. These messages are sent to what is called a message topic. A topic of a message must have a defined message type. This is so that ROS can convert them from data structures to byte streams at sender's end, transport them to the recipient, and then convert them back to data structures. To receive and send certain topic of messages the nodes needs to include what is called ROS Subscriber and ROS Publisher functions. These are functions included in the ROS library and are necessary for the nodes to communicate. An example of a node used in ROS is the *ouster_ros* node that can visualize recorded or real-time point cloud messages. Most open-source robot systems use ROS nodes to run different nodelets in the system. The nodelet package in ROS is designed to provide a way to run multiple algorithms in the same process with zero copy transport between algorithms.

A launch-file is a file type that launches multiple ROS nodes locally and remotely. This file can also change certain aspects of the code in the nodes by replacing topic names and setting parameter values. This is useful when you wish to connect different types of sensors to your code, by altering just the necessary parameters in the launch file, rather than changing the code itself.

3 SLAM Background Theory

3.1 The SLAM problem

The Springer Handbook of Robotics gives a great statement of what the SLAM problem represent [30]. A mobile robot roams the unknown environment, starting at an initial location. Its motion is uncertain, making it gradually more difficult to determine its current pose in global coordinates. As it roams, the robot senses its environment with a noisy sensor. The SLAM problem is the problem of building a map of the environment while simultaneously determine the robot's position relative to this map given noisy data.

Simultaneously Localization and Mapping or SLAM has its origin in studies done by Smith, Self, and Cheeseman back in 1986 on estimation for spatial relationships and covariance between coordinate frames [28]. They introduced a method on how to localize an object relative to another by manipulating the uncertainty associated with spatial information, in the form of sensed relationships, prior constraints, and relative motion. They made it possible to estimate the probability of certain events, based on the uncertainty of the robot and the surrounding objects relative location. Smith, Self, and Cheeseman made an effort to develop the methods in the context of state-estimation and filtering theory [29] to provide a solid basis for numerous extensions. This led to the introduction of SLAM in 1991 by Leonard and Durrant-Whyte [18].

Leonard and Durrant-Whyte introduced an algorithm for localization, based on current state, position estimates, and sensor observation. They were able to develop a mobile robot localization system that integrated a variety of beacon observations as input to a Kalman filter to maintain a robust vehicle location estimate. Although their algorithm had a number of limitations, most importantly the need for a priori environment description and infinite data limits, they set the standard for further work with more computationally tractable algorithms.

Durrant-Whyte, Fellow and Bailey have written one of the most cited papers on the general SLAM topic [2]. To illustrate their points, figure 4 shows the general idea

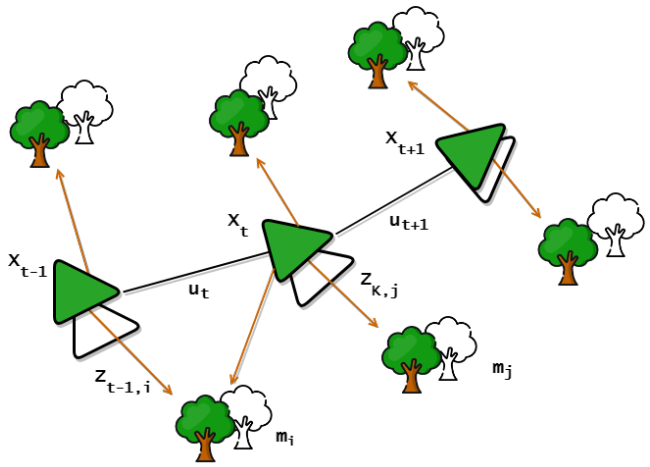


Figure 4: SLAM illustration. Objects in green are the real locations, the white objects are the estimated equivalents

of how SLAM works. The true locations are never known or measured directly in SLAM. Instead, observations are made between true robot and landmark locations, and an estimated result is presented in both robot and landmark location. In the illustration, a robot is moving through an environment taking relative observations of a number of unknown landmarks. In the figure, objects in green represent the real locations of robot and landmarks, while white is the estimated equivalents. To further understand this we need to define the notations used in figure 4 at time instant t. The notations used in SLAM varies in different papers, but the ones presented below are fairly common.

x_t: The state vector. Describes the robot's location and orientation. Put in consecutive order $X_{0:T}$, and we get the path of the robot.

u_t: The control vector. Describes the controls the robots receives in the prior state to drive it to the current state. Put in consecutive order $u_{1:T}$, and we get the robot controls.

m_i: Landmark vector. The environment may be comprised of landmarks, objects, surfaces, etc., This vector describes the location of these. Put together, the landmarks create a map m of the environment.

$z_{t,i}$: The observation vector. At time instant t, $z_{t,i}$ describes the observation of landmark m_i. Put in consecutive order, $z_{1:T}$ describes a set of observations, where e.g z_T could be a laser scan image.

The data that is obtained is not necessarily accurate. Uncertainty in the robot's motion and observations is common. SLAM is best described by a probabilistic approach. The robot or landmark is not seen in an exact position but has a probability distribution to its location. The use of the mean and variance from this distribution is a common way to estimate the positions in SLAM. Even though probabilistic distribution models vary depending on the system, there are mainly two forms of the SLAM problem. The full and online SLAM problem.

The full SLAM problem aims to estimate the entire path together with the map as shown in equation (4). Written this way, the full SLAM problem is the problem of calculating the probability distribution of the path x_T and the map m given observed sensor data and robot controls. Figure 5 illustrates a graphical model of the full SLAM problem. Algorithms for the full SLAM problem often process all data at the same time. The online SLAM problem seeks to recover only the recent pose and map, marginalizing out the previous poses. Usually done incremental one at a time, as shown in equation (5). A graphical model of the online SLAM problem would look familiar to the full problem in figure 5, but with an unknown colored field only for the current state and not the entire path.

$$\boldsymbol{p}(\boldsymbol{x}_{0:T}, \boldsymbol{m} \mid \boldsymbol{z}_{1:T}, \boldsymbol{u}_{1:T}) \tag{4}$$

$$\boldsymbol{p}(\boldsymbol{x}_t, \boldsymbol{m} \mid \boldsymbol{z}_{1:t}, \boldsymbol{u}_{1:t}) = \int_{\boldsymbol{x}_0} \dots \int_{\boldsymbol{x}_{t-1}} \boldsymbol{p}(\boldsymbol{x}_{0:t}, \boldsymbol{m} \mid \boldsymbol{z}_{1:t}, \boldsymbol{u}_{1:t}) d\boldsymbol{x}_{t-1} \dots d\boldsymbol{x}_0 \tag{5}$$

$$p(x_t \mid x_{t-1}, u_t) \tag{6}$$

$$p(z_{t+1} \mid x_{t+1}, m) \tag{7}$$

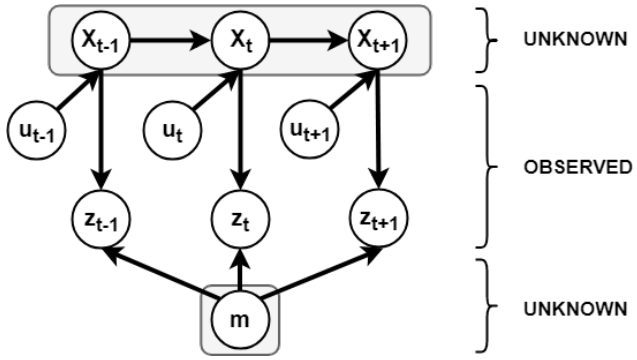


Figure 5: Graphical Model of the full SLAM-Problem

These two generalizations of the SLAM problem is usually structured into two models. The motion model and the observation model. The motion model describes the relative motion of the robot, while the observation model relates measurements with the robots pose. The models corresponds to the arcs in figure 6. Probabilistic motion models comprise the state transition probability shown in equation (6). It is an essential part of the prediction step of the Bayes filter which calculates the belief parameter that helps the robot infer its position and innovation. There are several different techniques to motion models designed for different use cases. Thrun, Fox, Burgard, and Dallaert introduced a robust motion model called Monte Carlo Localization in 2001 [32], able to estimate the pose in dynamic systems.

As for motion models, observation models offer vastly different models depending on the measurement devices and the system platform. The specifics of the model depends on the sensor. Cameras are best modeled by projective geometry, sonars by

describing the sound wave and its reflection on surfaces in the environment. The system planned in this paper uses a laser range scanner, the observation model should, therefore, be applicable to this sensor. Chapter 6 in Thrun [31] presents different approaches with different distributions for the probability estimates. Solutions using the RANSAC-algorithm is used in [9] and ICP (iterative closest point) methods in [15].

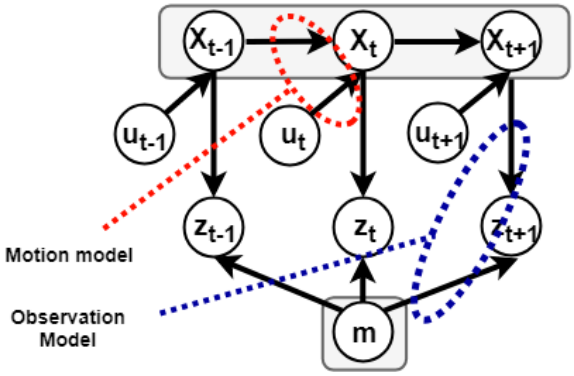


Figure 6: Grapical representation of the motion and observation model

SLAM is best explained as a concept rather than a simple algorithm, where all the blocks that are shown in figure 5 can be solved with different approaches. The basic idea is that after enough measurements, errors caused by vibrations and robot movements will be nullified. Each measurement is expected to be the same relative to the environment, meaning the position can be estimated over a large enough data set.

The data association problem is a problem worth mentioning when talking about SLAM. It is, in fact, one of the hardest problems to solve when you wish to track your position. The data association problem is the problem of deciding which target generated which observation. This would be easy in single-target tracking problems, but when the association is more ambiguous and tracks multiple observation in one scan or sample, the assignment of the observations become much harder to solve. One simple strategy to solve this is to pay attention to the measurement that is closest to

the prediction. This again becomes a problem when areas suffer from clutter in the landscape. A more sophisticated method is to keep track of multiple states/observations hypothesis and consider the possibilities of these hypotheses. This is a common method for particle filter strategies, where each particle can be a hypothesis of the current observation[21][22].

Another common problem in SLAM is the loop-closing problem. This is the problem of realizing that the observed measurements have been observed at an earlier stage. If one with certainty can conclude that this, in fact, has accrued, a loop-closing algorithm can be used to overcome the drift accumulated in the robot trajectory over time[17][8].

4 Simultaneous Localization and Mapping

There are several different paradigms to solve the SLAM problem. This chapter will review the three main paradigms used for SLAM, for which most others are derived. EKF-SLAM is historically the first introduced, but it has become less popular due to its limiting computational properties. Particle filtering is a popular method for online-SLAM and provides a perspective on addressing the data association problem in SLAM. Graph-Based SLAM is based on graphical representations and successfully applies sparse nonlinear optimization methods for solving the full SLAM problem.

4.1 EKF SLAM

As mentioned in chapter 2, Smith, Self, and Cheeseman were the first to propose the use of a single state vector to estimate the locations of a robot and a set of features in the environment. Together with this, an error covariance matrix represented the uncertainty in these estimates, including a correlation between the vehicle and feature state estimates. Based on this the EKF-SLAM algorithm was developed.

Simply put, the EKF-SLAM algorithm applies the extended Kalman Filter to online SLAM using maximum likelihood data association. As any EKF-algorithm, EKF-SLAM makes a Gaussian noise assumption for the robot motion and observed surroundings. The system covariance matrix mentioned above grows quadratically with the number of landmarks. Because of this, EKF-SLAM uses a feature-based map, composed of a relatively small number of point landmarks (<1000). The number of landmarks is preferably kept this low for computational reasons.

The EKF algorithm represents the robot estimate by a multivariable Gaussian shown in equation (8). The vector μ_t represents the state vector including the estimate of the robot's location and location of the features in the map. The dimension of μ_t depends on the system. If the system were a planar surface robot, the dimension would be $3 + 2N$, as three variables are needed to represent the location and $2N$ for the N landmarks. The matrix Σ_t is the error covariance matrix used as the expected error

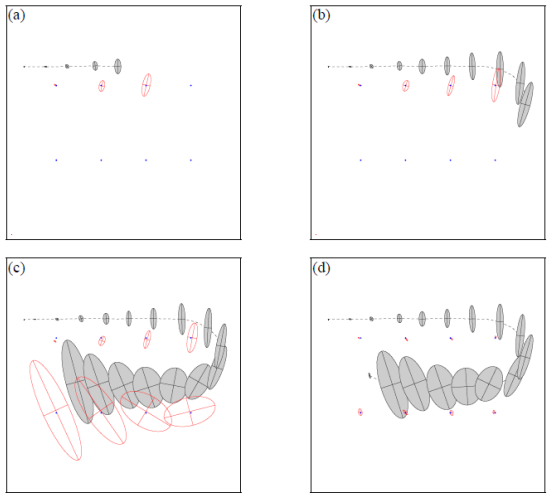


Figure 7: EKF-SLAM (*image courtesy of Michael Montemerlo, Standford University*)

for the guess μ_t. With the same example as above, the covariance matrix is of size $(3 + 2N) \cdot (3 + 2N)$, making the problem quadratic. The off-diagonal elements in the covariance matrix represent the correlations in the estimates of different variables. Since the robot's location is uncertain, and therefore the locations of the landmarks are uncertain, the nonzero correlations are included. In appendix A the complete algorithm for updating μ_t and Σ_t is shown, see [31] for an complete review. The algorithm uses an incremental maximum likelihood (ML) estimator to determine the correspondences.

$$\boldsymbol{p}(\boldsymbol{x}_t, \boldsymbol{m} \mid \boldsymbol{Z}_t, \boldsymbol{U}_t) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \tag{8}$$

Figure 7 illustrates the EKF-SLAM algorithm. The ellipses shown in the figure represents uncertainty in position (grey) and in landmark location (transparent). As the system moves, the pose uncertainty increases because of the errors in odometry mea-

surements. The system also senses landmarks in the environment and maps these with an uncertainty that combines the uncertainty in the measurement and the systems pose. This result in increasing uncertainty also for the landmark positions. The most important part happens in the last frame of the figure. Here the system observes the same landmark it did in the beginning. Through this observation, the systems state error is reduced, along with the uncertainty of the landmark locations. This happens because the uncertainty in the location estimates of the landmarks is vastly affected by the error in the position. The correlation in the covariance matrix spreads through the whole matrix and updates the previous landmark estimates. This effect is probably the most important in EKF-SLAM. As mentioned above, to solve the problem of uncertain data association, the system can use an ML-estimator to approximate which of the landmarks in the map most likely corresponds to the landmark just observed [33].

A key limitation to the method is the computational complexity. Sensor updates require time quadratic in the number of landmarks N to compute. This complexity stems from the fact that the covariance matrix maintained by the Kalman filters has $O(N^2)$ elements of which must be updated even if just a single landmark is observed. This limits the method for sets with a large number of landmarks [30][31].

4.2 Particle filtering

Particle filtering has become popular in recent decades. Particle filters are a recursive Bayes filter, where it represents a posterior through a set of particles. The posterior is the distribution of possible unobserved values conditional on the observed values. In SLAM, each particle is best taught as a concrete guess as to what the true value of the state may be, figure 8. By collecting many such guesses into a set of guesses, or a set of particles, the particle filter approximates the posterior distribution. Under mild conditions, the particle filter has been shown to approach the true posterior values as the particle set size goes to infinity. The particle filter is also not limited to Gaussian distributions.

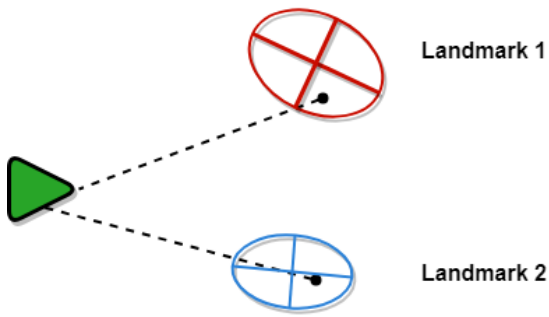


Figure 8: Particle illustration, making guesses of obstacle states

$$\boldsymbol{x} = (\boldsymbol{x}_{x:t}, \boldsymbol{m}_{1,x}, \boldsymbol{m}_{1,y} \ldots . \boldsymbol{m}_{N,x}, \boldsymbol{m}_{N,y}) \tag{9}$$

The key problem with the particle filter in the context of SLAM is that the path and the map tend to be large. Particle filters scale exponentially with the dimension of the underlying state space. Three or four dimensions are thus acceptable, but e.g. 100 dimensions are generally not, as seen from equation (9). The trick in making particle filters applicable to the SLAM problem is to use the particle set only to model the system's path, then each sample of particle sets is a path hypothesis. Now an individual map of landmarks can be computed. This key idea is known as the *Rao-Blackwellization* or *fastSLAM*.

Rao-Blackwellization performs a marginalization over the probability distribution in the state space. Instead of using sampling to represent the multivariate probability distribution of the state space, marginalizing out a subset of the state space, is a much more efficient method when using a Gaussian distribution, as seen in figure 9 where the map is marginalized to several smaller maps. This marginalization has become very popular in SLAM problems because jointly sampling over position and map is impractical. When the creators of FastSLAM realized that Rao-Blackwellization could help to marginalize the maps from the joint distribution, the SLAM problem became

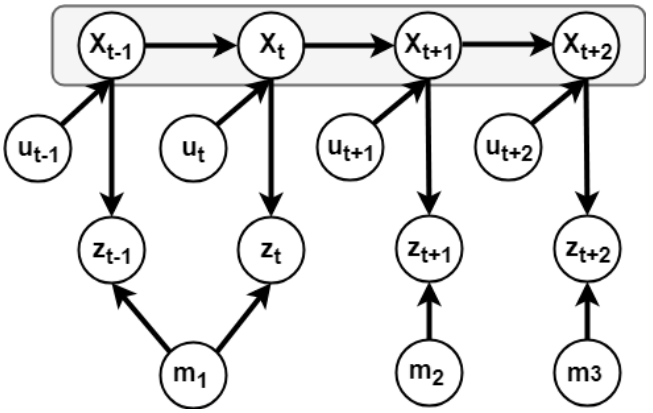


Figure 9: Graphical model of fast SLAM map marginalization

much more tractable.

FastSLAM is an algorithm based on particle filtering that recursively estimates the full posterior distribution over the systems pose and landmark locations, yet scales logarithmically with the number of landmarks in the map. The algorithm has been run successfully on as many as 50000 landmarks [20]. FastSLAM decomposes the SLAM problem into a robot localization problem and a collection of landmark estimation problems that are conditioned on the robots pose estimate. Each particle in FastSLAM processes N Kalman filters that estimate the N landmark locations conditioned on the path estimate. This results in an algorithm requiring $O(MN)$ time, where M is the number of particles. By developing a tree-based data structure, the FastSLAM algorithm obtains $O(MlogN)$.

$$\boldsymbol{x}_t^{[k]} \sim \boldsymbol{p}(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}^{[k]}, \boldsymbol{u}_t) \tag{10}$$

$$\boldsymbol{\omega}^{[k]} = |2\pi \boldsymbol{Q}|^{[k]}\ exp(-\frac{1}{2}(\boldsymbol{z}_t - \hat{\boldsymbol{z}}^{[k]})^T\ \boldsymbol{Q}^{-1}\ (\boldsymbol{z}_t - \hat{\boldsymbol{z}}^{[k]}) \tag{11}$$

The key steps of the fastSLAM algorithm are shown in equation (10) and (11). Equation (10) extends the path posterior by sampling a new pose for each sample. Equation (11) computes the particle weighting parameter $\omega^{[k]}$ that helps the algorithm update the belief of each observed landmarks for each sample, and then resample. k represents the particle number, Q is the measurement covariance matrix, and $\hat{z}^{[k]}$ is the expected observation. See [20] for a complete review.

4.3 Graph-Based SLAM

In Graph-Based SLAM, landmarks and robot locations can be thought of as nodes in a graph. Every consecutive pair of locations x_{t-1}, x_t is tied together by an edge/spatial-constraint that represents the information conveyed by the control reading u_t. Edges also exist between the nodes that correspond to locations u_t and landmarks m_i, assuming a landmark is observed, represented by the orange arcs in figure 10. Edges in this graph are soft constraints. Relaxing these constraints and finding a node configuration that minimizes the error in the edges yields the robot's best estimate for the map and the full path.

The construction of the graph is illustrated in figure 10. Suppose at time t_{t-1}, the robot senses landmark m_1. The edge between these two nodes is added to the yet incomplete graph, as shown by the red line. When catching the edges in a matrix format (which happen to correspond to a quadratic equation defining the resulting constraints), a value is added to the elements between x_{t-1} and m_1. If the system now moves, the control readings will create a new edge between x_{t-1} and x_t. By doing this in consecutive fashion, edges from observations and controls lead to a graph-matrix of increasing size. Nevertheless, this graph is sparse, in that each node is only connected to a small number of other nodes (assuming a sensor with limited sensing range). The number of constraints in the graph is (at worst) linear in the time elapsed and in the number of nodes in the graph.

In figure 11 and figure 12 an implementation of a Graph-based SLAM algorithm is shown. Every node in the graph corresponds to a robot position and a laser mea-

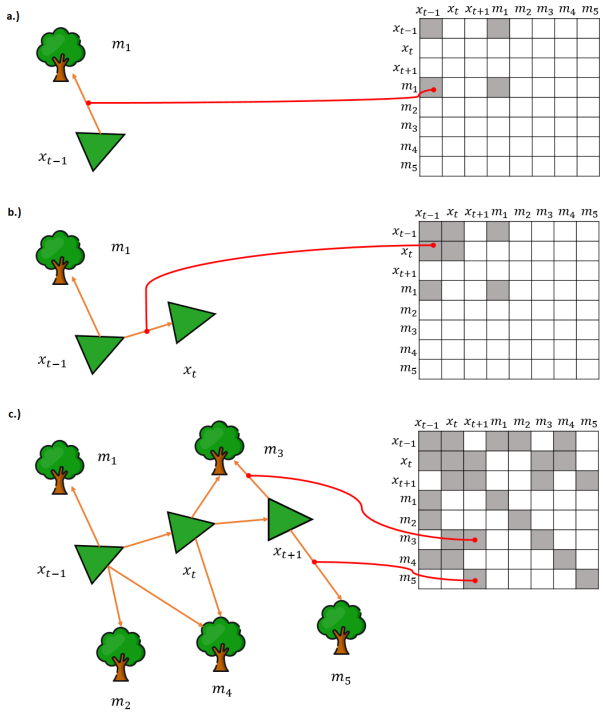


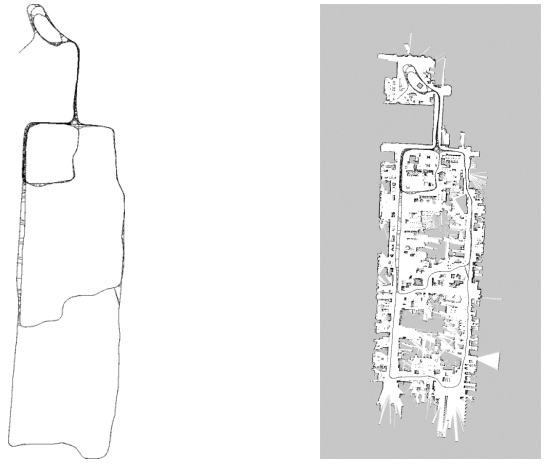
Figure 10: Construction of Graph

surement. We can see in the first figure that the uncertainty from the measurements and controls is large. By letting edges between two nodes be represented by spatial constraints, a graph is made in the last frame of figure 11. Once the graph is obtained the most likely map is obtained by correcting the nodes, see figure 12. Finally, the map can then be rerendered based on the now known poses.

Graph-SLAM methods were originally used to solve the full SLAM problem offline, but more recent techniques handle the online-SLAM problem by incrementing and re-using the previously computed solutions. In comparison to EKF-SLAM, graphical methods scale to higher-dimensional maps, exploiting the sparsity in the graphs.

(a) (b) (c)

Figure 11: Graph-Based SLAM, KUKA Halle 22, courtesy of P. Pfaff & G. Grisetti



(a) (b)

Figure 12: Graph-Based SLAM, KUKA Halle 22, courtesy of P. Pfaff & G. Grisetti

5 Dataset and Sensors

5.1 Ford Campus Data set

Most of the data used in this thesis are presented by the open source *Ford Campus Vision and Lidar Data set* [25], produces by the University of Michigan. The data is collected in downtown Michigan in an urban environment. The perception data used in the simulations comes from a *Velodyne* HDL64E *lidar* spinning at 10Hz providing raw point cloud data with a resolution of 80000-100000 points per scan. The navigation data used in the simulations is retrieved from a *Applanix POS-LV 420 INS*. It's a professional-grade turnkey position and orientation system combining a differential GPS and an IMU. Ran at a sample rate of 100Hz, the sensor provides the data set with the relative position, orientation, velocity, angular rate and acceleration estimates of the vehicle.

5.2 Ouster Lidar

The Ouster-1 LiDAR used in this project is a 16-beam 3D LiDAR with a 16.6 deg vertical field of view. It is able to capture 327 680 points per second. Every point received contains data information including range, intensity, reflectivity, ambient data, angles, and time-stamp. Ouster-1 has a built-in IMU for gyro and accelerometer readings. See appendix A for the Ouster-1 datasheet. The Ouster 1 software can be obtained from their GitHub branch [24] by either cloning it or downloading it and build it as a ROS node using ROS package building commands *cmake* and *make*.

To be able to test the SLAM algorithms, I recorded two different types of data-sets using the Ouster Lidar. One inside capturing two rooms, and one outside on the roof of NTNU EL-bygget. The sets were recorded by holding the LIDAR over my head and walking around trying to capture as much of the surroundings as possible. How this was done is explained better in section 6.1.

6 Implementation

The following chapters describe how the different approaches in this thesis were implemented. The Ouster Lidar has been implemented to run in ROS, and procedures on how the lidar is implemented in two algorithms are explained. The point cloud registration and GPS/IMU/LIDAR simulation has been implemented in Matlab. Most of the testing was done on the Ford Campus data-set, and here in a certain area of the data-set. The set is big and simulation-times suffer. Therefore most of the data is collected 5 minutes into the set, and here running for about 20 seconds. This accounts for approximately 200 LIDAR scans at 10Hz.

6.1 Ouster Lidar

The software for the Ouster 1 LIDAR was implemented in Ubuntu. To communicate with the lidar, a Dnsmasq server[7] was set up to read the LIDARs interface. To obtain this interface, the LIDAR was connected with an ethernet cable, and the interface code was copied over to the Dnsmasq's config file. The network configuration in Ubuntu was assigned a manual IP-address that was configured in the Dnsmasq config so that the Dnsmasq could allocate a range of IP-addresses for the LIDAR to connect to. After the initial set up, a simple terminal command, *sudo systemctl start Dnsmasq.service*, initializes the network infrastructure for the LIDAR to connect. By running, *journalctl -fu Dnsmasq*, I was able to obtain the IP-address and hostname of the Ouster LIDAR.

A test of the sensor could now, after downloading the sensors test software as described in section 5.2, be visualized by writing the following in the Ubuntu terminal:

- *Terminal 1:* source catkin_ws/devel/setup.bash
  roslaunch ouster_ros os1.launch os1_hostname:=<hostname> ...
  os1_udp_dest:=<udp_data_destination_ip-address>

- *Terminal 2:* source catkin_ws/devel/setup.bash
  rviz -d /<ouster_master_destination>/ouster_ros/viz.rviz

where
<hostname> was the IP-address along with the sensor-id obtained by looking at the running Dnsmasq server, and <udp_data_dest> was the manually assigned IP-address for the network. To record a set of point clouds, or a bag, a third terminal window should include:

- *Terminal 3:* source catkin_ws/devel/setup.bash
  rosbag record /os1_node/imu_packets /ose_node/lidar_packets

If the goal was to simply play a pre-recorded point cloud bag, the command for hostname and IP-address in the first terminal window was substituted by *replay:=true* and a fourth terminal window was assigned to play the recorded bag.

- *Terminal 4:* source catkin_ws/devel/setup.bash
  rosbag play –clock <bag_destination>

To end the Dnsmasq server, the command *sudo systemctl stop Dnsmasq.service* can be entered. After this implementation, the sensor should be able to run with other types of software by running the commands in terminal 2, setting it ready as a node in ROS.

6.2 Registration

Registration is an essential part when you want to map an environment for a vehicle or robot to move within. Both speed and accuracy are important factors to evaluate for point cloud registration when SLAM optimally runs in real-time. A simulation on two of the most qualified registration algorithms has been implemented to discover how they compare, and if accuracy and resolution is as important as the speed of the registration.

6.2.1 The ICP registration

The ICP registration algorithm is implemented through Matlab. Two consecutive lidar scans are converted from a set of 3D-points into a Matlab point cloud object. The first scan was appointed as the *Target* and the second as the *Source*. Every scan obtained from the Velodyne HDL64E returns about 80 000 points in the 3D coordinate system. Iterating over a set of that size demands a lot from the algorithm, but in most cases, a resolution of this size is not necessary. The two point clouds were therefor downsampled with a sample rate between 5-10 samples, resulting in two clouds consisting of about 8000 - 16000 points. The downsampled clouds were then matched using a k-d tree, which is a data structure used in computer science for organizing some number of points in a space with k dimensions. The k-d tree is very useful for range and nearest neighbor searches, which it was used for in this case. Because of measurement noise, it can be helpful to remove unwanted outliers. An outlier filter with a predefined weighting parameter d_{max} was applied, where the sample rate of the point cloud was adjusted to obtain better performance.

$$\boldsymbol{M} = \underset{M}{\operatorname{argmin}}\left(\sum_{i} \omega_i((M \cdot b_i - c_i)n_i)^2\right) \tag{12}$$

In equation (12) the sum of the squared distance between each source and target point was minimized returning the 3D rigid body transformation matrix between each pair of neighbour points. Earlier methods of the ICP algorithm uses a point-to-point error metric minimization, as the algorithm in chapter 2.5.2. In equation (12), the parameter $n_i = (n_{ix}, n_{iy}, n_{iz}, 0)^T$ is included as the unit normal vector from the tangent plane of point c_i. This is therefor called a point-to-plane error minimization[19]. The rigid body transformation M composed of a rotation matrix $R(\alpha, \beta, \gamma)$ and a translation matrix $T(t_x, t_y, t_z)$ is usually computed using nonlinear least-square methods. In the point-to-plane method an approximation was done for angles $\theta \approx 0$ on $\sin\theta \approx \theta$ and $\cos\theta \approx 1$. Therefor, when $\alpha, \beta, \gamma \approx 0$:

$$\boldsymbol{R}(\alpha,\beta,\gamma) \approx \begin{pmatrix} 1 & \alpha\beta-\gamma & \alpha\gamma+\beta & 0 \\ \gamma & \alpha\beta\gamma+1 & \beta\gamma-\alpha & 0 \\ -\beta & \alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \approx \begin{pmatrix} 1 & -\gamma & \beta & 0 \\ \gamma & 1 & -\alpha & 0 \\ -\beta & \alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \hat{\boldsymbol{R}}(\alpha,\beta,\gamma) \quad (13)$$

from

$$\boldsymbol{R}(\alpha,\beta,\gamma) = \boldsymbol{R}_x(\gamma)\cdot\boldsymbol{R}_y(\beta)\cdot\boldsymbol{R}_x(\alpha) = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ + & 0 & 0 & 1 \end{pmatrix} \quad (14)$$

where

$r_{11} = \cos\gamma\cos\beta$, $r_{12} = -\sin\gamma\cos\alpha + \cos\gamma\sin\beta\sin\alpha$, $r_{13} = \sin\gamma\sin\alpha + \cos\gamma\sin\beta\cos\alpha$,
$r_{21} = \sin\gamma\cos\beta$, $r_{22} = \cos\gamma\cos\alpha + \sin\gamma\sin\beta\sin\alpha$, $r_{23} = -\cos\gamma\sin\alpha + \sin\gamma\sin\beta\cos\alpha$,
$r_{31} = -\sin\beta$, $r_{32} = \cos\beta\sin\alpha$, $r_{33} = \cos\beta\cos\alpha$

This resulting in an approximation of $\hat{\boldsymbol{M}}$ to equation (15).

$$\hat{M} = T(t_x,t_y,t_z)\cdot\hat{R}(\alpha,\beta,\gamma) = \begin{pmatrix} 1 & -\gamma & \beta & t_x \\ \gamma & 1 & -\alpha & t_y \\ -\beta & \alpha & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (15)$$

Now for each paired point, the $((\hat{M}\cdot b_i - c_i)\cdot n_i)^2$ part of the approximated rigid body transformation can be written as a linear expression $Ax-b$. This cuts down simulation times and is able to perform when the relative orientation between two points is quite large.

When the rotation and translation between points in the two planes were calculated

through the M matrix, a transformation was performed aligning the source point cloud to the target point cloud. Matlab offers a function to merge multiple point clouds, so this was the last step of the ICP-method as new scans were transformed and merged into the reference. Figure 13 illustrates the implementation.

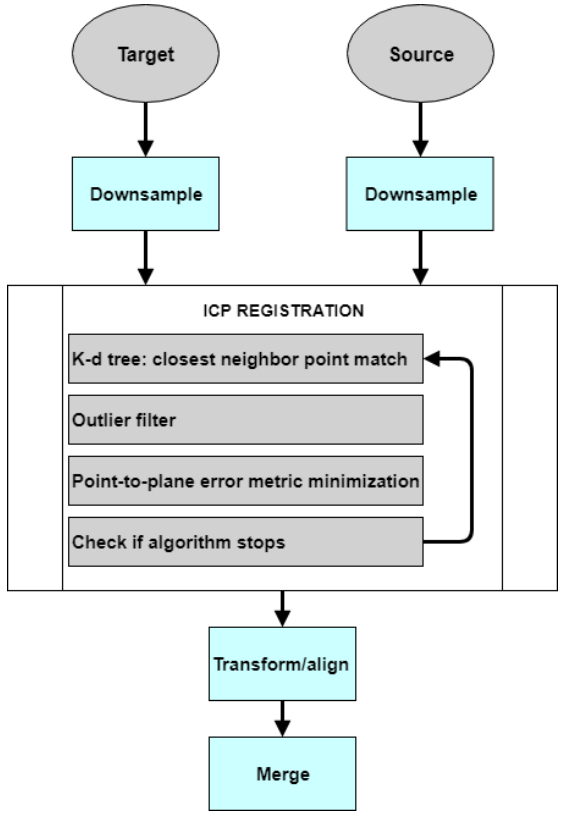


Figure 13: ICP Registration

6.2.2 NDT Registration

As for the ICP-registration, the NDT-registration method require a input of two consecutive scans. The first scan was divided into voxels in the 3D space with a certain grid-size for each side of the voxel [1]. Every point $p_i = (p_{ix}, p_{iy}, p_{iz})^T$ in the voxel was collected, and the mean $\mu = \frac{1}{n}\sum_i p_i$ and the covariance $\Sigma = \frac{1}{n}\sum_i (p_i - \mu)\cdot(p_i - \mu)^T$ was calculated to find the probability of measuring a sample of points within the voxel. This probability was modeled by the normal distribution $N(\mu, \Sigma)$ in equation (16).

$$p(p) \sim \exp(-\frac{(p-\mu)^T \Sigma^{-1}(p-\mu)}{2}) \tag{16}$$

This equation results in a voxel with probability densities for every point. The goal of the approach was to recover estimates for a pose, $\boldsymbol{x}$, consisting of positions x, y and z, as well as roll pitch and yaw angles ψ, ϕ and θ. After building the NDT for the first scan, these parameters were initialized at zero. The second scan was then divided in a similar way and mapped to the first map using these parameters. The corresponding normal distribution was then calculated for the mapped points, and a new cost parameter was calculated in equation (17) to evaluate the accuracy of the new parameters.

$$J(x) = \sum_i \exp(\frac{-(p_i^* - \mu_i)^T \Sigma_i^{-1}(p_i^* - \mu_i)}{2}) \tag{17}$$

$$p_i^* = Rp_i + t \tag{18}$$

where
R and t is the rotation and translation matrices related to the vehicle pose. μ_i and Σ_i is the mean and covariance for points in voxel$_i$, which corresponds to p_i^*. The cost function was then maximized using the Newton algorithm as mentioned in chapter 2.5.3. Figure 14 illustrates the implementation.

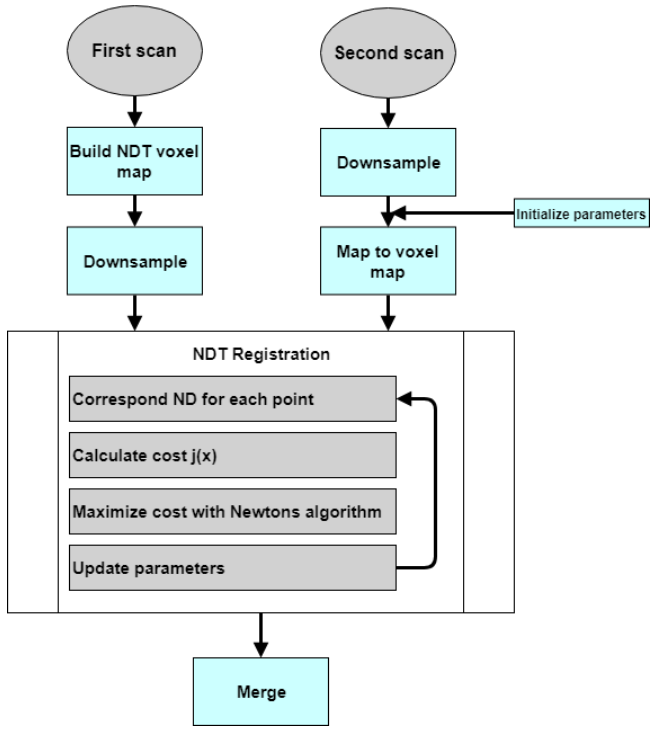


Figure 14: NDT registration

6.3 Localization

For all robotic applications, it is important to know the location and orientation of the robot or vehicle. Especially for SLAM systems that operate in dynamic environments, the certainty in position and pose is extremely important to not damage the system or the people and objects in the environment. An inertial navigation system (INS) with the fusion of an inertial measurement unit (IMU) and a global positioning system (GPS) has been implemented to simulate a pose estimation using an Extended Kalman filter. This was implemented to compare the results to a more common SLAM localization method. The purposed time window of this project didn't allow me to implement a

SLAM localization method, so the INS was compared to a SLAM solution provided by Matlab.

6.3.1 GPS & IMU

By using data from the Ford Campus data-set, a ground truth trajectory was created from the *Applanix Pos LV* professional grade position and orientation system, by extracting all poses of the vehicle in a certain time-space. The Matlab function *wayPointTrajectory* calculates ground truth pose based on a specified sampling rate, way-points, time of arrival and orientation. I wanted to include a GPS/IMU fusion INS to compare the position estimate with the ground truth, seeing how useful these sensors could be for SLAM integration. This whole implementation was done in Matlab for easy comparison of the results with the Ford Campus data-set, which deliver most of their parameters in Matlab *.m-files*.

The sampling rates for the GPS and IMU were sat at 10Hz and 100Hz respectively, the same sampling rate used in the Ford Campus set. The filter to fuse the IMU and GPS measurements was created as an Extended Kalman filter to track the position, orientation, velocity, and the sensor bias offset. The filter updates with a factor of $ImuSamplingRate/2$.

The filter has 16 different states, the same number as described in the Inertial navigation chapter in section 2.4.3. The different states is shown in table(1).

Table 1: INSfilter states

States	**unit**	**Index**
Orientation	(quaternion)	1:4
Gyroscope Bias (XYZ)	rad/s	5:7
Position	m	8:10
Velocity	m/s	11:13
Accelorometer Bias (XYZ)	m/s^2	14:16

Nonlinear differential equations for the system:

$$\begin{aligned} x_t &= f(x_{t-1}, u_t) + w_t \\ z_t &= h(x_t) + v_t \end{aligned} \tag{19}$$

with $w(t) \sim N(0, Q)$ and $v_t \sim N(0, R)$, meaning $w(t)$ and $v(t)$ is Gaussian noise with covariance Q and R.

x_t is defined as in equation (2), but with the velocity states denoted with a capital V_t to distinguish it from the measurement noise v_t in the differential equations.
Here

$$p_t = p_{t-1} + T_s V_{t-1} + \frac{T_s^2}{2}(R_b^n(q_{t-1})s_t - g) \tag{20}$$

$$V_t = V_{t-1} + T_s(R_b^n(q_{t-1})s_k - g) \tag{21}$$

$$q_t = (\cos(0.5 \cdot T_s||\omega_t||I_4 + \frac{1}{||\omega_t||}\sin(0.5 \cdot T_s||w_k||)\Omega_t)q_{t-1} \tag{22}$$

$$\Omega_t = \begin{bmatrix} 0 & \omega_{t,z} & -\omega_{t,y} & \omega_{t,x} \\ \omega_{t,z} & 0 & -\omega_{t,x} & \omega_{t,y} \\ \omega_{t,y} & -\omega_{t,x} & 0 & \omega_{t,z} \\ \omega_{t,x} & -\omega_{t,y} & \omega_{t,z} & 0 \end{bmatrix} \tag{23}$$

is the navigation equations, where T_s is the sampling rate, R_b^n is the rotation matrix rotating from the body frame to the navigation frame. The gravity vector g is assumed constant and subtracted to obtain accelerations in the tangent plane. q_t is the orientation as quaternions, calculated by the quaternion function in Matlab. The measurement noise model is described by

$$\tilde{u}_t = u_t + b(t) + n(t) \tag{24}$$

where u_t is defined as in equation (3). n is additive white noise with variance Q, and b is a slowly varying sensor bias.

The filter states are initialized by the ground truth, resulting in it converging faster to the estimated states. The prediction step of the EKF estimates the filter states based on ω_t from the measurement noise model. The Kalman gain updates the state estimates and the state covariance estimate. The GPS position estimate was updated at a lower sampling rate, contributing to the position estimate of the filter.

6.4 SLAM Algorithm

The choice and implementation of the SLAM algorithm were done on the background of the research done in TTK4551 - Engineering Cybernetics, Specialization Project as mentioned in the preface. The graph SLAM method seems to have become the standard in modern SLAM solutions. This is mainly because the method is able to handle a larger set of observations and a more complex map. Different SLAM solutions were tested to see how easy the Ouster Lidar would be to implement into these algorithms.

The two main solutions that were tested were the LOAM 3D SLAM [35] and the HDL Graph SLAM. In the end, the choice fell on the HDL Graph SLAM algorithm because it was easier to implement and adjust parameters, and it also offered loop closing capabilities. Also, the LOAM method resulted in more drift in the tests that were carried out in real-time.

6.4.1 HDL Graph Slam

High Definition LiDAR (HDL) graph SLAM, is an open source ROS package for real-time 6DOF(degrees of freedom) SLAM using a 3D LIDAR. It is a 3D SLAM method based on Graph SLAM (section 4.3) with the option of NDT, and ICP scan matching. It also supports several, graph constraints including GPS, IMU acceleration and orientation, and floor constraints detected in point clouds. Information and code is obtained from the hdl_graph_slam Github branch [16].

The HDL Graph SLAM code was implemented in the same manner as the Ouster LIDAR software by building a catkin workspace for use in ROS. The HDL-GS consists of the four ROS nodelets: *Prefiltering*, *Scan matching*, *Floor detection* and *Graph_SLAM*. The input cloud from the LIDAR measurements was first downsampled by the prefiltering nodelet. The filtered points from the prefiltering nodelet were then passed through to the scan matching nodelet which estimated the sensor pose by iteratively applying a scan matching between consecutive frames. Consecutively the floor detection nodelet detects floor planes by RANSAC [9]. The estimated odometry and the detected floor planes were sent to the Graph SLAM. To compensate for the accumulated error of scan matching, the algorithm performs loop detection and optimizes a pose graph which takes various constraints into account.

As many other SLAM algorithms, the HDL approach is built from two main c++ libraries, the Point Cloud Library (PCL) and the General Graph Optimization (g2o) libraries. The PCL is a great source for point cloud and 3D processing, including libraries for the most known registration algorithms and several filters applicable for point clouds. The g2o is useful for optimizing graph-based non-linear error functions,

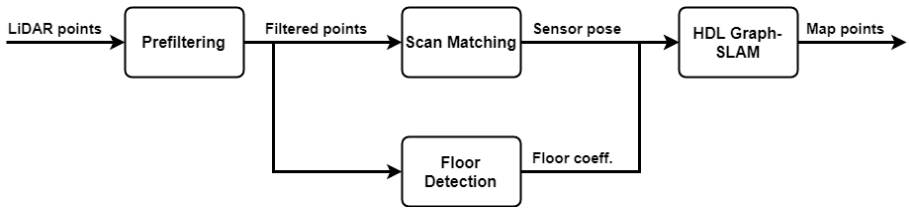


Figure 15: Illustration of HDL Graph SLAM nodelets

making it great for the localization aspect of the SLAM algorithm. When I later mention the inclusion of registration and localization methods for the HDL Graph SLAM algorithm, these are the libraries used to execute these tasks.

To use the Ouster LIDAR in the HDL-GS environment, a launch-file was produced, as explained in section 2.6, to connect the different nodes of the algorithm. The Ouster LIDAR sends out a message topic called *os1_cloud_node/points*, which is the topic of the point cloud. In the launch file section for the prefiltering-nodlet, this topic was mapped so that the algorithms receive point cloud inputs from the Ouster sensor. The launch-file specifies the downsampling resolution of a Voxelgrid applied to the point cloud topic. An outlier removal method is also included in the prefiltering part of the algorithm. The launch-file specifies which type of PCL outlier removal tool that is to be applied to the point cloud.

The scan matching nodlet implements the NDT and ICP registration method mentioned earlier through the PCL. The launch-file was used to decide which method to use, and how the parameters were tuned. The floor matching and Graph SLAM nodlet have not been tweaked with too much by the launch-file regarding this thesis. In the period of this thesis, I had a lot of trouble extracting the correct IMU topic, and use these important parameters in the algorithm. The sensor doesn't include any GPS, which also is an important part of the Graph SLAM nodelet. To watch the complete ROS node tree see appendix B.

To run the HDL GS algorithm after creating a launch-file that utilize your sensor, you have to apply the same procedure as for the Ouster sensor in section 6.1, where a new

terminal window includes:

- *Terminal:* source /catkin_ws/devel/setup.bash
  roslaunch hdl_graph_slam <launch-file>

The algorithm builds a graph based map as explained in section 4.3 where nodes represent the poses of the robot, while the edges represent the connection between these poses and the possibility of loop closures. The map and the robot localization was observed using *rviz*, which is a 3D-visualization tool in ROS. It was launched in a new terminal with the command:

- *Terminal:* source /catkin_ws/devel/setup.bash
  rviz -d /<3D-tool destination>

The terminal launching the algorithm and the launch-file monitors the map optimization and the loop closure detection candidates. The complete launch-file can be seen in appendix B.

7 Results

7.1 Registration

Both the Iterative Closest Point and the Normal Distribution Transform registration methods performed well in Matlab on a Dell OptiPlex 7060, with 32GB RAM and an Intel i7-8700 processor. The presented results use the same point cloud parameters for both methods, and the same method of downsampling was used.

The ICP registration algorithm was tested with the point-to-point and point-to-plane error minimization. The results are represented in figure 16 - 19. As we can see, the methods behave very similarly with quick convergence to within the set tolerance. The biggest difference lies in the number of iterations needed for the algorithms to satisfy the set tolerance. It also seems like the point-to-point method needs a couple of iterations to obtain the correct rigid transform between the point clouds, as seen in figure 16. The same holds for the root mean square error for the two methods, which is the Euclidean distance between the aligned points. The point-to-plane used fewer iterations to converge to the correct estimate. A test was carried out to test how this would affect the performance. Table 2 shows the running time of the two methods by registering 50 scans from the Ford Campus data set at different particle densities. This shows that the difference between the methods isn't substantial, but the point-to-plane is slightly faster and should, therefore, work better if there is need of a higher resolution point cloud to extract key features for use in e.g SLAM.

Table 2: Performance test ICP

Downsampled	**Point-to-Point [t]**	**Point-to-Plane [t]**
5 %	10.40	10.44
10%	12.45	11.90
15%	14.93	13.93
25%	20.36	17,35
50%	34,11	27,17

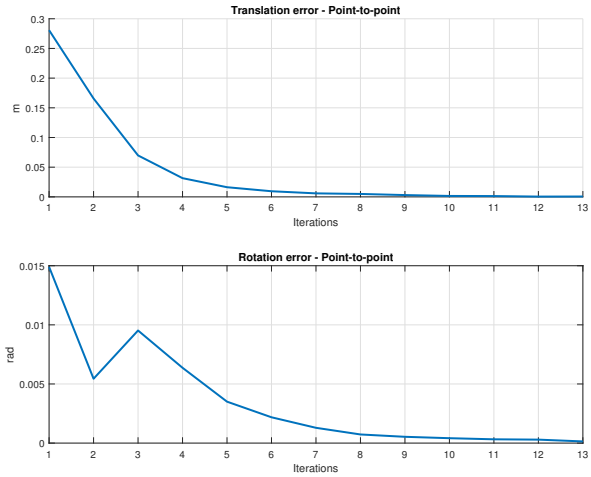


Figure 16: Point-to-point transformation

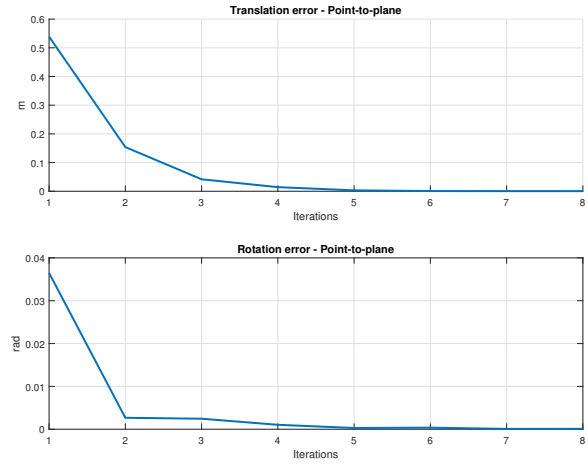


Figure 17: Point-to-plane transformation

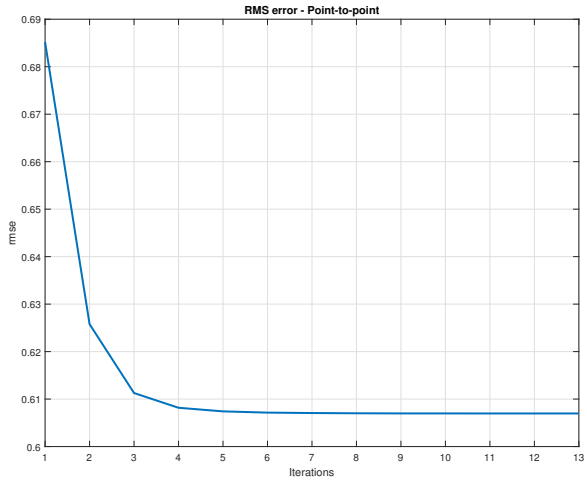


Figure 18: Point-to-point RMSE

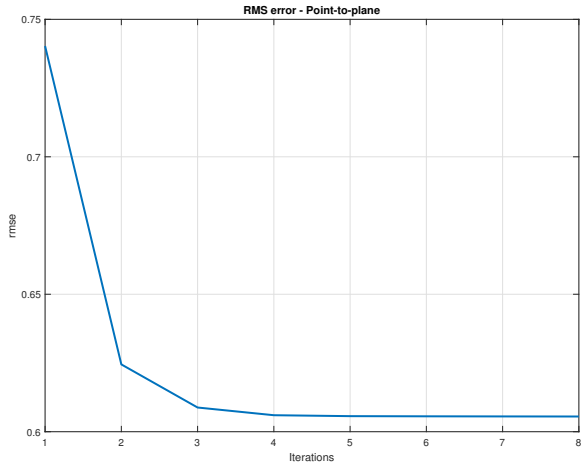


Figure 19: Point-to-plane RMSE

The main adjustable parameter for the NDT is the size of the voxels in the Voxelgrid. The root mean square error was monitored and plotted after each iteration of the algorithm. The result is presented in Figure 20, were the fastest conversion to the correct Euclidean distance between the aligned points was for a VoxelSize of approx 6 [cm]. Grid sizes below or above the ones shown in the figure gave unstable results, and did not seem to find the same estimated *rms-error* as the ones presented., although some parameters gave good results with faster simulation times. The number of iterations and the similar values in different voxel sizes tells me that the tuning of the NDT parameters is very dependent on the data it analyses. Some voxels may contain a lot of points in some areas, and not so much in others, making it all that more important to test the registration algorithm with different parameters when used in SLAM applications.

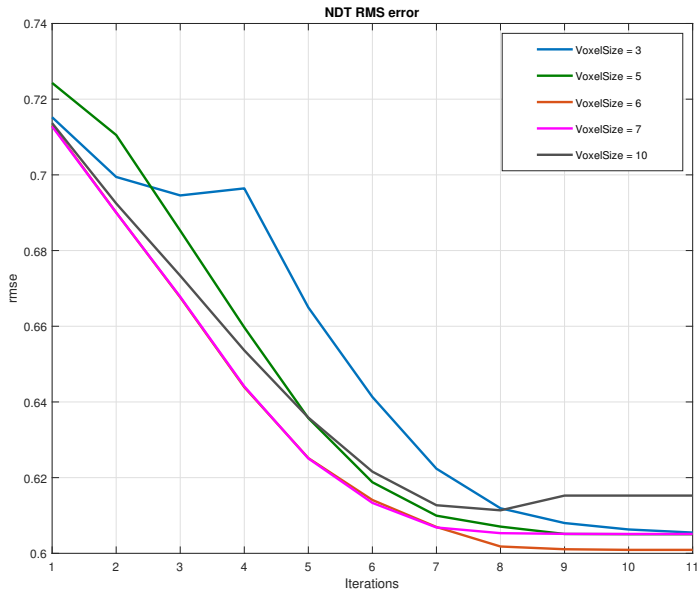


Figure 20: NDT RMSE with different VoxelGrid sizes

As one can see from table 3, running 200 LIDAR scans with the same parameter settings for the two methods, yields a much faster result for the Iterative Closest Point algorithm. However, as figure 21 and figure 22 try to illustrate, the Normal Distribution Transform algorithms are more precise in merging larger sets of scans. The ICP solution isn't as accurate as the NDT. The colored fields of figure 21 implies that the transformed points are somewhat skewed in comparison to the NDT map. This would be better illustrated with a 3D tool.

Table 3: Merging 200 scans for ICP and NDT

Method	**time [s]**
Iterative Closest Point	47.21
Normal Distribution Transform	880.48

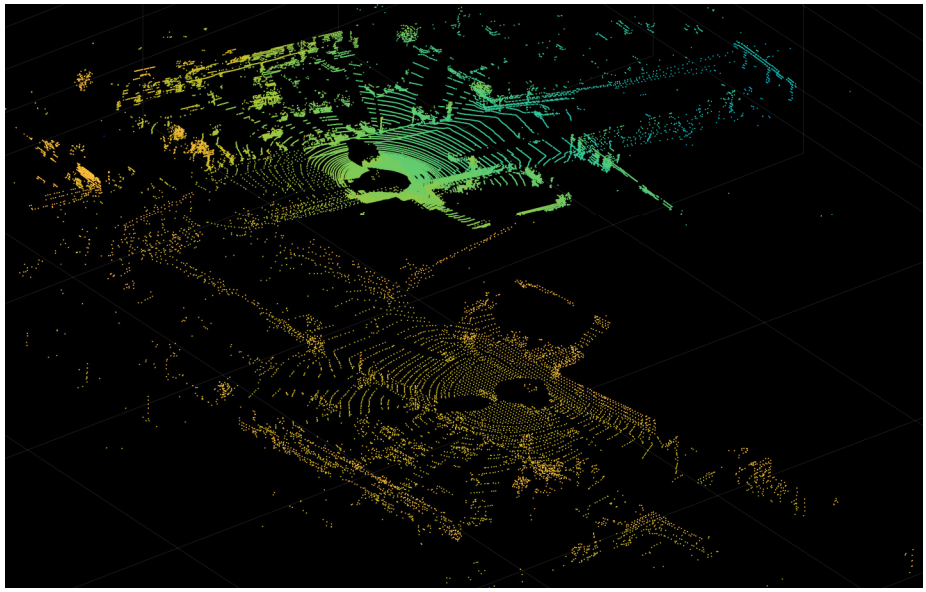


Figure 21: ICP merged

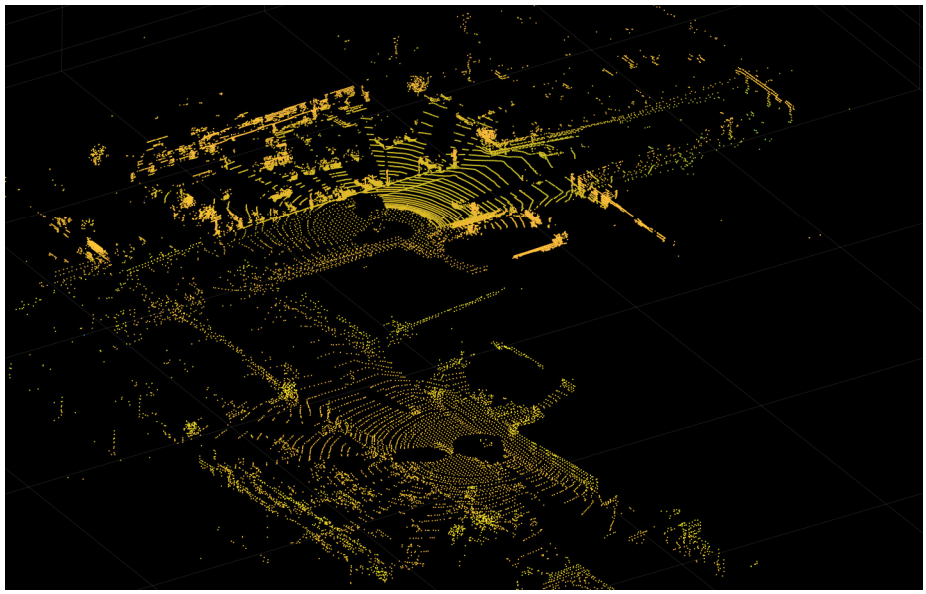


Figure 22: NDT Merged

7.2 Localization

The inertial navigation position and orientation estimates were compared to the ground truth trajectory by computing an end-to-end root mean square error, which is the average *rms-error* of all pose estimates compared with the pose of the ground truth. The simulation was done in the same time window as the 200 Velodyne scans used in the registration simulation (7.1). The ground truth trajectory includes a straight line and a curvature. The results of the INS sensor fusion is presented in figure 23 and 24. Here the *rms-error* for the position estimates is presented for axes x, y and z. The INS filter manages to estimate the position of the vehicle for about 14 seconds before the x and y estimates drift. The orientation is calculated as quaternions, and the error between the ground truth orientation and the INS estimated orientation is therefore presented as a quaternion distance, which is the same as angular distance. This result is presented in figure 24. The filter estimates the orientation with small errors, but there is also a tendency of drift in the orientation estimate.

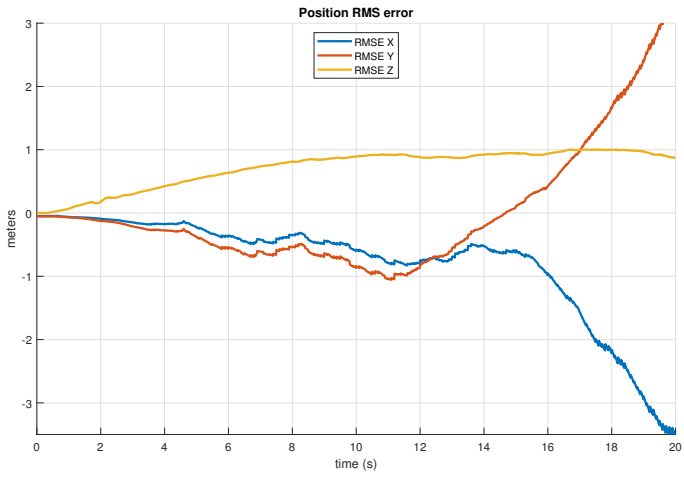


Figure 23: Root mean square error INS

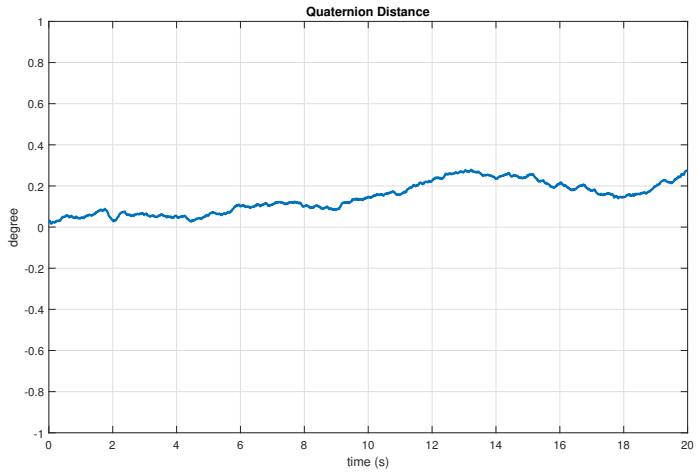


Figure 24: Quaternion distance error INS

The results of the INS sensor fusion was compared with a localization solution in Matlab, by applying the functions *slamObject* and *addScan* that returned a pose estimate based on the LIDAR data from the same trajectory. The results are presented in figure 25 and 26, and table 4. The simulation time presented in the table only accounts for the SLAM solution, as the INS estimations were almost instant. The SLAM function uses an Occupancy grid to map the environment as an evenly spaced field with binary random variables. By adjusting the resolution of this map, the localization and simulation time is greatly affected. The result is a pretty good position estimate, comparable with the INS fusion. With a map-resolution of about 3 grid cells per meter, the simulation time is quite high, but the performance does not really improve with more grid cells per meter. This is illustrated by figure 25 and 26 where the position estimate tracks the ground truth well for a map-resolution of 3 $cells/m$, and similarly with an Occupancy map with a grid size of 8 $cells/m$. With very low map resolutions, the SLAM algorithm has big problems with the calculation of the estimated trajectory.

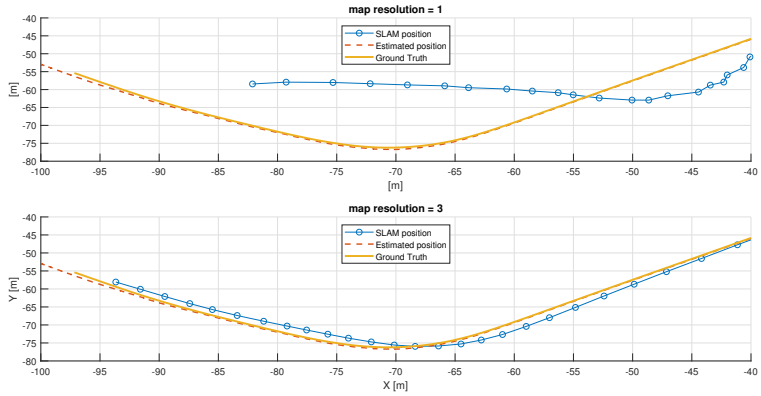


Figure 25: Curved waypoint trajectory, lower map resolution

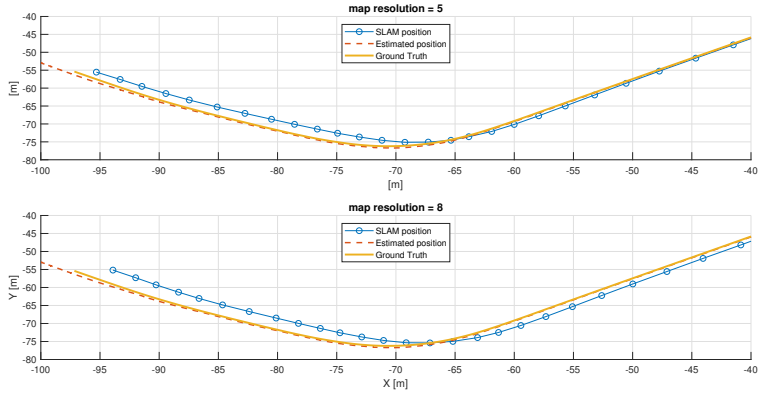


Figure 26: Curved waypoint trajectory, higher map resolution

Table 4: Curve End-to-End position RMS error with different map resolutions in SLAM algorithm

Fusion RMS error [m]	SLAM RMS error [m]	[cell/m]	time (s)
x: 1.1502 y: 0.9838 z: 0.7767	x: 14.628 y: 10.186 z: 1.8689	1.0	9.11
x: 1.1502 y: 0.9838 z: 0.7767	x: 1.2409 y: 0.5354 z: 1.3298	3.0	48.56
x: 1.1502 y: 0.9838 z: 0.7767	x: 0.3257 y: 1.2529 z: 1.3059	5.0	130.01
x: 1.1502 y: 0.9838 z: 0.7767	x: 0.8357 y: 1.3724 z: 1.2914	8.0	407.79

7.3 SLAM

The result of the simultaneous localization and mapping through the HDL Graph SLAM method is shown in figure 27 - 30. The algorithm was tested with the Ouster 1 LIDAR in an indoor and outdoor environment. The resulting figures present the mapped area with the estimated pose of the sensor represented as nodes with edges between them. The launch file for the ROS nodes is included in the appendix[B] along with the ROS node tree for the system.

The point cloud registration in the algorithm was tested with the registration methods proposed in the registration section of this report. Both NDT and ICP was tested with different parameter tuning. All the presented results in the HDL Graph SLAM maps is with the NDT solution. The ICP algorithm was not sufficient and had problems initializing the registration. This resulted in completely wrong pose estimates, without the possibility of graph optimization. The presented methods are therefore registered with the NDT registration method, but with different search methods to correspond between each normally distributed point in the voxel map.

Figure 27 shows a bird's eye view of room GG48 at EL-bygget, NTNU Gløshaugen, mapped with the Ouster 1 LIDAR. The balls in the figure represent the graph nodes, and the coloring represents the uncertainty in the pose estimates. The lines between the nodes is a representation of the edges. The *g2o* library for graph optimization demands a certain number of edges to initialize an optimization sequence, as explained in section 4.3. In the HDL Graph SLAM algorithm, this number was set to 10 edges.

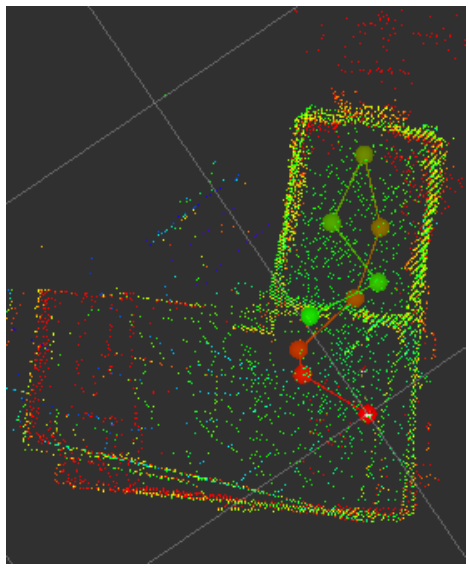


Figure 27: SLAM Ouster Indoors

The algorithm struggled in the indoor environment, and the NDT parameters had to be fine-tuned to even get an update of the map. This results in a skewed map.

Figure 29 and 28 shows a map of the roof of EL-bygget. Fewer disturbances in the environment and more distinct features results in a map that represents the real world much better than the indoors sample. Because of this, the trajectory of nodes tracks the sensor position with much higher accuracy than for the indoors. In the outdoor sample, the algorithm detects 10 edges after just a couple of seconds and is able to initialize the optimization of the graph, compute initial guesses for the pose and compare the error in the pose with the previous step. The blue lines indicate an update of the pose.

Figure 30 shows the map and trajectory of a sample from the higher resolution Ouster LIDAR required from Ouster [23]. The algorithm performed very well on this sample and was quick to initialize and update the graph. The limitation to working with a

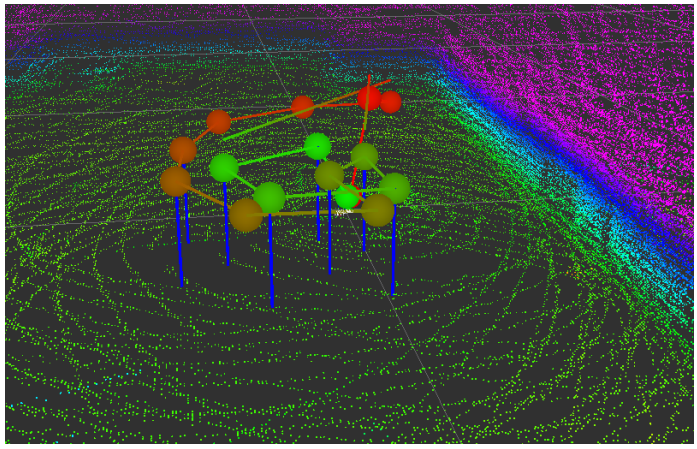


Figure 28: SLAM Ouster outdoors

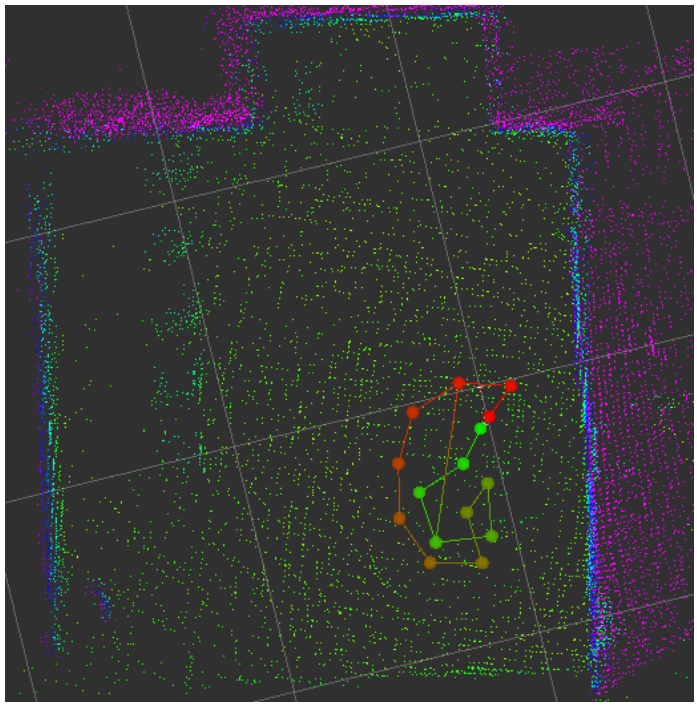


Figure 29: SLAM Ouster outdoors from above

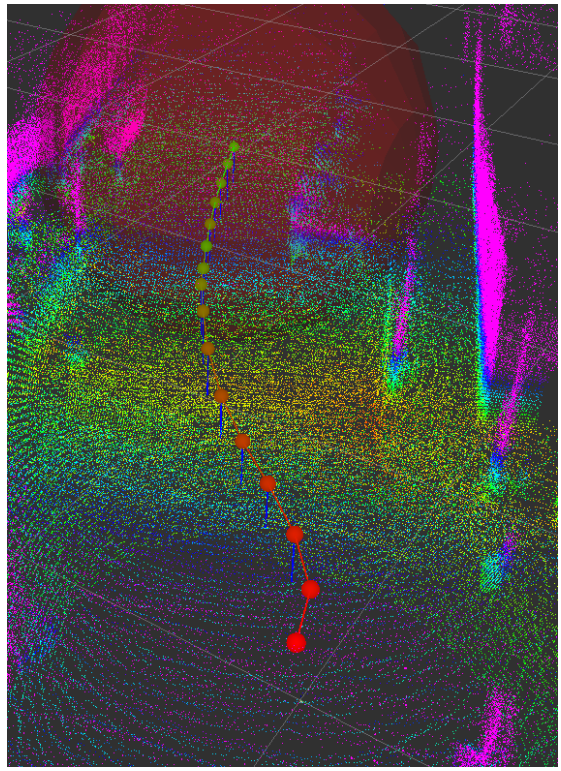


Figure 30: SLAM Ouster64 outdoors high resolution

LIDAR with four times the number of vertical beams is that the number of points makes the computation more demanding. You need a state of the art graphics card to visualize a longer trajectory with point cloud samples in this resolution.

8 Evaluation

8.1 Registration

The report shows a comparison between two different methods for point cloud registration. In the ICP algorithm, the point-to-point and point-to-plane solutions performed somewhat similarly, but the point-to-plane method seemed to consistently use fewer iterations. This means that scan matching with the point-to-point method will gradually take longer to converge than the point-to-plane.

When comparing the results for the NDT and ICP registration solutions, it seems that the ICP was much faster, but the NDT more accurate. The simulations were run in Matlab and consequently simulation times is much slower than the c++ solutions given by *PCL*. Apart from this, the simulations on LIDAR scans proved to be very dependent on point cloud resolution and observation of distinct features. Testing them in the SLAM environment yielded that the NDT algorithm outperformed the ICP significantly. I learned from the registration results that tuning the NDT algorithm was very dependent on the environment. The distinct features of the outdoors data set made it possible for NDT to match the scans with high enough accuracy for the algorithm to calculate good position estimates.

The Coherent Point Drift registration algorithm was very slow and inefficient to work with. Because of this, no simulations were tested with this solution other than some quick testing of performance, and no results are therefore presented on this topic.

The subject of registration was interesting, and there is a lot of methods that I didn't have the time to explore. Too big ambitions for the thesis consequently resulted in me not having the time or knowledge to develop and implement a registration solution of my own.

8.2 Localization

There is presented a localization solution using an INS filter that fuses an IMU and a GPS sensor to obtain estimated poses along a ground truth trajectory. The implemented solution works for a bit but drifts after about 15 seconds into the simulation, probably because of the sensor biases. The error growth rate may be reduced if a model for the vehicle dynamics was included in the information process, then being able to set certain constraints on the system.

The Matlab SLAM solution gave accurate pose estimates but was very computationally demanding on the computer. There is probably a reason that most SLAM and localization problems use c++ solutions to faster compute the necessary pose and map updates. A Markov Localization method was supposed to be tested as a part of this thesis, but unfortunately, I did not manage to implement the solution. A lot of time was was put into reading *.log* and *.xml* files from the Ford Campus data-set. The time that preferably should have been used on implementing a localization solution.

The team behind the HDL SLAM algorithm also offers a localization solution called HDL Localization. It is well tested for Velodyne LIDARs, and implementation of this solution on the Ouster would possibly give more precise position estimates, and combined with the SLAM algorithm a more complete system than I was able to test in this thesis.

8.3 SLAM

The SLAM method HDL Graph SLAM was primarily used as a test environment for the sensor functionality and filter implementation tests. Indoor testing was more complicated because the LIDAR didn't register points inside a circle with a diameter of approx one meter. The mapping was therefore not sufficient enough for the algorithm to estimate the pose and update the graph. When mapping outdoors with a more open environment, the testing was more successful and it was possible to observe how different registration solutions affected the mapping and localization.

Ouster INC is a fairly new supplier of LIDAR systems, and because of this, there was not much documentation of its implementations on state of the art SLAM solutions. The Velodyne is the standard in this field, so most solutions are made with this sensor in mind. Because HDL Graph SLAM was the easiest to implement with the Ouster LIDAR, this was the algorithm that was used to run SLAM.

The goal of this thesis was to design a SLAM system that easily could implement the Ouster LIDAR, and then test this system on a vehicle running real-time simultaneously localization and mapping. I realized far too late that this task was beyond my current knowledge and regrettably had to focus my attention on testing registration and localization solutions for implementing the Ouster LIDAR on current SLAM algorithms. Looking back, a focus on registration solutions with the Ouster LIDAR should have been my focus, not the full SLAM problem.

9 Conclusion

From my experience, it seems like a solution to the SLAM problem requires top quality measurements. The main advantage of LIDARs is that the measurements are highly accurate and that they work in environments were other sensors suffers, e.g. in darkness or undistinct environments. The quality of the LIDARs is also their drawback. State of the art LIDARs are very pricey compared to cameras and limits the research and sales potential of LIDAR SLAM solutions

To determine whether speed or accuracy is more important when it comes to mapping is a hard question to answer since both usually are essential. It comes down to in what system the map is to be used. Speed may be more important in industrial automation, and maybe not. The one thing that is certain is that in the revolution of self-driving systems, the need for both speed and accuracy increases.

A suggestion to improve the estimate of the vehicles pose is to research methods of combining INS with SLAM. The use of GPS and IMU is a well-researched field, so combining the knowledge from this with SLAM methods may result in pose estimations with higher estimation certainty than for today's standard methods.

10 Future work

Since the main goal of the thesis originally was to design my own SLAM algorithm, this is a logical step for future work in this project. The PCL and g2o libraries offer a lot of solutions for registration and localization in c++. A SLAM solution should be implemented in c++ because this is a language that runs very fast and managed to run NDT fast enough to register an accurate map.

I was able to test some registration solutions, but time prevented extensive research of localization possibilities. Some research regarding an INS fusion system was implemented, so further development of this area in SLAM would be an interesting prospect for future work. Not much research is done, but a data fusion algorithm using an Extended Kalman filter for estimation of velocity and position of a UAV was proposed by a team at the University of California[13]. Apart from this, there is not much work with this integrated solution. Using a sensor fusion with IMU and GPS as an INS, alongside SLAM to estimate a position update in an adaptive Kalman filter algorithm on a ground vehicle is a prospect for future work of this master thesis. LIDARs and GPS usually run on similar sampling times, so a fusion of these sensors should be possible for a well defined Kalman filter to handle.

References

[1] Akai, N., Morales, L. Y., Takeuchi, E., Yoshihara, Y. and Ninomiya, Y. [2017-06]. Robust localization using 3d NDT scan matching with experimentally determined uncertainty and road marker matching, *2017 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, pp. 1356–1363.
URL: *http://ieeexplore.ieee.org/document/7995900/*

[2] Bailey, T. and Durrant-Whyte, H. [2006]. Simultaneous localisation and mapping (SLAM): Part II state of the art, p. 10.

[3] Bellekens, B., Spruyt, V., Berkvens, R. and Weyn, M. [2014]. A survey of rigid 3d pointcloud registration algorithms, p. 6.

[4] Besl, P. and McKay, N. D. [1992-02]. A method for registration of 3-d shapes, **14**(2): 239–256.
URL: *http://ieeexplore.ieee.org/document/121791/*

[5] Biber, P. and Strasser, W. [2003]. The normal distributions transform: a new approach to laser scan matching, *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, Vol. 3, IEEE, pp. 2743–2748.
URL: *http://ieeexplore.ieee.org/document/1249285/*

[6] Cyril, S. [n.d.]. Robot motion and sensor models. Slide 48-60.
URL: *http://robots.engin.umich.edu/mobilerobotics/W18/media/Prof_Eustice_slides/L05.MotionSensor*

[7] *Dnsmasq - network services for small networks.* [n.d.].
URL: *http://www.thekelleys.org.uk/dnsmasq/doc.html*

[8] Dubé, R., Dugas, D., Stumm, E., Nieto, J., Siegwart, R. and Cadena, C. [2016-09]. SegMatch: Segment based loop-closure for 3d point clouds.
URL: *http://arxiv.org/abs/1609.07720*

[9] Fontanelli, D., Ricciato, L. and Soatto, S. [2007-09]. A fast RANSAC-based registration algorithm for accurate localization in unknown environments using

LIDAR measurements, *2007 IEEE International Conference on Automation Science and Engineering*, IEEE, pp. 597–602.
URL: *http://ieeexplore.ieee.org/document/4341827/*

[10] Fox, D., Burgard, W. and Thrun, S. [1998-11]. Active markov localization for mobile robots, **25**(3): 195–207.
URL: *https://linkinghub.elsevier.com/retrieve/pii/S0921889098000499*

[11] *Gaussian Mixture Model | Brilliant Math & Science Wiki* [n.d.].
URL: *https://brilliant.org/wiki/gaussian-mixture-model/*

[12] Grewal, M. S., Weill, L. R. and Andrews, A. P. [2001]. *Global positioning systems, inertial navigation, and integration*, John Wiley.

[13] Hening, S., Ippolito, C. A., Krishnakumar, K. S., Stepanyan, V. and Teodorescu, M. [2017-01-09]. 3d LiDAR SLAM integration with GPS/INS for UAVs in urban GPS-degraded environments, *AIAA Information Systems-AIAA Infotech @ Aerospace*, American Institute of Aeronautics and Astronautics.
URL: *http://arc.aiaa.org/doi/10.2514/6.2017-0448*

[14] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C. and Burgard, W. [2013-04]. OctoMap: an efficient probabilistic 3d mapping framework based on octrees, **34**(3): 189–206.
URL: *http://link.springer.com/10.1007/s10514-012-9321-0*

[15] Kohlbrecher, S., von Stryk, O., Meyer, J. and Klingauf, U. [2011-11]. A flexible and scalable SLAM system with full 3d motion estimation, *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, IEEE, pp. 155–160.
URL: *http://ieeexplore.ieee.org/document/6106777/*

[16] koide3 [2018-12]. *3D LIDAR-based Graph SLAM. Contribute to koide3/hdl_graph_slam development by creating an account on GitHub.*
URL: *https://github.com/koide3/hdl_graph_slam*

[17] Labbé, M. and Michaud, F. [2018-10]. RTAB-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-

term online operation.
URL: *http://doi.wiley.com/10.1002/rob.21831*

[18] Leonard, J. J. and Durrant-Whyte, H. F. [1991-06]. Mobile robot localization by tracking geometric beacons, **7**(3): 376–382.

[19] Low, K.-L. [2004]. Linear least-squares optimization for point-to-plane ICP surface registration, p. 3.

[20] Montemerlo, M., Thrun, S., Koller, D. and Wegbreit, B. [2002]. FastSLAM: A factored solution to the simultaneous localization and mapping problem, p. 6.

[21] Neira, J. and Tardos, J. [2001-12]. Data association in stochastic mapping using the joint compatibility test, **17**(6): 890–897.
URL: *http://ieeexplore.ieee.org/document/976019/*

[22] Olson, E., Leonard, J. and Teller, S. [2006]. Fast iterative alignment of pose graphs with poor initial estimates, *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, IEEE, pp. 2262–2269.
URL: *http://ieeexplore.ieee.org/document/1642040/*

[23] Ouster, I. [n.d.]. Index of /sample-data-1.12.
URL: *http://data.ouster.io/sample-data-1.12/index.html*

[24] *Ouster sample code. Contribute to ouster-lidar/ouster_example development by creating an account on GitHub* [2019-05-13]. original-date: 2018-05-08T03:42:01Z.
URL: *https://github.com/ouster-lidar/ouster_example*

[25] Pandey, G., McBride, J. R. and Eustice, R. M. [2011-11]. Ford campus vision and lidar data set, **30**(13): 1543–1552.
URL: *http://journals.sagepub.com/doi/10.1177/0278364911400640*

[26] Rushe, D. [2019-03-10]. 'i'm so done with driving': is the robot car revolution finally near?
URL: *https://www.theguardian.com/cities/2019/mar/09/im-so-done-with-driving-is-the-robot-car-revolution-finally-near-waymo*

[27] Sanchez, J., Denis, F., Checchin, P., Dupont, F. and Trassoudaine, L. [2017-09-30]. Global registration of 3d LiDAR point clouds based on scene features: Application to structured environments, **9**(10): 1014.
URL: *http://www.mdpi.com/2072-4292/9/10/1014*

[28] Smith, R. C. and Cheeseman, P. [1986-12]. On the representation and estimation of spatial uncertainty, **5**(4): 56–68.
URL: *http://journals.sagepub.com/doi/10.1177/027836498600500404*

[29] Smith, R., Self, M. and Cheeseman, P. [1987]. Estimating uncertain spatial relationships in robotics, p. 26.

[30] Stachniss, C., Leonard, J. J. and Thrun, S. [2016]. Simultaneous localization and mapping, *in* B. Siciliano and O. Khatib (eds), *Springer Handbook of Robotics*, Springer International Publishing, pp. 1153–1176.
URL: *https://doi.org/10.1007/978-3-319-32552-1_46*

[31] Thrun, S., Burgard, W. and Fox, D. [2002]. Probabilistic robotics, **45**(3).
URL: *http://portal.acm.org/citation.cfm?doid=504729.504754*

[32] Thrun, S., Fox, D., Burgard, W. and Dellaert, F. [2001-05]. Robust monte carlo localization for mobile robots, **128**(1): 99–141.
URL: *http://linkinghub.elsevier.com/retrieve/pii/S0004370201000698*

[33] Tipaldi, G. D., Spinello, L. and Burgard, W. [2013-05]. Geometrical FLIRT phrases for large scale place recognition in 2d range data, *2013 IEEE International Conference on Robotics and Automation*, IEEE, pp. 2693–2698.
URL: *http://ieeexplore.ieee.org/document/6630947/*

[34] Vebjørn, E. [2018-12-21]. Ouste-1 3d LiDAR SLAM.

[35] Zhang, J. and Singh, S. [2014-07-12]. LOAM: Lidar odometry and mapping in real-time, *Robotics: Science and Systems X*, Robotics: Science and Systems Foundation.
URL: *http://www.roboticsproceedings.org/rss10/p07.pdf*

A Appendix

1: **Algorithm EKF_SLAM**$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, N_{t-1})$**:**

2: $N_t = N_{t-1}$

3: $F_x = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 \end{pmatrix}$

4: $\bar{\mu}_t = \mu_{t-1} + F_x^T \begin{pmatrix} -\frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \omega_t \Delta t \end{pmatrix}$

5: $G_t = I + F_x^T \begin{pmatrix} 0 & 0 & \frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & \frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & 0 \end{pmatrix} F_x$

6: $\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + F_x^T \, R_t \, F_x$

7: $Q_t = \begin{pmatrix} \sigma_r & 0 & 0 \\ 0 & \sigma_\phi & 0 \\ 0 & 0 & \sigma_s \end{pmatrix}$

8: **for all observed features** $z_t^i = (r_t^i \;\; \phi_t^i \;\; s_t^i)^T$ **do**

9: $\begin{pmatrix} \bar{\mu}_{N_t+1,x} \\ \bar{\mu}_{N_t+1,y} \\ \bar{\mu}_{N_t+1,s} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \\ s_t^i \end{pmatrix} + r_t^i \begin{pmatrix} \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \\ 0 \end{pmatrix}$

10: **for** $k = 1$ **to** $N_t + 1$ **do**

11: $\delta_k = \begin{pmatrix} \delta_{k,x} \\ \delta_{k,y} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{k,x} - \bar{\mu}_{t,x} \\ \bar{\mu}_{k,y} - \bar{\mu}_{t,y} \end{pmatrix}$

12: $q_k = \delta_k^T \delta_k$

13: $\hat{z}_t^k = \begin{pmatrix} \sqrt{q_k} \\ \text{atan2}(\delta_{k,y}, \delta_{k,x}) - \bar{\mu}_{t,\theta} \\ \bar{\mu}_{k,s} \end{pmatrix}$

14: $F_{x,k} = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 & 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 & 0 & 1 & 0 \cdots 0 \end{pmatrix}$

15: $H_t^k = \frac{1}{q_k} \begin{pmatrix} \sqrt{q}_k \delta_{k,x} & -\sqrt{q}_k \delta_{k,y} & 0 & -\sqrt{q}_k \delta_{k,x} & \sqrt{q}_k \delta_{k,y} & 0 \\ \delta_{k,y} & \delta_{k,x} & -1 & -\delta_{k,y} & -\delta_{k,x} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} F_{x,k}$

16: $\Psi_k = H_t^k \, \bar{\Sigma}_t \, [H_t^k]^T + Q_t$

17: $\pi_k = (z_t^i - \hat{z}_t^k)^T \, \Psi_k^{-1} \, (z_t^i - \hat{z}_t^k)$

18: **endfor**

19: $\pi_{N_t+1} = \alpha$

20: $j(i) = \underset{k}{\text{argmin}} \; \pi_k$

21: $N_t = \max\{N_t, j(i)\}$

22: $K_t^i = \bar{\Sigma}_t \, [H_t^{j(i)}]^T \Psi_{j(i)}^{-1}$

23: **endfor**

24: $\mu_t = \bar{\mu}_t + \sum_i K_t^i \, (z_t^i - \hat{z}_t^{j(i)})$

25: $\Sigma_t = (I - \sum_i K_t^i \, H_t^{j(i)}) \, \bar{\Sigma}_t$

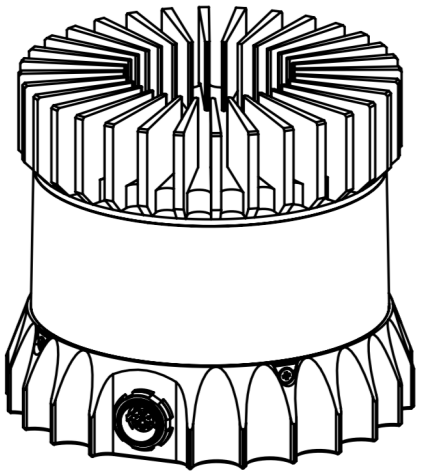
26: **return** μ_t, Σ_t

OUSTER

Ouster, Inc.
350 Treat Ave
San Francisco, CA 94110
LIDAR@ouster.io

OS-1

High Resolution Imaging LIDAR



SUMMARY

The OS-1 offers a market leading combination of price, performance, reliability and SWAP. It is designed for indoor/outdoor all-weather environments and long lifetime. As the smallest high performance LIDAR on the market, the OS-1 can be directly integrated into vehicle facias, windshield, side mirrors, and headlight clusters.

HIGHLIGHTS

- Fixed resolution per frame operating mode
- Camera-grade ambient and intensity data
- Multi-sensor crosstalk immunity
- Industry leading intrinsic calibration
- Open source drivers

OPTICAL PERFORMANCE

Range	0.5 m - 120 m @ 80% reflective lambertian target, 225 w/m2 sunlight, SNR of 12 0.5 m - 40 m @ 10% reflective lambertian target, 225 w/m2 sunlight, SNR of 12 * range of 0.0-80m (min range of 0.0m) in enhanced low range mode
Range Accuracy	Zero bias for lambertian targets, slight bias for retroreflectors
Range Resolution	1.2 cm
Range Repeatability (1 sigma / standard deviation)	SNR >250: ± 1.5cm SNR 100: ± 3 cm SNR 12: ± 10 cm
Vertical Resolution	64 or 16 beams
Horizontal Resolution	2048, 1024, or 512 (configurable)
Field of View	Vertical: +16.6° to -16.6 ° (33.2°) - uniform spacing / Horizontal: 360°
Angular Sampling Accuracy	Vertical: ±0.01° / Horizontal: ±0.01°
Rotation Rate	10 to 20 hz (configurable)
# of Returns	1 (strongest)

LASER

Laser Product Class	Class 1 eye-safe per **[IEC 60825-1:2007 & 2014]**
Laser Wavelength	850 nm
Beam Diameter Exiting Sensor	10 mm
Beam Divergence	0.13° (FWHM)

LIDAR OUTPUT

Connection	UDP over gigabit ethernet
Point Per Second	1,310,720 (64-channel) 327,680 (16-channel)
Data Per Point	Range, intensity, reflectivity, ambient, angle, time stamp
Time Stamp Resolution	10 ns
Data Latency	< 10 ms

IMU OUTPUT

Connection	UDP over gigabit ethernet
Samples Per Second	1,000
Data Per Sample	3 axis gyro, 3 axis accelerometer
Time Stamp Resolution	10 ns

Data Latency	< 10 ms

CONTROL INTERFACE

Connection	TCP over gigabit ethernet
Time Synchronization	Input sources: • IEEE1588 precision time protocol • External PPS • Internal 10 ppm drift clock Output sources: • Configurable 1-60Hz output pulse
LIDAR Operating Modes	Hardware triggered angle firing (guaranteed fixed resolution per rotation): • 64 x 2048 @ 10hz • 64 x 1024 @ 10hz or 20hz • 64 x 512 @ 10hz or 20hz Fixed timing firing: • Configurable measurement period between 50 µsec and 1 second
Additional Programmability	Multi-sensor rotation phase tuning Queryable intrinsic calibration information: • Beam angles • IMU pose correction matrix

MECHANICAL/ELECTRICAL

Power Consumption	14-16 W typical, 18 W peak
Operating Voltage	22-26 V, 24 V nominal
Connector	Proprietary pluggable connector (Power + data + DIO)
Dimensions	Diameter: 85 mm (3.34 in) Height: 73 mm (2.87 in)
Weight	380 g (13.4oz)
Mounting	4 M3 screws / 2 locating 3mm pins

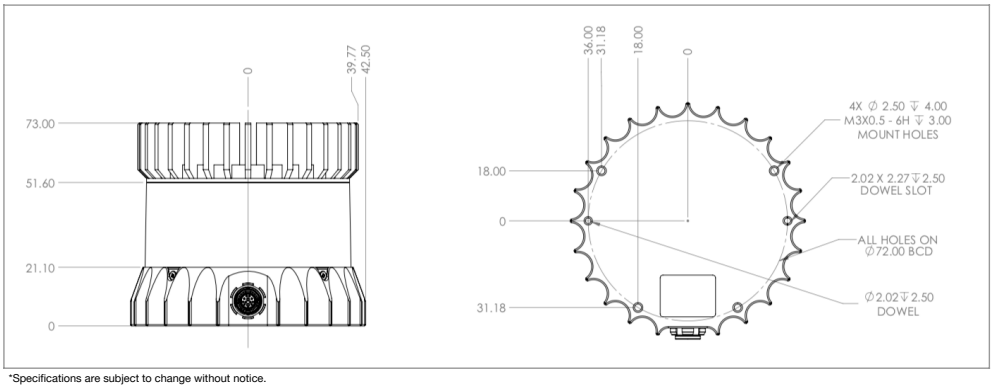
OPERATIONAL

Operating Temperature	-20C to +50C (with Mount)
Storage Temperature	-40C to +105C
Ingress	IP67
Shock	500 m/s2 amplitude, 11 ms duration
Vibration	5 Hz to 2,000 Hz, 3 Grms
Certification	FCC, CE, RoHS

ACCESSORIES

Included Interface Box	PolyCarb/FR4, 100g, 75mm x 50mm x 25mm (LxWxH), 2m CAT6 cable, 24V power adapter, 5m sensor cable
Optional Mount	Aluminum, 530g, 110mm x 110mm x 20.5mm (LxWxH), 4x M8 thru holes

EXTERIOR DIMENSIONS



*Specifications are subject to change without notice.

B Appendix

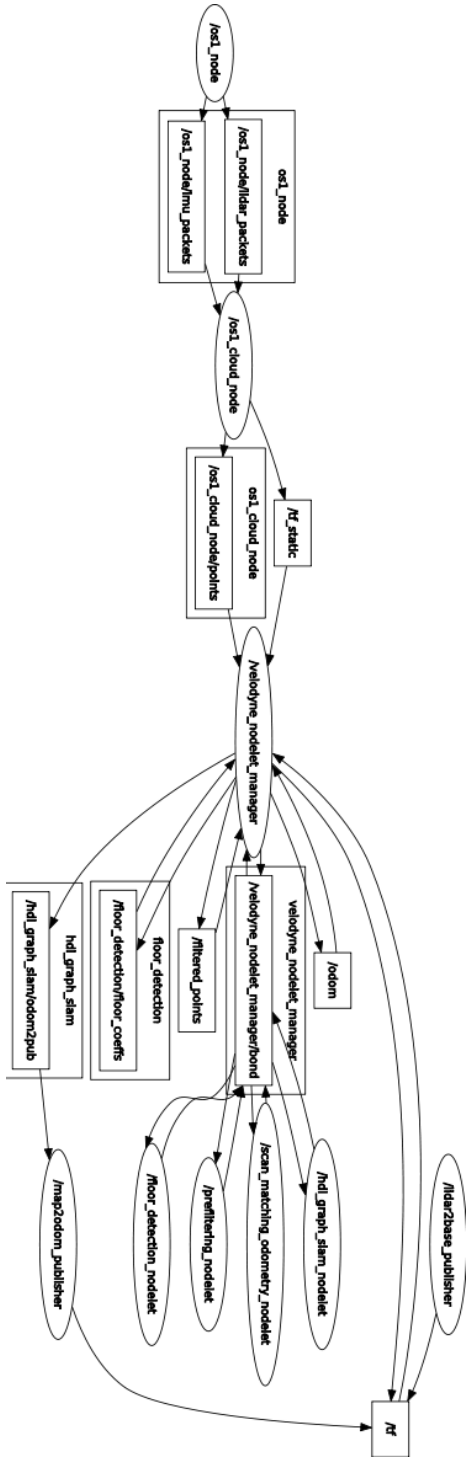
```
<?xml version="1.0"?>
<launch>
  <!-- arguments -->
  <arg name="nodelet_manager" default="velodyne_nodelet_manager"/>
  <arg name="enable_floor_detection" default="true" />
  <arg name="enable_gps" default="false" />
  <arg name="enable_imu_acc" default="true" />
  <arg name="enable_imu_ori" default="true" />


<node pkg="nodelet" type="nodelet" name="$(arg nodelet_manager)"
         args="manager" output="screen"/>
<node pkg="tf" type="static_transform_publisher" name=
        "lidar2base_publisher" args=
                "0 0 0 0 0 0 os1_sensor tf_static 10" />

<node pkg="tf2_ros" type="static_transform_publisher"
        name="laser_boradcaster"
        args="0 0 0.03618 0 0 1 0 os1_sensor os1_lidar"/>
<node pkg="tf2_ros" type="static_transform_publisher"
        name="imu_broadcaster"
        args="0.006254 -0.011775 0.007645 0 0 0 1 os1_sensor
        os1_imu"/>


  <!-- prefiltering_nodelet -->
  <node pkg="nodelet" type="nodelet" name="prefiltering_nodelet"
        args="load hdl_graph_slam/PrefilteringNodelet
        $(arg nodelet_manager)">
    <remap from="/velodyne_points" to="/os1_cloud_node/points"/>
    <param name="use_distance_filter" value="true" />
```

```
    <param name="distance_near_thresh" value="0.1" />
    <param name="distance_far_thresh" value="200.0" />
    <!-- NONE, VOXELGRID, or APPROX_VOXELGRID -->
    <param name="downsample_method" value="VOXELGRID" />
    <param name="downsample_resolution" value="0.1" />
    <!-- NONE, RADIUS, or STATISTICAL -->
    <param name="outlier_removal_method" value="RADUIS" />
    <param name="statistical_mean_k" value="10" />
    <param name="statistical_stddev" value="5" />
    <param name="radius_radius" value="0.5" />
    <param name="radius_min_neighbors" value="2" />
  </node>

  <!-- scan_matching_odometry_nodelet -->
  <node pkg="nodelet" type="nodelet" name=
        "scan_matching_odometry_nodelet"
        args="load hdl_graph_slam/ScanMatchingOdometryNodelet
        $(arg nodelet_manager)">

      <param name="odom_frame_id" value="odom" />
      <!--<remap from="/odom" to="/os1_cloud_node/tf_static"/> -->
      <param name="base_frame_id" value="os1_lidar"/>
      <param name="keyframe_delta_trans" value="1.0" />
      <param name="keyframe_delta_angle" value="1.0" />
      <param name="keyframe_delta_time" value="1.0" />
      <param name="transform_thresholding" value="false" />
      <param name="max_acceptable_trans" value="1.0" />
      <param name="max_acceptable_angle" value="1.0" />
      <param name="downsample_method" value="NONE" />
      <param name="downsample_resolution" value="0.1" />
      <!-- ICP, GICP, NDT, GICP_OMP, or NDT_OMP-->
```

```
    <param name="registration_method" value="NDT" />
    <param name="ndt_resolution" value="1.0" />
    <param name="ndt_num_threads" value="0" />
    <param name="ndt_nn_search_method" value="KDTREE" />
</node>

<!-- floor_detection_nodelet -->
<node pkg="nodelet" type="nodelet" name="floor_detection_nodelet"
      args="load hdl_graph_slam/FloorDetectionNodelet
      $(arg nodelet_manager)"
      if="$(arg enable_floor_detection)">
  <param name="tilt_deg" value="0.0" />
  <param name="sensor_height" value="2.0" />
  <param name="height_clip_range" value="1.0" />
  <param name="floor_pts_thresh" value="512" />
  <param name="use_normal_filtering" value="true" />
  <param name="normal_filter_thresh" value="20.0" />
</node>

<!-- hdl_graph_slam_nodelet -->
<node pkg="nodelet" type="nodelet" name="hdl_graph_slam_nodelet"
      args="load hdl_graph_slam/HdlGraphSlamNodelet
      $(arg nodelet_manager)">

  <!-- Trying to connect Ouster IMU -->
  <remap from="/gpsimu_driver/imu_data" to="os1_cloud_node/imu"/>

  <!-- frame settings -->
  <param name="map_frame_id" value="map" />
  <param name="odom_frame_id" value="odom" />
  <!-- optimization params -->
```

```
<!-- typical solvers: gn_var, gn_fix6_3, gn_var_cholmod,
    lm_var, lm_fix6_3, lm_var_cholmod, ... -->
<param name="g2o_solver_type" value="lm_var_cholmod" />
<param name="g2o_solver_num_iterations" value="512" />
<!-- keyframe registration params -->
<param name="enable_gps" value="$(arg enable_gps)" />
<param name="enable_imu_acceleration" value=
    "$(arg enable_imu_acc)" />
<param name="enable_imu_orientation" value=
    "$(arg enable_imu_ori)" />
<param name="max_keyframes_per_update" value="10" />
<param name="keyframe_delta_trans" value="2.0" />
<param name="keyframe_delta_angle" value="2.0" />
<!-- loop closure params -->
<param name="distance_thresh" value="10.0" />
<param name="accum_distance_thresh" value="15.0" />
<param name="min_edge_interval" value="5.0" />
<param name="fitness_score_thresh" value="0.5" />
<!-- scan matching params -->
<param name="registration_method" value="NDT_OMP" />
<param name="ndt_resolution" value="1.0" />
<param name="ndt_num_threads" value="0" />
<param name="ndt_nn_search_method" value="DIRECT7" />
<!-- edge params -->
<!-- GPS -->
<param name="gps_edge_robust_kernel" value="NONE" />
<param name="gps_edge_robust_kernel_size" value="1.0" />
<param name="gps_edge_stddev_xy" value="20.0" />
<param name="gps_edge_stddev_z" value="5.0" />
<!-- IMU orientation -->
<param name="imu_orientation_edge_robust_kernel" value="NONE"/>
```

```
    <param name="imu_orientation_edge_stddev" value="1.0" />
    <!-- IMU acceleration (gravity vector) -->
    <param name="imu_acceleration_edge_robust_kernel" value="NONE"/>
    <param name="imu_acceleration_edge_stddev" value="1.0" />
    <!-- ground plane -->
    <param name="floor_edge_robust_kernel" value="NONE" />
    <param name="floor_edge_stddev" value="10.0" />
    <!-- scan matching -->
    <!-- robust kernels: NONE, Cauchy, DCS, Fair, GemanMcClure,
        Huber, PseudoHuber, Saturated, Tukey, Welsch -->
    <param name="odometry_edge_robust_kernel" value="NONE" />
    <param name="odometry_edge_robust_kernel_size" value="1.0" />
    <param name="loop_closure_edge_robust_kernel" value="Huber" />
    <param name="loop_closure_edge_robust_kernel_size" value="1.0" />
    <param name="use_const_inf_matrix" value="false" />
    <param name="const_stddev_x" value="0.5" />
    <param name="const_stddev_q" value="0.1" />
    <param name="var_gain_a" value="20.0" />
    <param name="min_stddev_x" value="0.1" />
    <param name="max_stddev_x" value="5.0" />
    <param name="min_stddev_q" value="0.05" />
    <param name="max_stddev_q" value="0.2" />
    <!-- update params -->
    <param name="graph_update_interval" value="1.0" />
    <param name="map_cloud_update_interval" value="5.0" />
    <param name="map_cloud_resolution" value="0.05" />
  </node>

  <node pkg="hdl_graph_slam" type="map2odom_publisher.py"
        name="map2odom_publisher" />
</launch>
```


/os1_node
os1_node
/os1_node/lidar_packets
/os1_node/imu_packets
/os1_cloud_node
os1_cloud_node
/os1_cloud_node/points
/tf_static
/velodyne_nodelet_manager
velodyne_nodelet_manager
/velodyne_nodelet_manager/bond
/odom
/filtered_points
floor_detection
/floor_detection/floor_coeffs
hdl_graph_slam
/hdl_graph_slam/odom2pub
/lidar2base_publisher
/hdl_graph_slam_nodelet
/scan_matching_odometry_nodelet
/prefiltering_nodelet
/floor_detection_nodelet
/map2odom_publisher
/tf

NTNU
Norwegian University of
Science and Technology