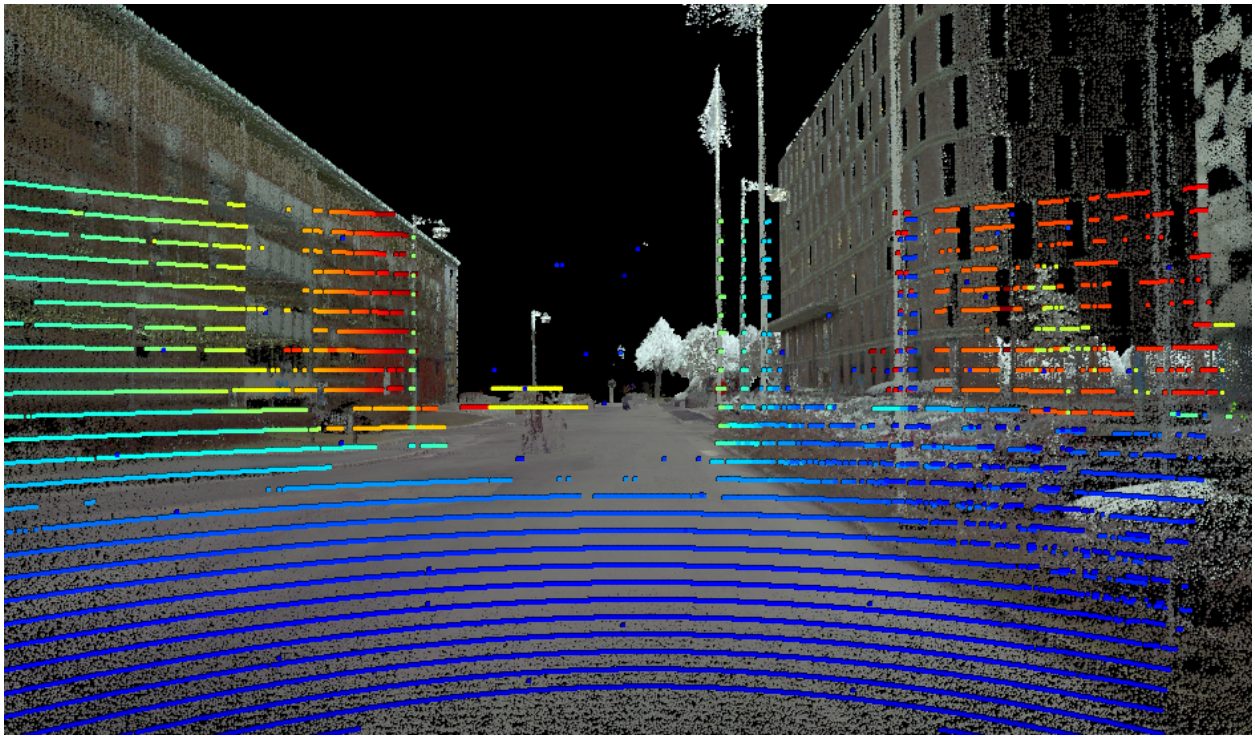
CHALMERS
UNIVERSITY OF TECHNOLOGY



Precision localization in dense point cloud maps

Master's thesis in Systems, Controls and Mechatronics

GUNNAR BOLMVALL

MÅNS ÖSTMAN

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis 2018:EX056

Precision localization in dense point cloud maps

GUNNAR BOLMVALL

MÅNS ÖSTMAN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signals Processing and Biomedical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2018

Precision localization in dense point cloud maps
GUNNAR BOLMVALL
MÅNS ÖSTMAN

Supervisor: Lars Hammarstrand, Electrical Engineering
Examiner: Lars Hammarstrand, Electrical Engineering

Master's Thesis 2018:EX056
Department of Electrical Engineering
Division of Signals Processing and Biomedical Engineering
Signal Processing Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by [Name of printing company]
Gothenburg, Sweden 2018

Precision localization in dense point cloud maps
GUNNAR BOLMVALL
MÅNS ÖSTMAN
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Today autonomous vehicles are a very hot topic within the technology industry. Traditional car and software companies are striving towards the same goal of delivering fully autonomous cars within a near future. To have a safe and durable autonomous vehicle, accurate real time localization in the environment is crucial. Global Navigation Satellite System (GNSS) based localization typically struggles in urban environments as the signal quality is worse than in open areas. By using LiDAR sensors, high accuracy localization can be achieved even in these areas. In order to evaluate these algorithms a ground truth is needed, in the case of urban environments this might be hard to compute due to the limitations of GNSS. An alternative approach to computing ground truth would be to utilize the onboard LiDAR sensor(s) to find the location within a dense point cloud map.

In this thesis we propose a method to perform offline localization, using point cloud data from a test vehicle, within a dense point cloud map. The method computes a transformation using the Generalized ICP algorithm which aligns a LiDAR scan with the point cloud map. The transformation is then modeled as measurements in a RTS-smoother which estimates bias along the trajectory in six degrees of freedom. The bias is then added to the initial trajectory to get the refined trajectory.

The results shows the refined trajectory to be more accurate than the initial trajectory, however not precise enough to be used for ground truth computation. The largest limiting factor is thought to be that the Generalized ICP transformations are sensitive to noise, such as dynamic objects, in the LiDAR data. This makes the algorithm heavily dependent on removing all false transformations, which has been a hard problem to solve. With some modifications the algorithm could probably be made robust and accurate enough for the intended purpose of ground truth computation.

Keywords: Localization, LiDAR, RTS-smoothing, Generalized ICP

Acknowledgements

We would like to thank our supervisor and examiner Lars Hammarstrand for the continuous support during project. We would also like to thank Erik Stenborg for his useful input during the first half of the project. Finally, we would like to thank Arpit Karsolia and Christian Berger for their help collecting and extracting data from the test vehicle.

Gunnar Bolmvall & Måns Östman, Gothenburg, June 2018

Contents

List of Figures

List of Tables

Acronyms

COPPLAR Campusshuttle Cooperative Perception and Planning Platform.

ENU East, North, Up.

GICP Generalized Iterative Closest Point.
GNSS Global Navigation Satellite System.

ICP Itarative Closest Point.
ISO International Organization for Standardization.

LiDAR Light Detection And Ranging.

PCA Principle Component Analysis.
PCL Point Cloud Library.

RMS Root Mean Square.

SLAM Simultaneous Localization And Mapping.

WLS Weighted Least Squares.

1
Introduction

This chapter introduces the problem this thesis aims to solve, along with some relevant background.

1.1 Background

Today autonomous vehicles are a very hot topic within the technology industry. Traditional car and software companies are striving towards the same goal of delivering fully autonomous vehicles within a near future. To reach the goal, universities and companies are conducting research to find new technologies that ensures safety and durability of the vehicle. SAFER Vehicle and the Traffic Centre at Chalmers is one example of a cooperative competence center for research and innovation within the Swedish industry, academia and authorities [1].

Between 2016 and 2018, SAFER is financing the Campusshuttle Cooperative Perception and Planning Platform (COPPLAR) project, which aims to take the first steps towards a cooperative self-driving vehicle that can handle challenging city traffic and changing weather conditions [2]. The project is a cooperation between SAFER, Chalmers University of Technology, University of Gothenburg, Volvo Car Corporation, Veoneer, Zenuity and AstaZero. The developed self-driving vehicle is supposed to navigate the route between Chalmers' two campuses.

An autonomous vehicle requires many systems in order for it to work independently. One such system is tasked with localization of the vehicle. Localization has typically been done using Global Navigation Satellite System (GNSS) sensors, however these are rather bad at handling localization withing urban areas due to a loss of signal quality. As the COPPLAR project has to handle urban driving, other localization methods has to be used. One popular approach is to use Light Detection And Ranging (LiDAR) sensors paired with a pre-collected point cloud reference map in order to accurately compute the vehicle location. LiDAR based localization that is used online in a vehicle has to compromise accuracy in order for the computations to be able to run in real time. This means that in order to evaluate how accurate an algorithm is, ground truth data is needed. This is traditionally obtained by post-processing GNSS data which poses a problem as even with post-processing, the inaccuracies in urban areas are likely to persist. The solution to computing ground truth data could be to utilize the LiDAR data when post-processing the location of the vehicle, as most of the time, not all of it is utilized in online algorithms. This

thesis will examine such an approach for computing ground truth data.

1.2 Objective/Purpose

The purpose of this thesis is to create a high precision localization algorithm that can be used to compute ground truth data for localization. In order to fulfill the purpose, a vehicle trajectory has to be estimated in relation to an accurate pre-collected dense point cloud map. The vehicle poses are to be estimated in six degrees of freedom: latitude, longitude, altitude, roll, pitch and yaw.

1.3 Scope

Some limitations have been put on the thesis in order to keep the work load from getting too extensive, these are:

- The pre-collected point cloud map is considered ground truth as the localization is to be computed in relation to it. Further the point cloud map will not be processed in any way to remove occasional noise etc.
- The sensor data from the vehicle is internally filtered by a positioning system from Applanix that has been installed in the test vehicle. As the output from the system is a filtered trajectory of the vehicle poses, it won't be further processed, instead focus will be on correcting it.
- Post-processing of the LiDAR data will be kept to a minimum. Noise, such as dynamic objects will not be removed.

1.4 Related work and contribution

There exist some research within the topic of localization using LiDAR-data and several different approaches to solving the problem have been proposed. What many have in common with this Thesis is that they utilize a similar dense point cloud to localize within.

In [3, 4, 5, 6], a visual method to localize a vehicle within a point cloud map. This is done by projecting the point cloud data to a proposed location of the camera and find the most likely position by comparing the synthetic image with the one captured by the vehicle camera in some manner. Projection of the point cloud has been explored briefly in this thesis, however, mostly for evaluation purposes. In [7] a high density point cloud is used for localization and map extraction. Although the focus of the article is on map extraction, the point cloud dataset has been recorded by the same third-party company that recorded the data for this Thesis. Some other articles that deal with point cloud localization include [8], where curbs are extracted from the vehicle LiDAR-data and matches to a 2D-map of existing curbs. In [9] the dense point cloud is used to extract landmarks that can be matched with landmarks

extracted from the vehicle LiDAR-data.

All of the mentioned articles have one thing in common, which is that they are striving towards solutions that operate in real time. This is rather natural as the demand from the automotive industry lies within real time localization. This means that most of the algorithms don't utilize the complete point cloud data. As this Thesis doesn't have any constraints on computational complexity, the limits that most of the articles deal with doesn't apply, opening up for approaches that are more computational heavy.

2
Data set and sensors

This chapter describes the used data set, vehicle sensors and the evaluated route.

2.1 Pre-collected dense point cloud map

The dense point cloud map has been collected and post processed by a third party company. Each point in the map has a global cartesian position, label, color and intensity. The position is recorded according to the SWEREF99 coordinate system which is the Swedish official reference system of mapping geodetic coordinates to cartesian coordinates [10]. The label is a classification of either 'road surface', 'ground' or 'other'.

Along the route, the point cloud has been split into boxes of size 200 × 200m to facilitate processing of the data. Depending on the number of points captured, each box may consist of 100 to 35 000 000 points. Figure 2.1 shows an example of what the point cloud looks like. As can be seen in the figure, ghost objects/silhouettes of trams, cars and buses can be observed. This is due to limitations in the post processing software.

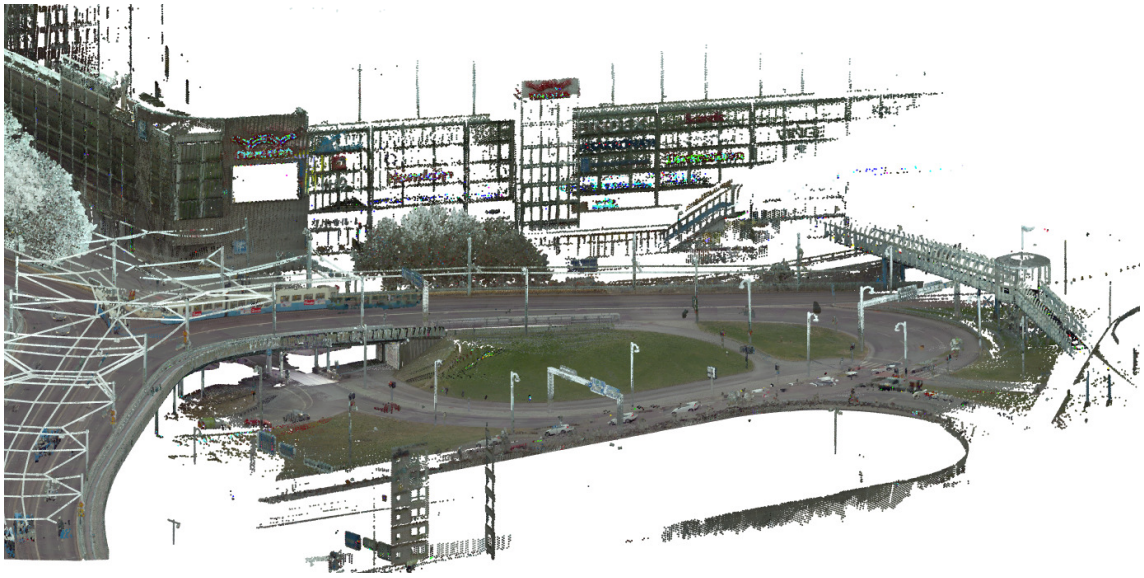


Figure 2.1: Point cloud map of the exit from Götaälvbron consisting of 17′221′981 points.

2.2 Vehicle sensors

The test vehicle is a Volvo XC90 equipped with external logging equipment, an Applanix Global Navigation Satellite System (GNSS) and a Velodyne Light Detection And Ranging (LiDAR). The sensor positon is shown in Figure 2.2. Using these sensors, a dataset has been generated consisting of global positions and Velodyne point clouds.

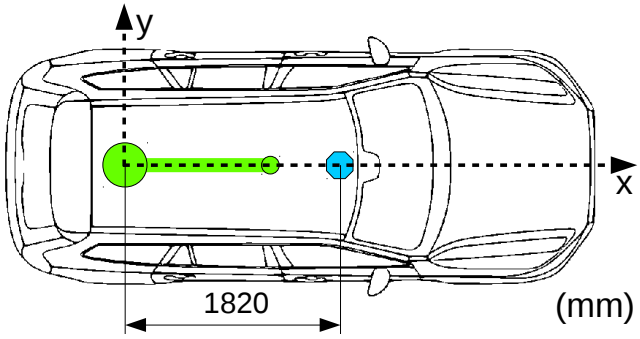


Figure 2.2: Position of Applanix (●) and Velodyne (⬣).

2.2.1 Applanix LV420

The Applanix LV420 is a high accurate GNSS sensor with two external antennas mounted on top of the car. The sensor can achieve a Root Mean Square (RMS) accuracy of less than $10cm$ in (X,Y,Z) and 0.2^o in (roll, pitch, heading) under good conditions [11]. Log frequency is $200Hz$. The main purpose of this sensor is to capture an accurate global position and orientation of the vehicle.

2.2.2 Velodyne HDL-32E

The Velodyne Light Detection And Ranging (LiDAR) is mounted on top of the car right above the front windshield. The sensor has 32 laser/detector pairs, a range of up to 70m, 10 Hz frame rate, field of view of 360 degrees and it captures up to 72000 points per frame [12]. Due to the mounting position the car covers about 90 degrees looking backwards of the car, which results in a field of view of about 270 degrees. The main purpose of this sensor is capture the surroundings relative to the car. Figure 2.3 shows an example of one Velodyne point cloud.

2.3 Route

The route goes between Chalmers' two campuses in Gothenburg, Campus Johanneberg and Campus Lindholmen. In Figure 2.4 the route is marked in both directions along with the boxes which the dense point cloud has been split into.

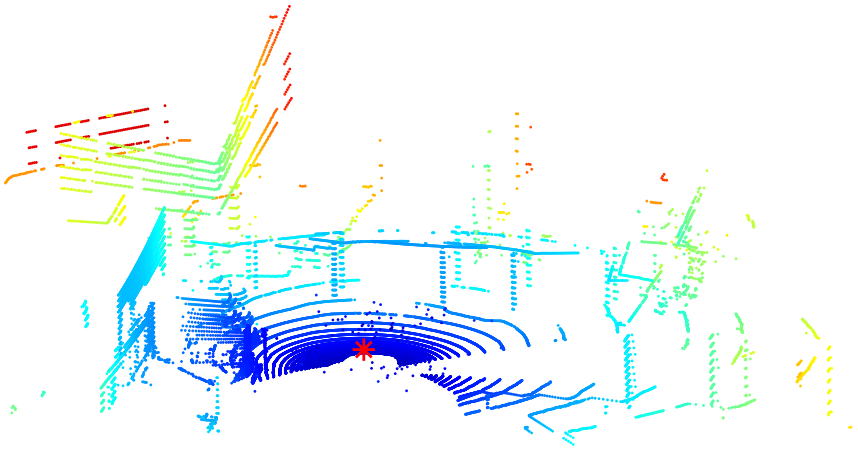


Figure 2.3: A point cloud from the Velodyne LiDAR. The range is color coded with blue close to the sensor goes to red far away. The Velodyne position is marked with an asterisk (*).

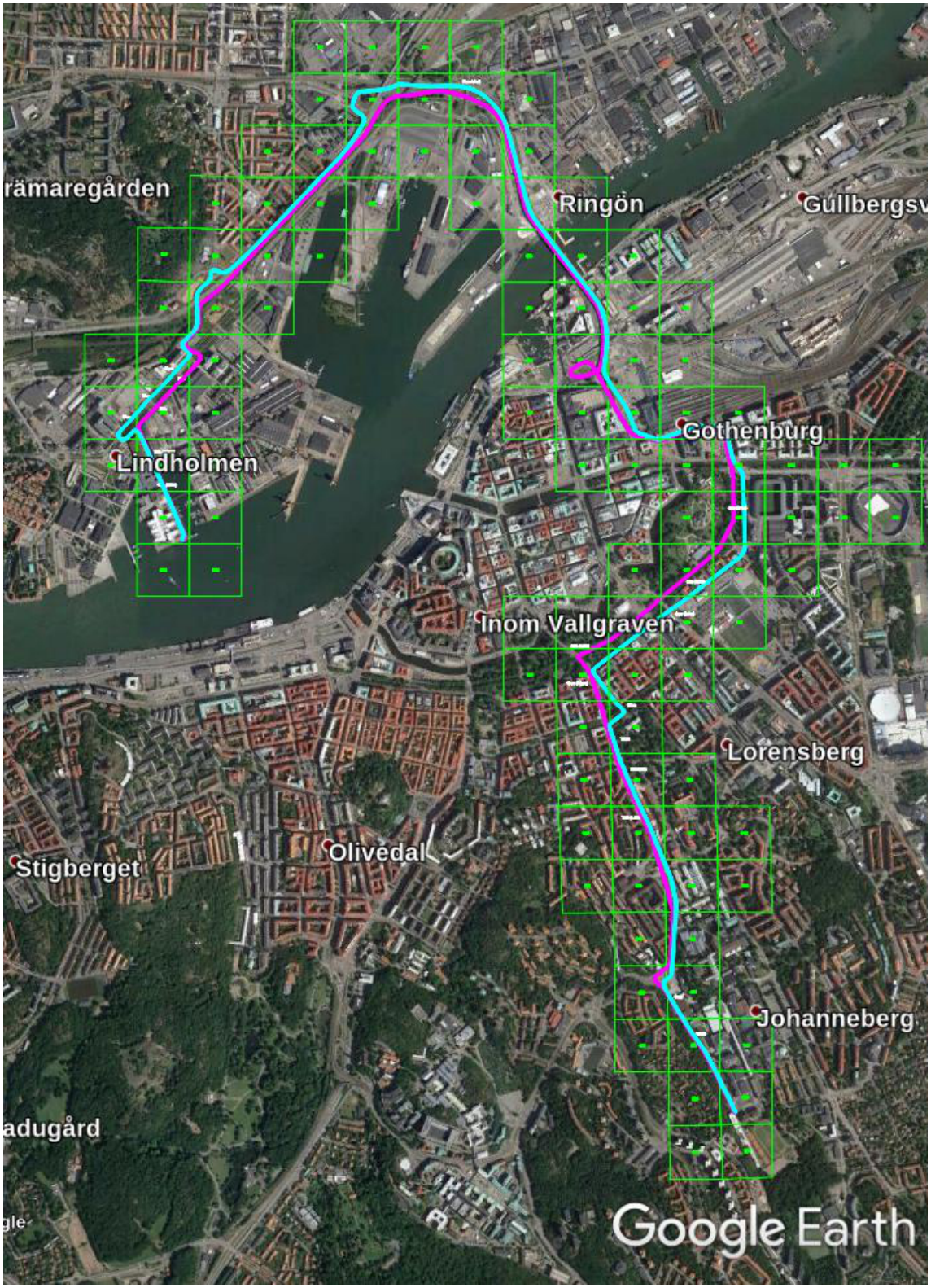


Figure 2.4: Cyan trajectory marks route from Johanneberg to Lindholmen and pink in the opposite direction. The green boxes shows how the dense point cloud data has been divided into smaller segments.

3
Theory

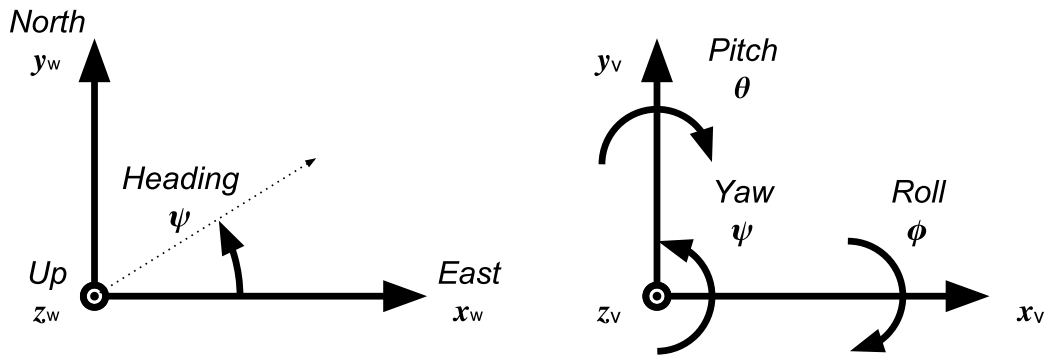
This chapter covers relevant theory for the thesis.

3.1 Geometry

Representing a local orientation and position (pose) of an object in a global environment is a common problem in fields of computer graphics, camera pose estimation, 3D reconstruction and Simultaneous Localization And Mapping (SLAM) etc. To solve these types of problems one needs a way of transforming the pose of observed objects in the local coordinate frame to a pose in a global coordinate frame.

The global frame (world frame) $\{W\}$ is often defined using the Cartesian coordinates and an East, North, Up (ENU) axis convention. Here the x-axis align with east, y-axis north and the z-axis up as shown in Figure 3.1a.

The vehicle frames $\{V\}$ on land are often defined using the International Organization for Standardization (ISO) 8855 framework. In ISO systems, the x-axis points in the driving direction, y-axis to the left and z-axis up. The rotations of the body follows the right hand rule of orientation with roll (θ) in x, pitch (ϕ) in y and yaw (ψ) in z. This is shown in Figure 3.1b.



(a) World frame **(b)** Vehicle frame

Figure 3.1: Coordinate systems of two reference frames.

3.1.1 Affine transformations

An affine transformation matrix can include any sequence and combination of translation, rotation, scaling, homothety, shear mapping and composition. When transforming between world and vehicle frames, rotation and translation are primarily of interests. When performing rotation in 3D, the sequence of rotation is important. The euler angles (ϕ, θ, ψ) are measured relative to the world frame and by following the right hand rule and ISO standard the final rotation matrix of an vehicle is calculated as

$$\mathbf{R} = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi). \tag{3.1}$$

The translation consists of vector specifying the position of the vehicle frame in the world frame as $\mathbf{t} = \begin{bmatrix} x & y & z \end{bmatrix}^T$. To perform affine transformation for a point $\mathbf{x_v} \in \{V\}$ to $\mathbf{x_w} \in \{W\}$ one combines the rotation and translation as

$$\mathbf{x}_w = \mathbf{R}_v\mathbf{x}_v + \mathbf{t}. \tag{3.2}$$

This equation can be more compactly written using homogeneous coordinates. The definition of a homogeneous coordinate is that an arbitrary point $\mathbf{x} = \begin{bmatrix} x & y & z \end{bmatrix}^T$ can be multiplied with any scalar $\lambda \neq 0$ and still represent the same point. This is done my adding an extra dimension to the Cartesian coordinate as

$$\begin{bmatrix} \mathbf{x}_w \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_v & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}_v \\ 1 \end{bmatrix} \tag{3.3}$$

To simplify the equation even more, the following notation is used

$$\tilde{\mathbf{x}}_w = \mathbf{T}\tilde{\mathbf{x}}_v. \tag{3.4}$$

Since $\mathbf{T}$ becomes an invertable homogeneous transformation matrix, transforming from $\mathbf{x_w} \in \{W\}$ to $\mathbf{x_v} \in \{V\}$ is done by

$$\tilde{\mathbf{x}}_v = \mathbf{T}^{-1}\tilde{\mathbf{x}}_w. \tag{3.5}$$

Using homogeneous coordinates, consecutive transformations are easily done my multiplying the transformations in sequence as

$$\tilde{\mathbf{x}}_w = \mathbf{T}_1\mathbf{T}_2\tilde{\mathbf{x}}_v. \tag{3.6}$$

3.2 Bayesian Filtering and smoothing

Bayesian filtering is used to estimate the states of general probabilistic state space models [13]. These models consists of a sequence of conditional probability distributions on the form:

$$x_k \sim p(x_k|x_{1:k-1}, y_{1:k-1})$$
$$y_k \sim p(y_k|x_{1:k}, y_{1:k-1})$$

where x_k is the state and y_k is the measurement at time step k. $p(x_k|x_{1:k-1}, y_{1:k-1})$ is the process model which describes the dynamics of the system and $p(y_k|x_{1:k}, y_{1:k-1})$ is the measurement model which describes the distribution of measurements given the state.

The model is assumed to be Markovian, giving it two certain properties. The first property is that the state sequence forms a Markov chain, meaning that x_k given x_{k-1} is independent of all other previous states. The second property is that the measurements are conditionally independent, this implies that the measurement y_k given the current state x_k is independent of all previous states and measurements. With the use of these properties the process and measurement models can be simplified as

$$\begin{aligned} p(x_k|x_{1:k-1}, y_{1:k-1}) &= p(x_k|x_{k-1}) \\ p(y_k|x_{1:k}, y_{1:k-1}) &= p(y_k|x_k). \end{aligned}$$

The aim of Bayesian filtering is to compute the marginal posterior distribution of the state x_k given the measurement history $y_{1:k}$ up to time step k:

$$p(x_k|y_{1:k}).$$

This can be achieved by recursively computing the predicted distribution $p(x_k|y_{1:k-1})$ and posterior distribution $p(x_k|y_{1:k})$ using the Bayesian filtering equations. There are three steps involved in this computation: initialization, prediction and updating. The recursion start from the prior distribution $p(x_0)$, this is the initialization of the computation. The predicted distribution at time step k can then be computed by the Chapman-Kolmogorov equation

$$p(x_k|y_{1:k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|y_{1:k-1})dx_{k-1}. \tag{3.7}$$

The postrior distribution can be computed by using Bayes' rule:

$$p(x_k|y_{1:k}) = \frac{1}{Z_k}p(y_k|x_k)p(x_k|y_{1:k-1}) \tag{3.8}$$

where Z_k is the normalization constant given by

$$Z_k = \int p(y_k|x_k)p(x_k|y_{1:k-1})dx_k. \tag{3.9}$$

3.2.1 Kalman filter

If the process and measurement models are assumed to be linear Gaussian it is possible to solve the Bayesian filter equations with a Kalman filter [13]. The simplified filter model will be on the form

$$\begin{aligned} x_k &= A_{k-1}x_{k-1} + q_{k-1}, & q_{k-1} &\sim \mathcal{N}(0, Q_{k-1}) \\ y_k &= H_k x_k + r_k, & r_k &\sim \mathcal{N}(0, R_k) \end{aligned} \tag{3.10}$$

where x_k is the state, y_k is the measurement, A_{k-1} is the transition matrix of the process model, q_k is the process noise, H_k is the measurement model matrix and r_k

is the measurement noise. The posterior distributions for the filtering problem will under these assumptions also be Gaussian and can be represented by their mean $\hat{x}_{k|k}$ and covariance $P_{k|k}$. The Kalman filter computes the mean and covariance of the posterior distributions through the following prediction and update steps.

- Prediction

$$\begin{aligned} \hat{x}_{k|k-1} &= A_{k-1}\hat{x}_{k-1|k-1} \\ P_{k|k-1} &= A_{k-1}P_{k-1|k-1}A_{k-1}^T + Q_{k-1} \end{aligned} \tag{3.11}$$

- Update

$$\begin{aligned} v_k &= y_k - H_k\hat{x}_{k|k-1} \\ S_k &= H_kP_{k|k-1}H_k^T + R_k \\ K_k &= P_{k|k-1}H_k^TS_k^{-1} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_kv_k \\ P_{k|k} &= P_{k-1|k-1} - K_kS_kK_k^T \end{aligned} \tag{3.12}$$

Recursion is started from prior mean x_0 and covariance P_0.

3.2.2 Rauch-Tung-Striebel smoother

The Rauch-Tung-Striebel smoother, which is also called the Kalman smoother, can be used for computing the optimal smoothing solution $p(x_k|y_{1:N})$ to the linear filtering model (3.10) [13]. The difference from the solution computed by the Kalman filter is that the smoothed solution is conditional on the whole sequence of measurement data $y_{1:N}$. The Kalman filter on the other hand only utilizes the measurements obtained up to time step k, that is, $y_{1:k}$. The smoother works by extending the Kalman filter with a backward recursion using equations given as

$$\begin{aligned} \hat{x}_{k+1|k} &= A_k\hat{x}_{k|k} \\ P_{k+1|k} &= A_kP_{k|k}A_k^T + Q_k \\ G_k &= P_{k|k}A_k^TP_{k+1|k}^{-1} \\ \hat{x}_{k|N} &= \hat{x}_{k|k} + G_k(\hat{x}_{k+1|N} - \hat{x}_{k+1|k}) \\ P_{k|N} &= P_{k|k} - G_k[P_{k+1|k} - P_{k+1|N}]G_k^T \end{aligned} \tag{3.13}$$

where $\hat{x}_{k|k}$ and $P_{k|k}$ are the mean and covariance computed by the Kalman filter. The recursion is started from the last time step N , with $\hat{x}_{N|N} = \hat{x}_{N|k}$ and $P_{N|N} = P_{N|k}$. Note that the first two of the equations are simply the Kalman filter prediction equations.

3.2.3 Fusion of independent estimates

In [14], it is described how the *sensor fusion formula* can be used to fuse two direct measurements, y_1 and y_2, of some parameter x . The formula can be extended to independent estimates by converting the linear model $y_k = H_kx + e_k$ to

equivalent measurements by using a local Weighted Least Squares (WLS) estimate $\hat{x}_k = (H_k^T R_k^{-1} H_k)^{-1} H_k^T R_k^{-1} y_k$ with the properties

$$\begin{aligned} E(\hat{x}_1) = E(\hat{x}_2) = x, \\ Cov(\tilde{x}_1) = P_1, \\ Cov(\tilde{x}_2) = P_2. \end{aligned}$$

Since the estimates can be interpreted as measurements in a linear model,

$$\begin{aligned} \hat{x}_1 = x + \tilde{x}_1, \quad Cov(\tilde{x}_1) = P_1, \\ \hat{x}_2 = x + \tilde{x}_2, \quad Cov(\tilde{x}_2) = P_2, \end{aligned}$$

the WLS estimate gives

$$P = (P_1^{-1} + P_2^{-1})^{-1} \tag{3.14}$$
$$\hat{x} = P(P_1^{-1}\hat{x}_1 + P_2^{-1}\hat{x}_2). \tag{3.15}$$

The formula requires that the two estimates are independent.

3.3 Iterative closest point

The Itarative Closest Point (ICP) algorithm was introduced by Besl and McKay [15]. The algorithm is used for aligning two corresponding point clouds

$$\begin{aligned} A = \{a_i\}_{i=1...N} \\ B = \{b_i\}_{i=1...N} \end{aligned}$$

where A is denoted as the source cloud and B is denoted as the target point cloud. The algorithm computes the transformation matrix $\mathbf{T}$ that transform the source cloud to the target cloud given perfect correlation

$$B = \mathbf{T}A. \tag{3.16}$$

The computation is done iterative in a few steps as follows:

1. The source cloud is transformed according to transformation $\mathbf{T}$.
2. Correspondences are computed for the point clouds by finding the nearest neighbor for each point.
3. A transformation $\mathbf{T}$ is computed that minimizes the distance between the corresponding points.
4. The computation is terminated once some predefined condition has been fulfilled.

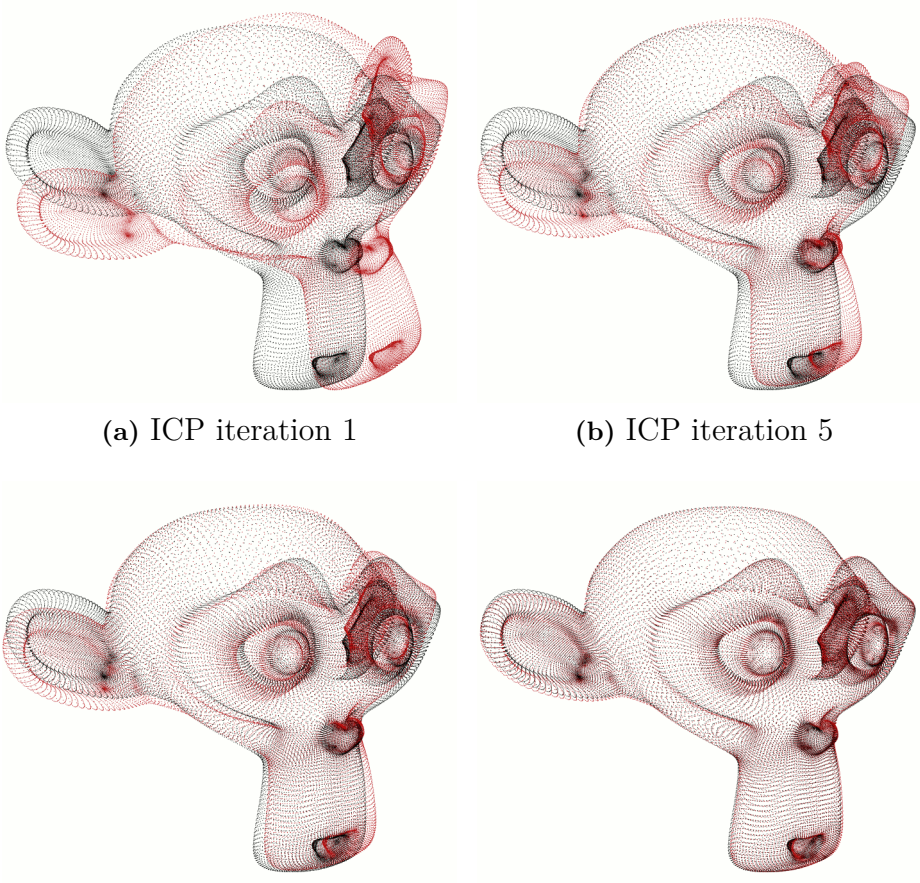
The iterations start with $\mathbf{T} = \mathbf{T_0}$ which is based on prior knowledge about the transformation. Correspondences between the point cloud are computed in two steps. First the closest point in B is computed for all points in A, giving proposed corresponding pairs $a_i = b_i$. Since the point clouds rarely fully overlap with each other there will be points in both cloud not corresponding to any point in the other. Hence for a proposed pair to be considered corresponding the distance between them has to be lower than some defined threshold d_{max}.

$$b_i - \mathbf{T}a_i < d_{max}.$$

Once the set of corresponding points has been computed $\mathbf{T}$ is calculated as

$$\mathbf{T} = \underset{\mathbf{T}}{argmin} \frac{1}{N_p} \sum_{i=1}^{N_p} ||b_i - \mathbf{T}a_i||^2 \tag{3.17}$$

where a_i and b_i are corresponding points in homogeneous coordinates. An converging example of the ICP algorithm is shown in Figure 3.2.



(a) ICP iteration 1 **(b)** ICP iteration 5

(c) ICP iteration 10 **(d)** ICP iteration 20

Figure 3.2: Illustration of how a point cloud converges iteratively using the ICP algorithm. The red cloud represents the source cloud and the black one the target cloud.

3.3.1 Generalized Iterative Closest Point

Generalized Iterative Closest Point (GICP) is a modified variant of the standard ICP that includes the surface information of the two point clouds in a probabilistic model. This effects the minimization step of the original algorithm while the rest of the computations remains the same [16].

In GICP the point clouds A and B are assumed to be measurements of some underlying set of points where the clouds are indexed according to their correspondences

$$\hat{A} = \{\hat{a}_i\}_{i=1\ldots N}$$
$$\hat{B} = \{\hat{b}_i\}_{i=1\ldots N}.$$

Each measured point in A and B can then be drawn from the modeled Gaussian distribution to represent the uncertainty of the point in the alignment process as

$$a_i \sim \mathcal{N}(\hat{a}_i, C_i^A)$$
$$b_i \sim \mathcal{N}(\hat{b}_i, C_i^B).$$

where $\{C_i^A\}$ and $\{C_i^B\}$ are covariance matrices associated with the measured points.

Since the shape of a surface can be seen as piece-wise differentiable, the clouds sampled from the real world are assumed to be locally planar. With two measurements of the same scene from two different perspectives, the measurements are not expected to correspond to the exact same point but close and located on the same plane. This is modeled using the covariance matrix as

$$\mathbf{C} = \begin{pmatrix} \epsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.18}$$

where the covariance in the surface normal direction is set to a small constant ϵ and the covariance in the plane direction is set to a large constant, 1. To make $\mathbf{C}$ represent the measurement covariance, ϵ has to be align with the surface normal vector which is done by rotating the matrix as

$$C_i = \mathbf{R}_{v_i}\mathbf{C}\mathbf{R}_{v_i}^T \tag{3.19}$$

where $\mathbf{R}_{v_i}$ corresponds to the orientation of the surface normal vector v_i. Estimating the surface normal can be done in many ways but one common way is to use Principle Component Analysis (PCA). First a covariance matrix is calculated of the k closest points to i and from this matrix the surface normal vector is determined by the eigenvector with the smallest eigenvalue.

Given perfect correspondences for all points in the point clouds, and the correct transformation, $\mathbf{T}^*$, we know that

$$\hat{b}_i = \mathbf{T}^*\hat{a}_i.$$

For an arbitrary rigid transformation $\mathbf{T}$, we define $d^{(\mathbf{T})} = b_i - \mathbf{T}a_i$, and consider the distribution from which $d^{(\mathbf{T}^*)}$ is drawn,

$$\begin{aligned} d^{(\mathbf{T}^*)} &\sim \mathcal{N}(\hat{b}_i - (\mathbf{T}^*)\hat{a}_i,\ C_i^B + (\mathbf{T}^*)C_i^A(\mathbf{T}^*)^T) \\ &= \mathcal{N}(0,\ C_i^B + (\mathbf{T}^*)C_i^A(\mathbf{T}^*)^T). \end{aligned}$$

To find the transformation matrix $\mathbf{T}$, maximum likelihood estimation can be used

$$\mathbf{T} = \underset{\mathbf{T}}{argmax} \prod_i p(d^{(\mathbf{T})}) = \underset{\mathbf{T}}{argmax} \sum_i log(p(d^{(\mathbf{T})})) \tag{3.20}$$

which can be simplified as

$$\mathbf{T} = \underset{\mathbf{T}}{argmin} \sum_i d^{(\mathbf{T})^T}(C_i^B + \mathbf{T}C_i^A\mathbf{T}^T)^{-1}d^{(\mathbf{T})}. \tag{3.21}$$

This defines the key step in ICP and replaces the old minimization problem.

4
Method

This chapter present how the algorithm has been implemented. For the most part MATLAB is used as the programming framework. To perform point cloud operations the C++ library Point Cloud Library (PCL) is used. PCL is a standalone, large scale library for 3D point cloud processing and has many point cloud algorithms implemented, including GICP [17].

4.1 Overview

The initial trajectory gathered by the Applanix system gives estimates of the pose of the vehicle. For each estimate $\hat{p}_i$ there is assumed to be some bias b_i that when added to the measurement gives the true vehicle pose p_i

$$p_i = \hat{p}_i + b_i, \qquad i \in I = \{1, 2, ..., N\}.$$

The main idea of the developed algorithm is to estimate the bias b_i. The bias is computed for a subset of trajectory indices by the GICP algorithm. This subset of bias measurements is used in a RTS-smoother to estimate the bias along the rest of the trajectory. The final trajectory is computed by fusing the bias with the initial trajectory.

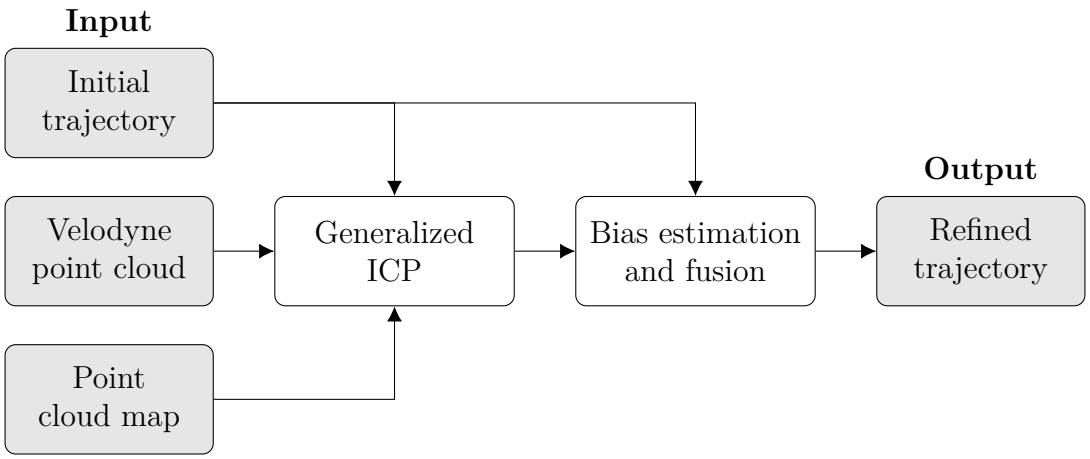


Figure 4.1: A simple flow chart of the implemented algorithm.

4.2 Initial trajectory

The initial trajectory is measured at the location of the rear Applanix antenna, which is mounted on the roof of the vehicle and performs internal filtering of the raw measurements.

4.2.1 Trajectory position

The poses of the initial trajectory can be transformed to represent the Velodyne sensor, instead of the Applanix antenna. This is done by translating the pose by $\delta = 1.82m$ along the x-axis of the vehicle. By using the offset δ, heading $\boldsymbol{\psi}$ and pitch $\boldsymbol{\theta}$ the trajectory offset is calculated using spherical to cartesian coordinates as

$$\begin{bmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{y} \\ \Delta\mathbf{z} \end{bmatrix} = \begin{bmatrix} \sin(\boldsymbol{\theta})\cos(\boldsymbol{\psi}) \\ \cos(\boldsymbol{\theta})\sin(\boldsymbol{\psi}) \\ \sin(\boldsymbol{\theta}) \end{bmatrix} \delta. \tag{4.1}$$

The velodyne trajectory is then obtained by adding the offset to the initial trajectory as

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \\ \mathbf{z} \end{bmatrix} + \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{y} \\ \Delta\mathbf{z} \end{bmatrix}. \tag{4.2}$$

4.2.2 Height correction

The GNSS is relatively inaccurate at measuring height, especially where few satellites are in range. Since the position of the Velodyne sensor relative the road surface is expected to be close to constant and the height of the road is known, a correction vector can easily be computed as

$$\Delta\mathbf{z} = \mathbf{h} - Z_{ref}. \tag{4.3}$$

Here each index of $\mathbf{h}$ is calculated as the average distance from the corresponding trajectory pose to the 10 closest points labeled as road surface in the point cloud map. Z_{ref} is the distance from the ground to the Velodyne sensor when the vehicle is stationary. To keep small oscillations due to the vehicle dynamics, the correction vector is smoothed using moving average with an arbitrary selected window size of 50. Lastly the height is updated by subtracting the correction from the original measured height as

$$\mathbf{z} = \mathbf{z} - \Delta\mathbf{z}. \tag{4.4}$$

4.3 Velodyne point cloud

The point cloud data captured by the Velodyne sensor is stored without any processing and therefore has to be processed somewhat to properly represent the vehicle

surroundings. The following section describes how the Velodyne data is processed before it can be used by the GICP algrithm.

4.3.1 Ego-motion compensation

The Velodyne LiDAR is operating at $10Hz$, meaning each revolution takes $0.1s$ to complete. As the vehicle is moving during the duration of each revolution, the captured point cloud has to be shifted according to the vehicle movement in order to correctly correspond to the surroundings. Depending on how late in a scan a point was captured it will have to be shifted by different amounts. This is done by moving each point by the same amount as the vehicle has traveled since the start of the scan.

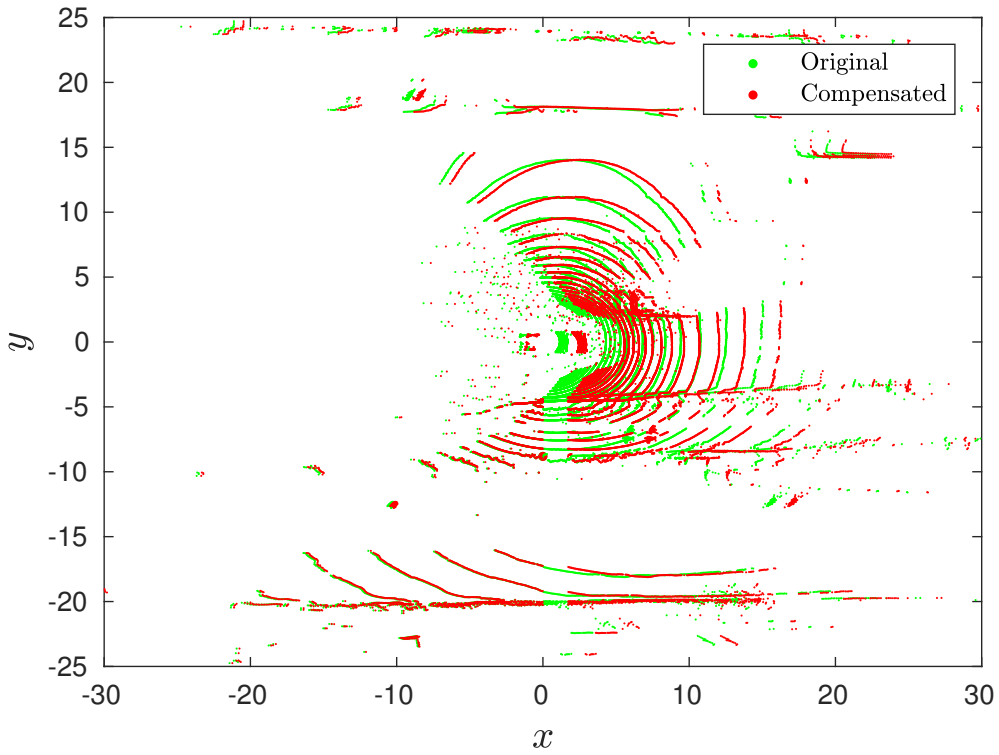


Figure 4.2: Birdview of the original velodyne scan in green and the ego-motion compensated in red.

If the car is assumed to be moving linearly at constant speed during the scan cycle, calculations are simplified since compensation only has to be made separately along the local x and z axes. The computation is made by multiplying the time since the start of a scan by the velocity of the vehicle, which is expressed as

$$\begin{aligned} \mathbf{x} &= \mathbf{x}_{in} + \frac{1}{2\pi}\left(\frac{3\pi}{2} - \boldsymbol{\varphi}\right)\mathbf{v}t \\ \mathbf{z} &= \mathbf{z}_{in} + \frac{1}{2\pi}\left(\frac{3\pi}{2} - \boldsymbol{\varphi}\right)\mathbf{v}_z t \end{aligned} \tag{4.5}$$

where $\mathbf{v} = \sqrt{\mathbf{v}_x^2 + \mathbf{v}_y^2}$, t the time it takes to do one scan and $\boldsymbol{\varphi}$ is the angles to the points. $\boldsymbol{\varphi}$ is calculated by transforming the cartesian x and y coordinates to polar coordinates, since the scan starts at $3\pi/2$ and rotates clockwise the angle has to be calculated as $3\pi/2 - \boldsymbol{\varphi}$. Figure 4.2 shows the result of the compensation.

4.3.2 Denoise filter

To reduce the noise in the Velodyne cloud, a denoise filter based on *k-nearest neighbour* is applied. The filter removes points if it has less than 8 neighbouring points within a radius of $1m$. The choice of neighbouring points and radius has been selected arbitrarily.

4.3.3 Cloud constraints

As most points in the cloud tend to be closer to the vehicle, an upper boundary of $30m$ from the sensor origin is applied. A lower boundary is also applied of $3m$ to remove the vehicle itself from the point cloud as can be seen in Figure 4.2. These constrains do not discard that much Velodyne data but allows for more of the point cloud map to be removed when computing biases which reduce the computational complexity.

4.4 Point cloud map

Since the point cloud map contains about 765 million points, a subset of the cloud has to be selected before running the GICP algorithm. The subset is selected based on the pose of Velodyne sensor. This limits the GICP to converge within the likely region and reduces the risk of converging to a local minima. The point cloud is also uniformly down-sampled with a ratio $\eta = 0.1$ to speed up computations for the GICP algorithm.

4.4.1 Cloud constrain

Given the pose, the subset is extracted with a radius of $r = 35m$ and height of $-5m \leq z \leq 10m$. Within this region the GICP will most likely find the global optimum to fit the Velodyne cloud within the point cloud map. The radius is selected with a margin of $5m$ compared to the range in the Velodyne cloud and the height is based on the highest/lowest point possible in the Velodyne cloud.

4.5 Matching data

The Velodyne and trajectory data are captured by different systems which operate at different frequencies. Hence there is a discrepancy between the timestamps in the two datasets. This requires the data to be matched so that the location of the Velodyne scans can be properly derived.

4.5.1 Selecting indices for GICP

As the trajectory data is more dense than the Velodyne data, bias computations using GICP are not possible to make for every index i of the trajectory. A subset of indices $j \in J \subsetneq I$ has to be selected from the trajectory that has corresponding Velodyne measurements. Due to the time discrepancy the measurements will not perfectly align, hence, each Velodyne scan is associated with the closest trajectory measurement that fulfills $t_{trajectory} < t_{velodyne}$. The subset J is then created by choosing trajectory points with associated Velodyne scans spaced $2.5m$ apart. This is done to keep the computation time for the GICP down and results in about 20% of the Velodyne data being utilized.

4.5.2 Shifting input

The time discrepancy may be small, but it has the implication that the trajectory pose associated with a Velodyne scan does not exactly represent the pose of the scan. The trajectory data also contains the velocity of the vehicle which can be used to calculate an approximate pose of the scan. Only the spatial dimensions of the pose are taken into consideration as there are no available measurements of the angular velocity and the angles generally change slower. The computation can then be made as

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \Delta t, \tag{4.6}$$

where $\Delta t = t_{velodyne} - t_{trajectory}$. This approximation effectively means that the bias computed by GICP will be computed for the pose corresponding to the velodyne data, however, it will be associated with the trajectory pose. As the bias is assumed to change at a slow rate and the time discrepancy is rather small, on average $\Delta t \approx 0.01s$, the poses are assumed to have the same bias.

4.6 Generalized ICP

The purpose of performing ICP is to refine the initial alignment of the Velodyne (source) cloud $\mathcal{A}_j^V$ to the optimal alignment in the world (target) cloud map $\mathcal{B}_j^W$. The two clouds are initially aligned using homogeneous transformation matrices. From the initial trajectory pose the position and rotational matrix are extracted and then inserted to one homogeneous transformation each, as

$$\mathbf{T}_{R_j} = \begin{bmatrix} \mathbf{R}(\psi, \theta, \phi)_j & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix}, \tag{4.7}$$

$$\mathbf{T}_{t_j} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{t}_j \\ \mathbf{0} & 1 \end{bmatrix}. \tag{4.8}$$

Using both matrices, the alignment transformation can then be expressed as $\mathbf{T}_j = \mathbf{T}_{t_j}\mathbf{T}_{R_j}$. But to avoid numerical errors in the final transformation generated by

the Generalized ICP algorithm, both clouds are instead transformed to align at the global origin as

$$\mathcal{A}_j^{\hat{W}} = \mathbf{T}_{R_j}\mathcal{A}_j^V, \tag{4.9}$$

$$\mathcal{B}_j^{\hat{W}} = \mathbf{T}_{t_j}^{-1}\mathcal{B}_j^W. \tag{4.10}$$

From here, the covariance is estimated for each point in both clouds with $\epsilon = 0.001$ and $k = 20$ for the surface normal according to eq (3.19). The algorithm then minimizes equation (3.21) until it converges or has reached the maximum number of iterations (20) without converging. The GICP then outputs a homogeneous transformation matrix $\mathbf{T}_{G_j}$, which if the algorithms has converged, transforms the source cloud optimally such that it aligns with the target cloud

$$\mathbf{T}_{G_j}\mathcal{A}_j^{\hat{W}} = \mathcal{B}_j^{\hat{W}}. \tag{4.11}$$

4.6.1 Fitness score

There is no guarantee that the GICP algorithm converges to the global optimum, it might get stuck in a local one or not converge at all. Hence, there has to be some quality measurement of each output. GICP outputs a vector containing distances between the points in the source cloud to the closest point in the target cloud. This distance vector can be used for computing the sum of squared distances as a quality metric, this is called fitness score. In order to make the fitness score more dynamic, a maximum allowed distance δ for the distances to be included in the computations, is set. The fitness score of a transformation can then be expresses as a function

$$f(\delta, \mathbf{d}) = \frac{1}{n}\sum_n d_n^2, \quad d_n \in \mathbf{d} < \delta. \tag{4.12}$$

Generally speaking, lower fitness scores correlates with a better alignment of the point clouds, however this only applies to point clouds that highly correlate with each other. If the point clouds contain a lot of noise the fitness score has a tougher time describing the quality of the alignment.

4.6.2 Fitness score filter

As stated section 4.6.1, there is no guarantee that the computed transformation matrix perfectly aligns the point clouds. Bad initial alignment, noisy Velodyne data or a lack of features in the point clouds are all reasons the algorithm might not converge optimally. In order to solve this problem the output has to be filtered to remove bad transformations. This is done by first calculating a mean fitness score

$$f_j = \frac{1}{n}\sum_n f(\delta_n, \mathbf{d}_j)$$

where $\delta_n \in \{0.01, 0.02, \ldots, 10\}$. Properly converged transformations tend to have a lower fitness score regardless of maximum allowed distance, hence, computing the

mean of the filter should be more robust. All mean fitness scores that satisfies the condition

$$f_j < \varepsilon \tag{4.13}$$

are included in the set $k \in K \subsetneq J$. In order to calculate a suitable threshold ε, 200 transformations spread out evenly on the trajectory are annotated as $+$, meaning a transformation that properly aligns the clouds, as -, meaning a transformation failing to align the clouds, or as *0*, meaning a transformation almost aligns the cloud properly, but not quite. These annotations are used to create two sets of mean fitness scores corresponding to the annotated transformations, $\mathbf{f}^+$ and $\mathbf{f}^-$. A simple cost function is then used to compute the threshold ε as

$$\begin{aligned} \varepsilon = \underset{t}{argmax} \quad & \alpha \sum_n a_n + \beta \sum_m b_m \\ a_n = & \begin{cases} 1, & \text{if } f_n^+ < t \\ 0, & \text{otherwise} \end{cases} \\ b_m = & \begin{cases} 1, & \text{if } f_m^- < t \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{4.14}$$

where $\alpha = -2$ and $\beta = 1$. This is in order to penalize inclusion of bad transformations more than to incentivize inclusion of good ones.

4.7 Bias Estimation

The last part of the algorithm is to update the trajectory poses using the result from GICP and then calculate the bias for the full trajectory using RTS-smoothing.

4.7.1 Pose correction

The transformation matrix that the GICP-algorithm calculates transforms the rotated Velodyne point cloud to the translated point cloud map, combining equations 4.10 and 4.11 this is expressed as

$$\mathbf{T}_{G_k}\mathbf{T}_{R_k}\mathcal{A}_k^V = \mathbf{T}_{t_k}^{-1}\mathcal{B}_k^W.$$

In order to update the trajectory poses, the desired transformation is the one that transforms the Velodyne point cloud to the point cloud map. By multiplying with $\mathbf{T}_{t_j}^{-1}$ on both sides in this transformation matrix is obtained as

$$\mathbf{T}_{t_k}\mathbf{T}_{G_k}\mathbf{T}_{R_k}\mathcal{A}_k^V = \mathcal{B}_k^W.$$

As the Velodyne data is recorded in the local coordinate system, the updated pose can be descriped as $\mathbf{T}_k = \mathbf{T}_{t_k}\mathbf{T}_{G_k}\mathbf{T}_{R_k}$. This transformation can then be divided into a 3×3 rotation matrix and a 3×1 translation vector. The matlab function **rotm2eul** is used to convert the rotation matrix to euler angles and the space coordinates correspond directly to the translation vector. The updated poses are denoted as $\hat{p}_k^u$

4.7.2 Outlier filter

In addition to the filter described in section 4.6.2, a chi-squared test is performed to remove very unlikely outputs. This can be viewed as a second step in the task of removing bad GICP output. The applanix system outputs uncertainties in every degree of freedom as RMS values, which can be used to create the covariance matrix

$$P_{gnss} = diag(\begin{bmatrix} x_{rms}^2 & y_{rms}^2 & z_{rms}^2 & \psi_{rms}^2 & \theta_{rms}^2 & \phi_{rms}^2 \end{bmatrix})$$

for the trajectory pose estimates. A measurement is considered an outlier if it deviates more than three standard deviations from the trajectory pose uncertainty. To test whether a measurement is an outlier or not, a chi-squared test is made. This is done by calculating the squared mahalanobis distance

$$d^2 = \Delta p_k^T P_{gnss_k} \Delta p_k, \tag{4.15}$$

which is chi-squared distributed, and comparing it to the threshold γ. The threshold γ is determined as the value for which the cumulative chi-distribution takes on the probability 0.997, which corresponds to three standard deviations. Indices k that has a d^2 larger than γ, are discarded from the K set.

4.7.3 RTS-Smoothing

The bias along the trajectory is assumed to be changing at a very slow rate that is uncorrelated with the vehicle motion. This makes a random walk model suitable as the process model of the RTS-smoother.

The state, denoted as $\hat{b}$, and measurement vectors are both made up by the biases in all degrees of freedom of a vehicle pose

$$\hat{b}, y = \begin{bmatrix} b_x & b_y & b_z & b_\phi & b_\theta & b_\psi \end{bmatrix}^T.$$

Since the process model of choice is a random walk model $A = \mathbf{I}_6$. The measurement model $H = \mathbf{I}_6$, as the measurements are direct. The measurements y_k are calculated as $p_k^u - \hat{p_k}$.

The process noise covariance is tuned so that the covariance of the estimates increases at a slow rate as the bias has been assumed to change slowly. The measurement noise covariance is determined as

$$R_k = (f_k - f_{min}) * R_0$$

where f_{min} is the smallest fitness score mean and R_0 a covariance matrix that corresponds with a near perfect direct measurement of the states. The dimensions corresponding to position are rotated by the rotation matrix for p_k in order to represent the uncertainty in longitudinal, lateral and height instead of global position.

The RTS-smoother has to be slightly modified as the measurements are more sparse than the trajectory poses. Usually the filtering part of the RTS-smoother contains

a prediction and an update step, however, as the trajectory only contains sparse measurements indexed by K the update step is only to be done at these indices. This results in the filtering part of the RTS-smoother being modified as follows:

for $i = 1$ **to** N **do**
  $\hat{b}_{i|i-1}$, $P_{i|i-1} \leftarrow predictionStep(\hat{b}_{i|i}, P_{i|i})$
  if $i \in K$ **then**
    $\hat{b}_{i|i}$, $P_{i|i} \leftarrow updateStep(\hat{b}_{i|i-1}, P_{i|i-1}, y_k)$
  else
    $\hat{b}_{i|i}$, $P_{i|i} \leftarrow \hat{b}_{i|i-1}$, $P_{i|i-1}$
  end if
end for

4.7.4 Fusion of trajectory

The final estimation of the trajectory is obtained by fusing the initial trajectory estimate with the bias estimates. The *sensor fusion formula* described in section 3.2.3 can be used for this purpose. By denoting $\hat{x_1} = \hat{p}$, $P_1 = P_{gnss}$, $\hat{x_2} = \hat{p} + \hat{b}$ and $P_2 = P$, equation 3.15 can be rewritten as

$$\hat{p}_i = \hat{p}_i + (P_{gnss_i}^{-1} + P_i^{-1})^{-1} P_i^{-1} \hat{b}_i \tag{4.16}$$

which is used for computing the estimates of the refined trajectory. The estimates can be considered independent as the GICP does not depend on the initial estimate in any other way than as a starting point for alignment.

5
Results

This chapter presents how the implemented algorithm has been evaluated and what results were obtained from said evaluation.

5.1 Fitness score and outlier filters

The fitness removal filter and outlier filter are evaluated as a combined filter, as they both perform the same task of removing bad GICP outputs. They are evaluated by comparing the annotated labels to the filter classification. As described in section 4.6.2, the filter is tuned with a set of annotated data, further denoted the validation set. The filter performance is then determined by the classification accuracy on a test set, which has been annotated in the same way as the validation set, however for different trajectory poses.

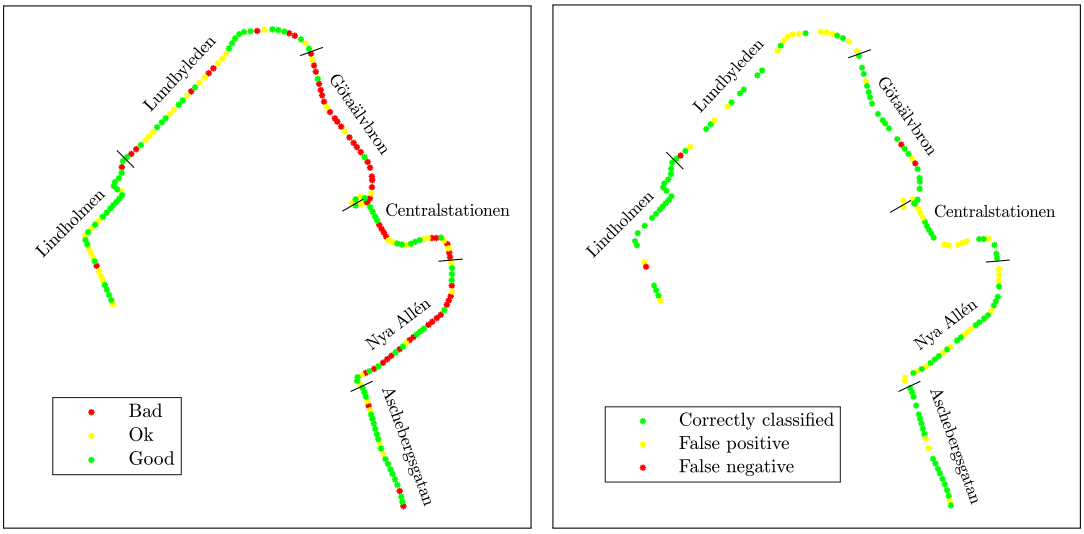
5.1.1 Validation data

The evaluation of the validation set shows the result of the filter tuning. As the filter either classifies the ICP output as *+ (Good)* or *- (Bad)*, the data annotated as *0 (Ok)* will fall into either of those categories. The filter classification of the validation set is shown in table 5.1.

Table 5.1: Filter classification of the annotated validation set.

Validation set			
Filter classification	**Label**		
	Good	*Bad*	*Ok*
Good	42	4	15
Bad	45	57	32
Total	87	61	47

The distribution of the annotated labels along the trajectory is shown in Figure 5.1a. The distribution of filter classification performance along the trajectory is shown in Figure 5.1b. As the data annotated as *0* is classified as either *+* or -, these point are not regarded as correctly classified for either case, neither can they be false positives or false negatives.



(a) Validation set annotations **(b)** Classification results of validation set

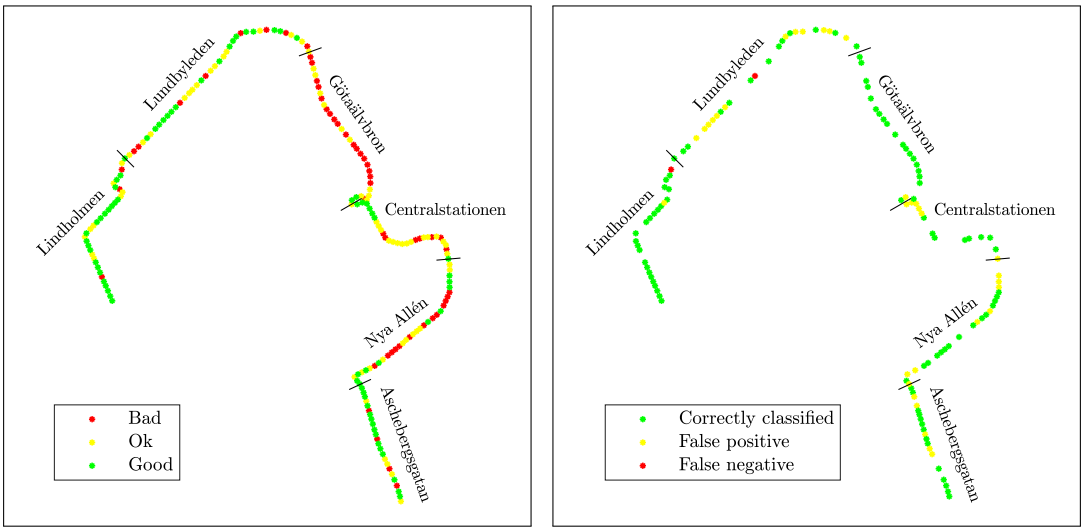
Figure 5.1: Labels of the validation set along the trajectory **(a)** and filter classification performance along the trajectory **(b)**.

5.1.2 Test data

The evaluation of the test set shows the performance of the filter on unseen data. The results are presented in the same manner as in section 5.1.1.

Table 5.2: Filter classification of the annotated test set.

Test set			
Filter classification	**Label**		
	Good	*Bad*	*Ok*
Good	41	2	16
Bad	33	52	50
Total	74	54	66



(a) Test set annotations **(b)** Classification results of test set

Figure 5.2: Labels of the test set along the trajectory **(a)** and filter classification performance along the trajectory **(b)**.

5.1.3 Complete trajectory

Figure 5.3 shows how the poses along the entire trajectory are classified by the combined filter.

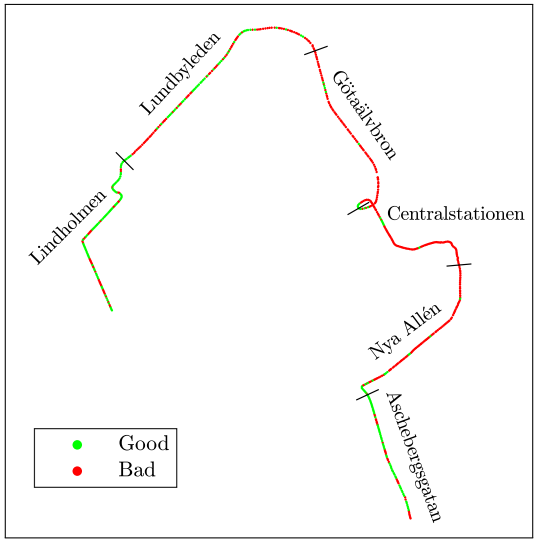


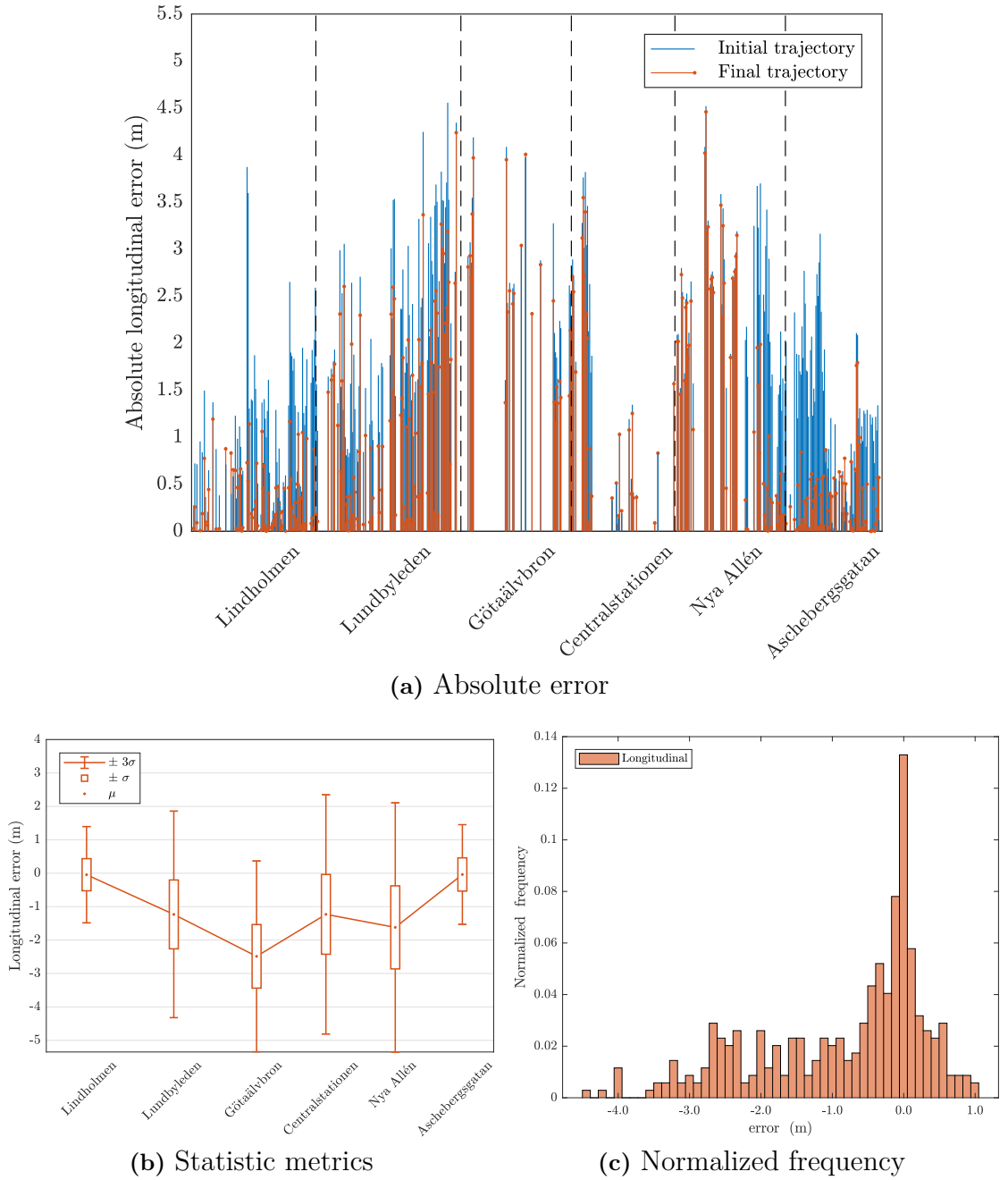
Figure 5.3: Filter classification along the entire trajectory.

5.2 Trajectory accuracy

The trajectory accuracy is evaluated by measuring the error between the refined trajectory and a set of ground truth poses along the trajectory. The ground truth is generated by first performing GICP on a new subset of poses not used in the smoothing process. This subset is selected by picking the trajectory poses with corresponding Velodyne data that lies the furthest from the poses used for smoothing. The new GICP output is labeled as either good or bad. The labeling is done by visually inspecting the alignment of source and target cloud where only the correct ones are labeled as good. All the good transformations are then used as ground truth and all false transformations are discarded and not included in the evaluation set. The evaluation set is split into six parts: *Lindholmen, Lundbyleden, Götaälvbron, Centralstationen, Nya Allén and Aschebergsgatan* according to the text labels in Figure (5.1). The results are then presented in Longitudinal, Lateral, Altitude, Heading, Pitch and Roll error.

5.2.1 Longitudinal error

Subfigure **(a)** shows the absolute longitudinal error of the initial and the final trajectory. No lines implies unavailable ground truth data. Subfigure **(b)** shows the statistic metrics, mean value μ, standard deviation $\pm\sigma$ and $\pm 3\sigma$ of the longitudinal error. The metrics are calculated for each part of the trajectory. Subfigure **(c)** shows the normalized frequency of the longitudinal error of all evaluation points.



(a) Absolute error

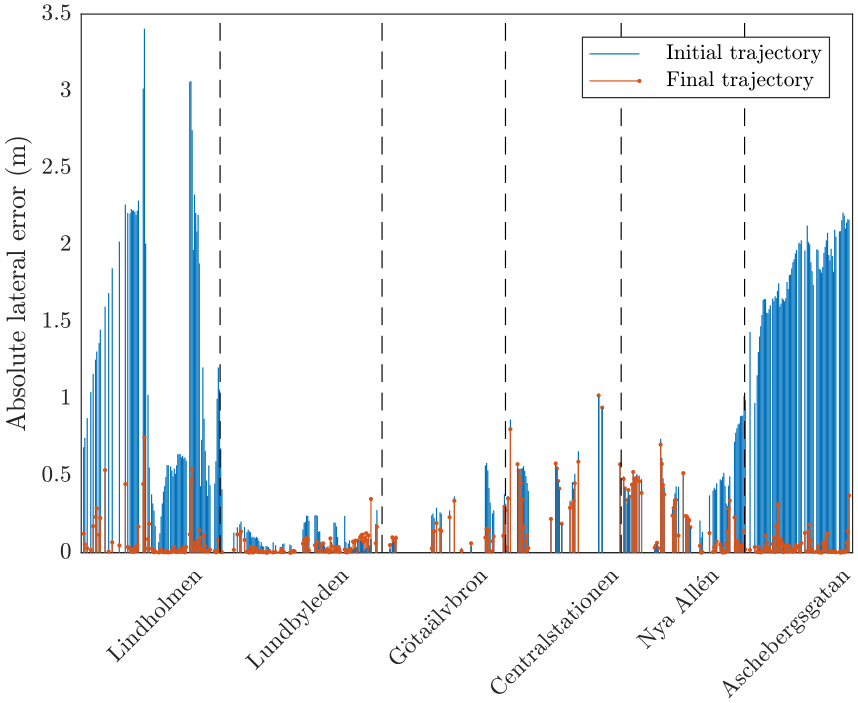
(b) Statistic metrics

(c) Normalized frequency

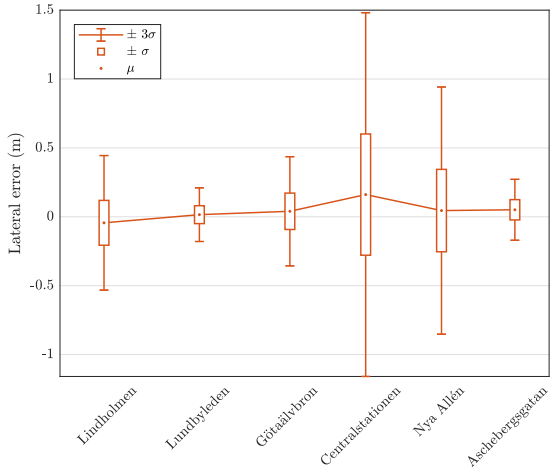
Figure 5.4: Absolute longitudinal error.

5.2.2 Lateral error

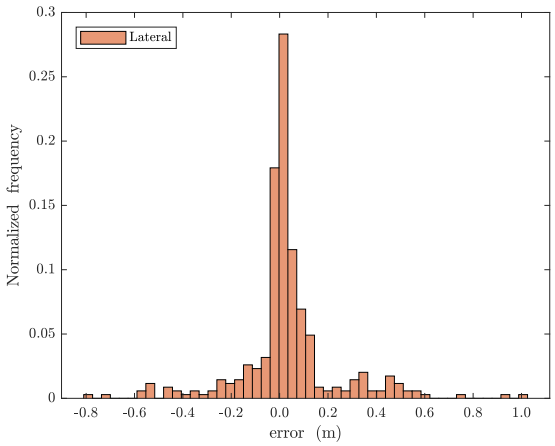
Subfigure **(a)** shows the absolute lateral error of the initial and the final trajectory. No lines implies unavailable ground truth data. Subfigure **(b)** shows the statistic metrics, mean value μ, standard deviation $\pm\sigma$ and $\pm 3\sigma$ of the lateral error. The metrics are calculated for each part of the trajectory. Subfigure **(c)** shows the normalized frequency of the lateral error of all evaluation points.



(a) Absolute error



(b) Statistic metrics

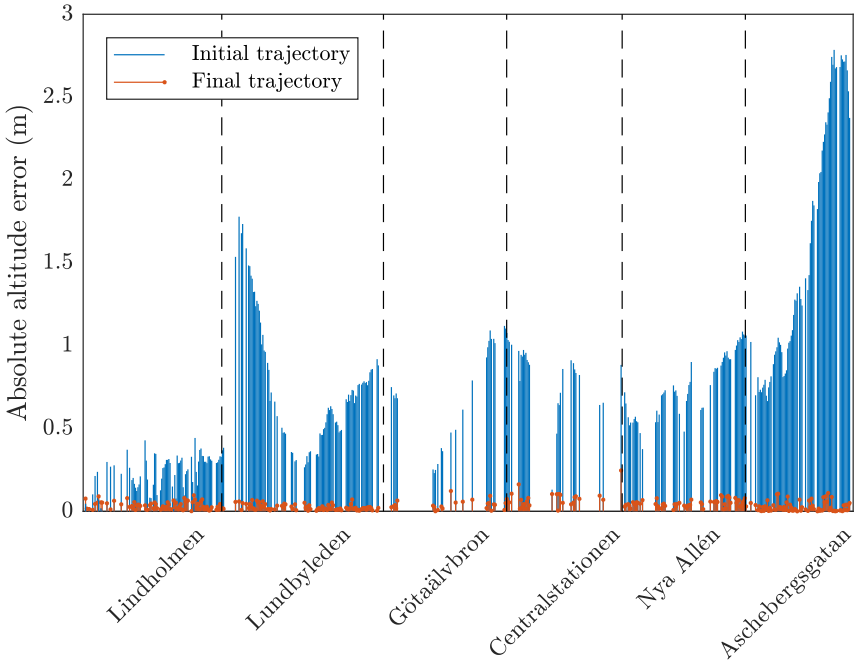


(c) Normalized Frequency

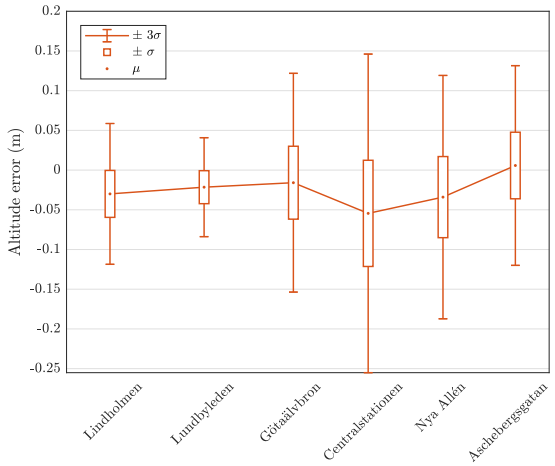
Figure 5.5: Absolute lateral error.

5.2.3 Altitude error

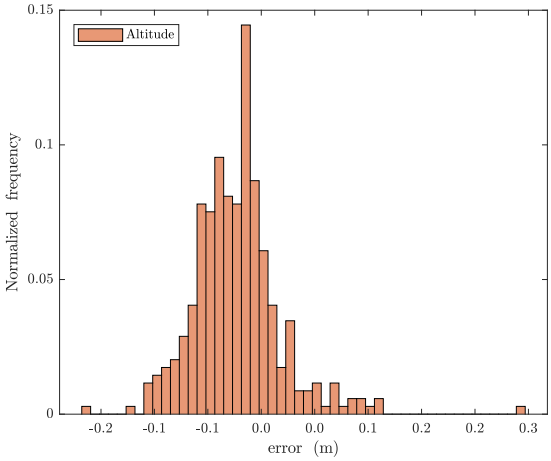
Subfigure **(a)** shows the absolute altitude error of the initial and the final trajectory. No lines implies unavailable ground truth data. Subfigure **(b)** shows the statistic metrics, mean value μ, standard deviation $\pm\sigma$ and $\pm 3\sigma$ of the altitude error. The metrics are calculated for each part of the trajectory. Subfigure **(c)** shows the normalized frequency of the altitude error of all evaluation points.



(a) Absolute error



(b) Statistic metrics

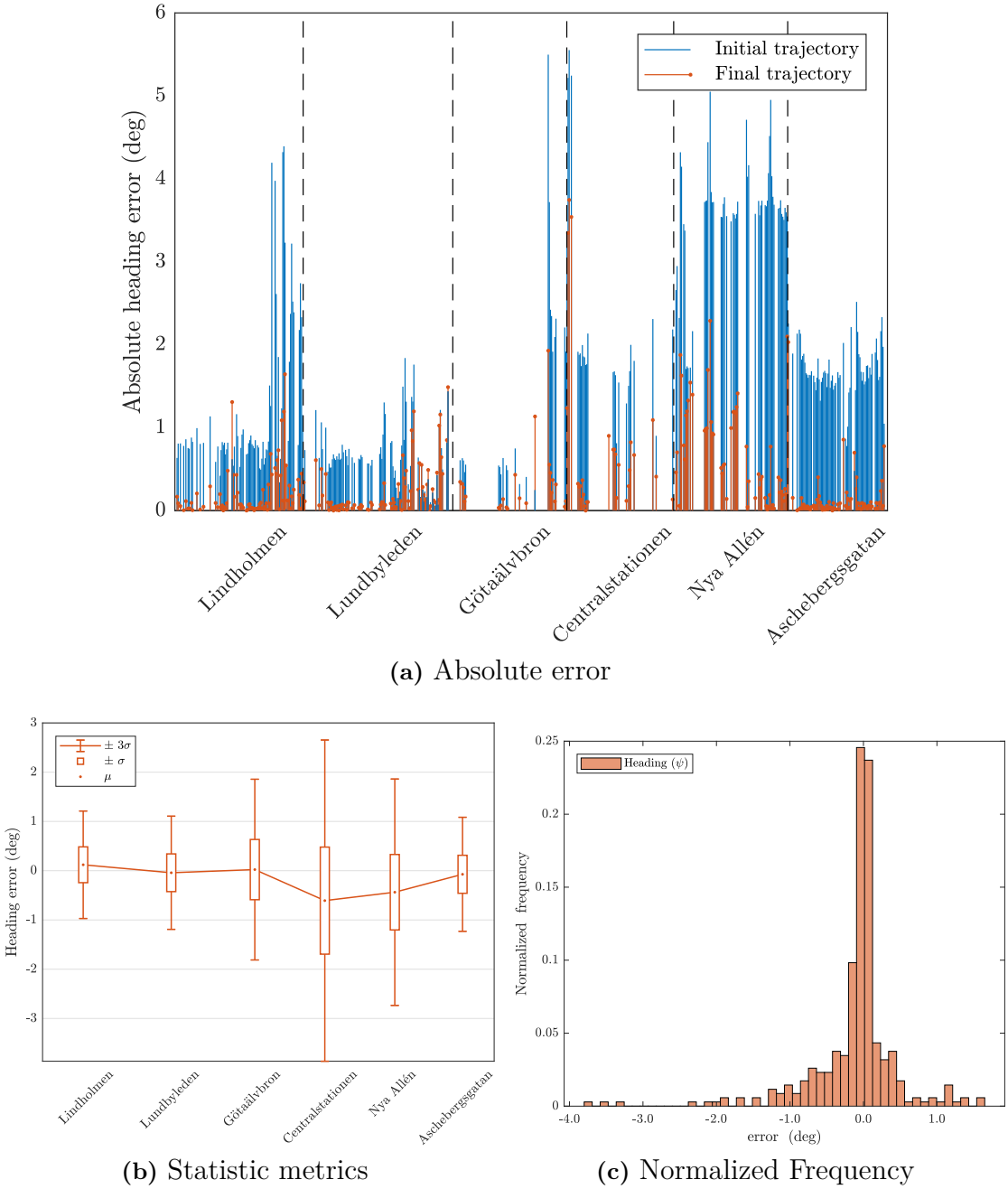


(c) Normalized Frequency

Figure 5.6: Absolute altitude error.

5.2.4 Heading error

Subfigure **(a)** shows the absolute heading error of the initial and the final trajectory. No lines implies unavailable ground truth data. Subfigure **(b)** shows the statistic metrics, mean value μ, standard deviation $\pm\sigma$ and $\pm 3\sigma$ of the heading error. The metrics are calculated for each part of the trajectory. Subfigure **(c)** shows the normalized frequency of the heading error of all evaluation points.



(a) Absolute error

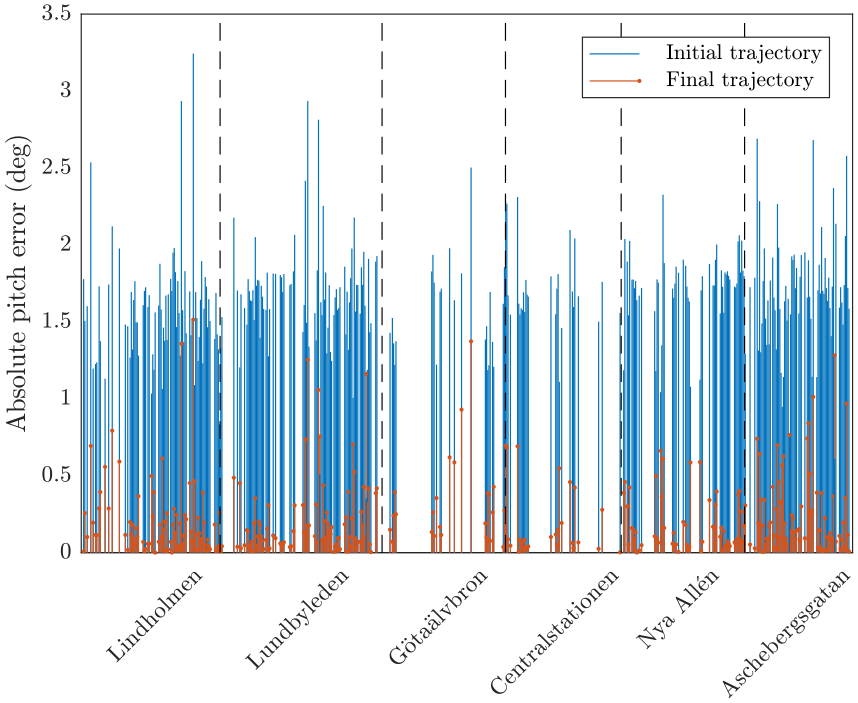
(b) Statistic metrics

(c) Normalized Frequency

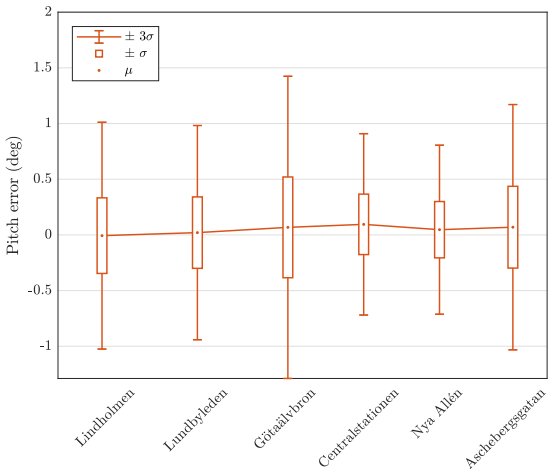
Figure 5.7: Absolute heading error.

5.2.5 Pitch error

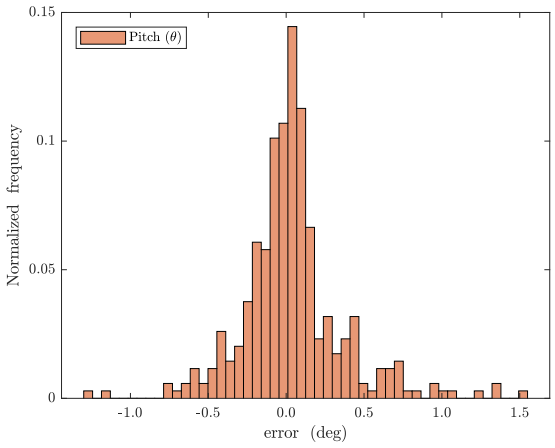
Subfigure **(a)** shows the absolute pitch error of the initial and the final trajectory. No lines implies unavailable ground truth data. Subfigure **(b)** shows the statistic metrics, mean value μ, standard deviation $\pm\sigma$ and $\pm 3\sigma$ of the pitch error. The metrics are calculated for each part of the trajectory. Subfigure **(c)** shows the normalized frequency of the pitch error of all evaluation points.



(a) Absolute error



(b) Statistic metrics

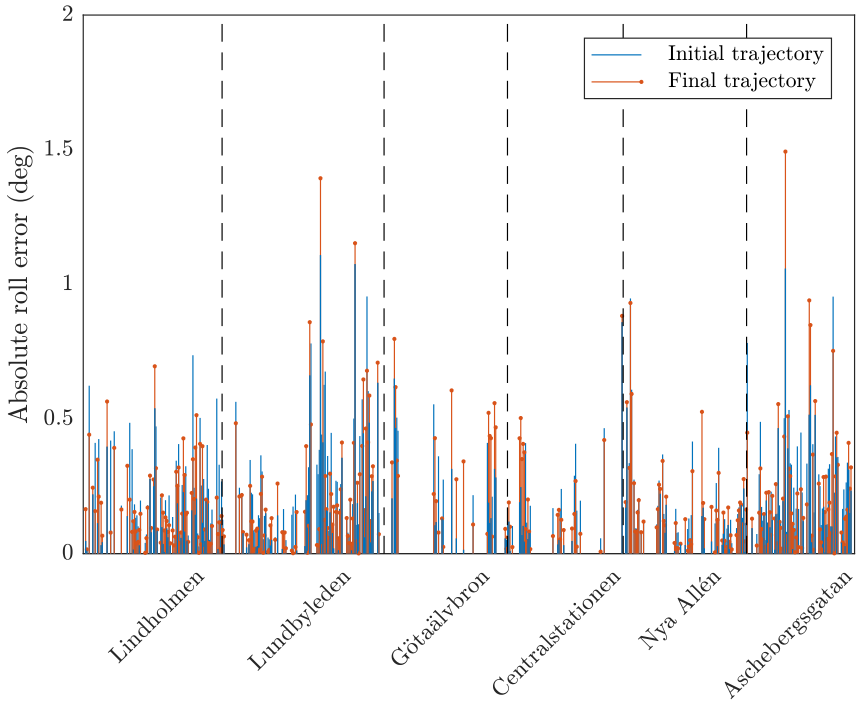


(c) Normalized frequency

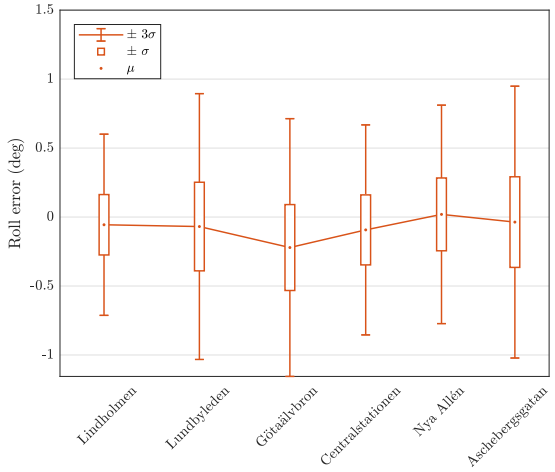
Figure 5.8: Absolute pitch error.

5.2.6 Roll error

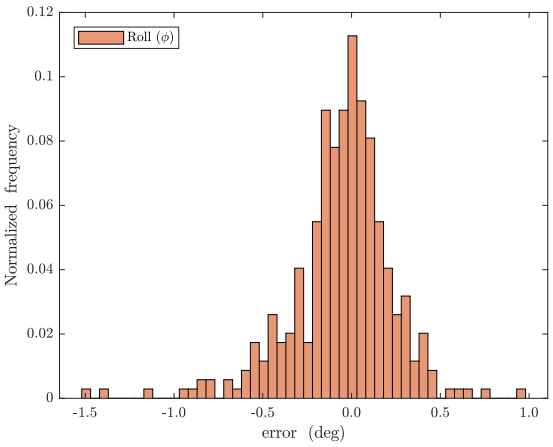
Subfigure **(a)** shows the absolute roll error of the initial and the final trajectory. No lines implies unavailable ground truth data. Subfigure **(b)** shows the statistic metrics, mean value μ, standard deviation $\pm\sigma$ and $\pm 3\sigma$ of the roll error. The metrics are calculated for each part of the trajectory. Subfigure **(c)** shows the normalized frequency of the roll error of all evaluation points.



(a) Absolute error



(b) Statistic metrics



(c) Normalized frequency

Figure 5.9: Absolute roll error.

5.3 Visual illustration

To show the visual difference between the initial and smoothed pose two synthetic images are generated from the point clouds. The Velodyne points are color coded by the distance to the point starting from blue ($3m$) to red ($70m$). The difference is evident at the flag and light poles where one can see how the points line up in the smoothed trajectory. It is also evident at the building to the right, where the red segment aligns with the rectangular shaped windows.

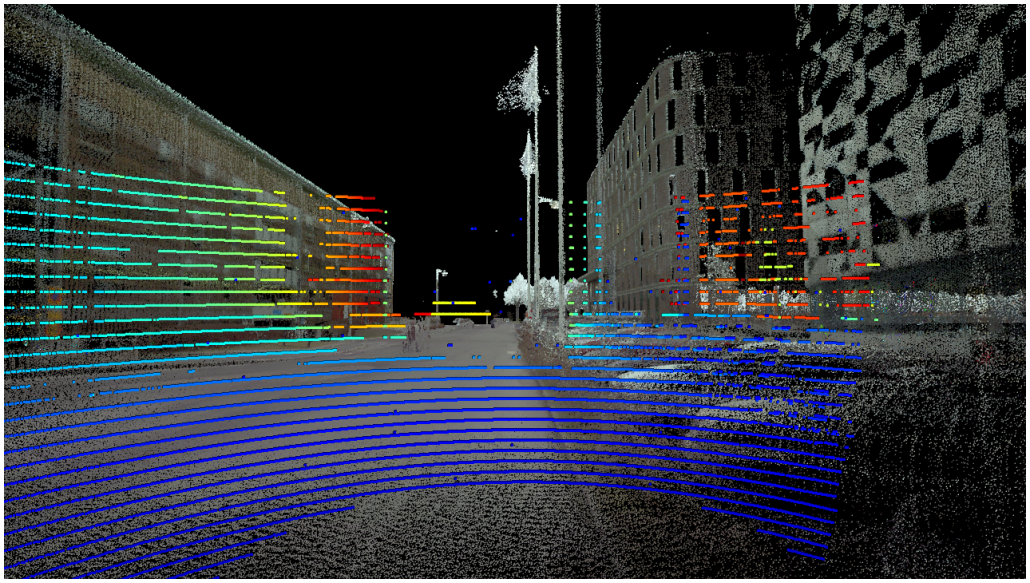


Figure 5.10: Projected point cloud based on initial trajectory pose.

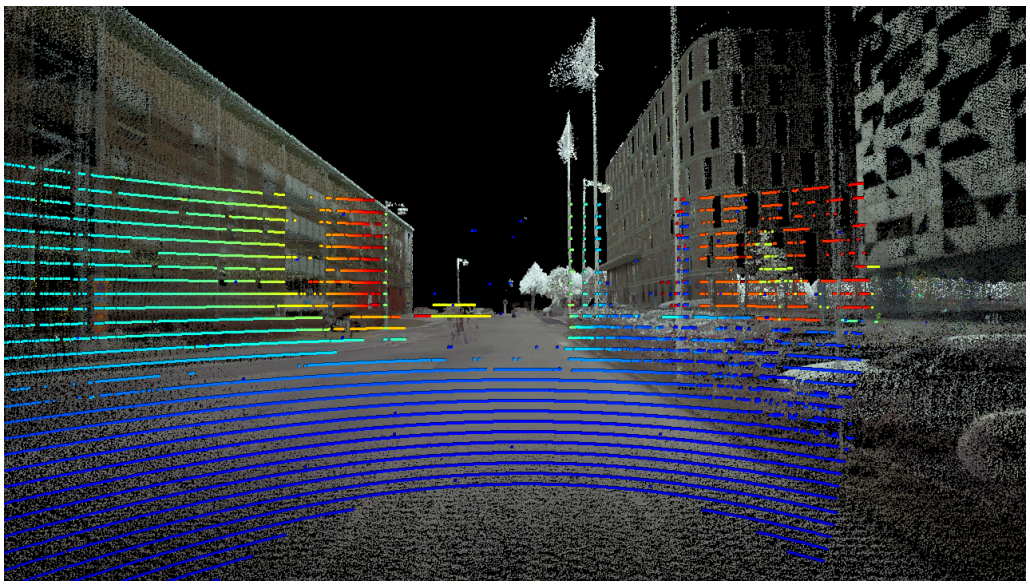


Figure 5.11: Projected point cloud based on smoothed trajectory pose.

6
Discussion

This chapter includes discussion of the result, drawn conclusions and suggested future work.

6.1 GICP performance

Overall the GICP performs well and is able to find the transformation that aligns the Velodyne cloud in the map for multiple scenarios. When the correct transformation is found, the alignment is close to a perfect fit. However, there are situations when the GICP fails to find the correct transformation and get stuck in a local minimum instead. When the test and validation sets in Figures 5.1a and 5.2a were annotated, primarily four scenarios were identified as likely causes for the incorrect transformations.

1. **Noisy velodyne data** is the most common cause of incorrect transformations. The noise consists mainly of surrounding traffic captured in the Velodyne cloud. Typical objects are truck trailers, buses and trams which often stands for a large percentage of the Velodyne points. Without any possibility of excluding those points the transformation often becomes incorrect.

2. **Few longitudinal features** is primarily an issue on *Götaälvbron*. The GICP often aligns the cloud correct laterally but not longitudinally. This is most likely due to a lack of tall buildings/structures/edges/corners on the sides of the bridge. With more surrounding structures like that, the longitudinal positioning becomes better.

3. **Symmetric environments** is another issue discovered as the GICP struggles to find the correct fit when there are several potential fits. The resulting alignment in these situations often ends up somewhere in the middle between the potential fits.This is primarily present along *Nya Allén* where the trees are placed in a symmetric pattern.

4. **Turns** were found to be hard to align as well. Due to the Linear compensation of the Velodyne point cloud, the vehicle turn rate is not considered resulting in an inaccurate point cloud representation of the surrounding. and therefore becomes an inaccurate representation of the surroundings.

6.2 Filtering of GICP output

As can be seen in Figures 5.1b and 5.2b the fitness score filter combined with the outlier filter is quite good at correctly classifying the data. By studying tables 5.1 and 5.2 we see that this is mostly due to almost all poses being classified as *bad*. This is good as this is the intended behaviour of the filters, to remove the bad output. This comes at a relative high price however, in both the validation set and test set approximately half of the *good* output are false positives. We believe this is mostly a consequence of noisy Velodyne data, however, noise in the point cloud map and over-confident estimates from the Applanix system might also contribute to good data being removed. Regarding the output labeled as *ok* this is mostly classified as bad, which is quite good as even though the output has not quite converged to the true pose. Hence, we believe letting this output through the filter is certainly better than letting through bad output, however, given a perfect filter it would be desireable to also remove these.

It is our firm belief that excluding good data is better than including bad data, however if the ultimate goal is to compute ground truth data either a higher portion of the output needs to be good, or a filter that is better at distinguishing bad from good output is needed. A third option is to entirely skip the fitness score filter, only removing outliers and then determine the covariance of the remaining output. This solution would, however, require a robust method to determine the covariance of the output. The current approach of scaling a minimum covariance with the fitness score does not solve this problem as some of the good output would be considered unnecessary uncertain while also setting a quite small covariance for some bad output. If we were to come up with a better way of determining covariance, removing the fitness score filter would certainly be an interesting approach as it keeps more information for the RTS-smoother.

6.3 Bias estimation and fusion

From the absolute error plots in section 5.2 it is clear that the refined trajectory is more accurate than the initial one for all metrics, except maybe roll. It is worth to keep in mind that the initial trajectory estimates are calculated online and for a fairer comparison a post processed trajectory might have been preferable. Post processing, however, requires licensed Applanix software that was not available during the course of this thesis.

In broad terms the performance has improved the most in the *Lindholmen* and *Aschebergsgatan* segments, which is especially clear when studying Figures 5.5a and 5.6a. This is probably due to the many quite tall buildings situated close to the road, reducing the quality of the GNSS signal. Overall these segments also see the most GICP output being used, as can be seen in Figure 5.3. This is positive as it proves the concept of bias estimation works as intended. Further, there is a quite clear correlation between the absence of GICP output and less accurate performance.

This can be seen in the statistic metrics plot where the segments *Lundbyleden, Götaälvbron, Centralstationen* and *Nya Allén* usually have a mean deviating further from 0 and a larger standard deviation. The fusion of the trajectory brings with it the fact that a lot of the bias will not be used when the estimate is uncertain or when the initial estimate is very certain. Hence the inaccuracy in these segments probably have two sides to them. Firstly there is a lack of measurements of the bias, leading to the bias smoother outputting high covariance along with the estimate. The other side is that the initial estimates probably are over-confident in a lot of these segments, leading to a lot of the initial error being retained in the refined estimates.

Besides the accuracy of the algorithm there are a couple of other interesting conclusions to be drawn from the results. Starting with the altitude it look like the distribution in Figure 5.6c is slightly biased. This could imply that the reference height used when correcting the height is a little of and should be measured again to either confirm or dismiss this suspicion. Looking at the absolute error of the pitch for the initial estimate it is almost constant, which could imply a constant bias for this metric as well. By plotting a histogram of the distribution, see Figure 6.1, it becomes clear that there is a bias of approximately 1.75 *deg*. This could be accounted for in the preprocessing of the trajectory estimates in order to improve performance, as without the bias the pitch error would be very low from the start.

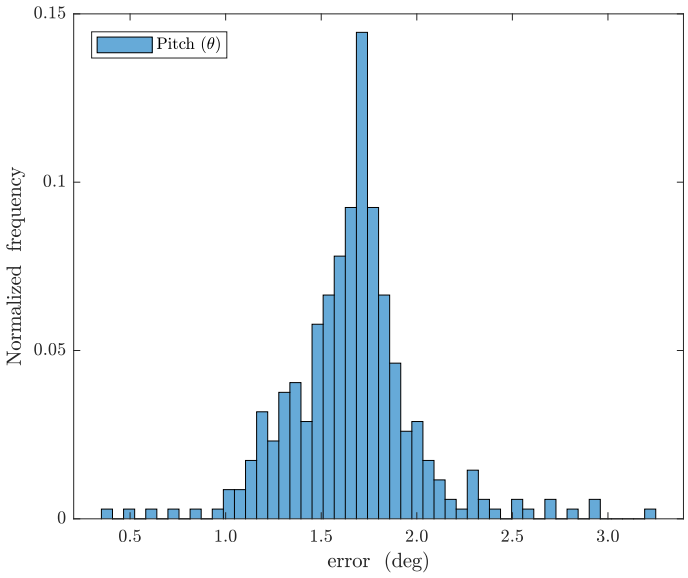


Figure 6.1: Normalized frequency of the pitch error for the Applanix estimate.

To conclude the discussion of the bias estimation it can be said that the accuracy of the refined trajectory is very high when good measurements of the bias exist, however it falls short in the abscence of measurements, mostly due to the inaccuracies of the Applanix estimates.

6.4 Future work

Based on the discussion held previously in this chapter, there are a couple of promising improvements we believe would make the algorithm more robust and accurate.

6.4.1 Removing noise from Velodyne data

The algorithm proposed in this thesis has shown to be very sensitive to noise in the Velodyne data. If the noise could be removed, a better accuracy would most likely be achieved.

6.4.2 Non-linear ego motion compensation of Velodyne point cloud

The assumption of linear motion when ego-motion compensating the Velodyne data has been shown to hold poorly when the heading of the car changes rapidly. In order to solve this we propose to shift the Velodyne data non-linearly instead. As all of the data regarding how the vehicle has moved is available this should be quite easy to compute.

6.4.3 Iterative refinement of trajectory

By itertatively refining the trajectory two large benefits are gained. First the convergence rate has been noted to increase when computing GICP for the refined trajectory, which is done when computing ground truth. This is likely due to the GICP being less prone to get stuck in local minimas when the initial pose is closer to the true pose. The second benefit of refining the trajectory iteratively is that convergence times of GICP drop by quite much when the initial pose is better, leading to the possibility of using more of the Velodyne data in the same amount of time.

6.4.4 Removing cloud constrains

The current removal of point cloud data by imposing constraints on the point cloud in retrospect seem unnecessary. The only reason this was done in the first place was to reduce the computational time so that the calculations could be completed within a reasonable time frame during the development. However, as stated in the scope of the Thesis, the computational complexity of the solution is not a limiting factor. These constraints should therefore be removed in order to include as much data as possible for the ICP-calculations.

It is hard to tell how much this would improve the results of the algorithm, as most of the Velodyne data is included in the majority of the poses. In some cases, mainly where the features are sparse, using all of the Velodyne data should definitely yield better results than when removing data.

Bibliography

[1] Safer. (2017) About us. [Online]. Available: https://www.saferresearch.com/about#block-aboutus

[2] L. Hammarstrand. (2016) Copplar - campusshuttle cooperative perception and planning platform. [Online]. Available: https://www.saferresearch.com/projects/copplar-campusshuttle-cooperative-perception-and-planning-platform

[3] R. W. Wolcott and R. M. Eustice, "Visual localization within LIDAR maps for automated urban driving," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Chicago, IL, USA, September 2014, pp. 176–183.

[4] G. Pascoe, W. Maddern, and P. Newman, "Direct visual localisation and calibration for road vehicles in changing city environments," in *IEEE International Conference on Computer Vision: Workshop on Computer Vision for Road Scene Understanding and Autonomous Driving*, Santiago, Chile, December 2015.

[5] R. W. Wolcott and R. M. Eustice, "Visual localization within lidar maps for automated urban driving," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2014, pp. 176–183.

[6] T. Caselitz, B. Steder, M. Ruhnke, and W. Burgard, "Monocular camera localization in 3d lidar maps," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016. [Online]. Available: http://ais.informatik.uni-freiburg.de/publications/papers/caselitz16iros.pdf

[7] E. Hägg and Y. Ning, "Map representation and lidar-based vehicle localization," Master's thesis, Chalmers University of Technology, 2016.

[8] L. Wang, Y. Zhang, and J. Wang, "Map-based localization method for autonomous vehicles using 3d-lidar **this work is supported in part by the national natural science foundation of china under grant no. 61473209." *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 276 – 281, 2017, 20th IFAC World Congress. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405896317300630

[9] C. Brenner, "Vehicle localization using landmarks obtained by a lidar mobile mapping system," *Proceedings of the Photogrammetric Computer Vision and Image Analysis*, vol. 38, pp. 139–144, 2010.

[10] Lantmäteriet. (2018) Sweref99 reference system. [Online]. Available: https://www.lantmateriet.se/en/Maps-and-geographic-information/GPS-and-geodetic-surveys/Reference-systems/Three-dimensional-systems/SWEREF-99/

[11] Applanix. (2015) Specification. [Online]. Available: https://www.applanix.com/pdf/specs/POSLV_Specifications_dec_2015.pdf
[12] V. Lidar. (2018) Specification. [Online]. Available: http://velodynelidar.com/hdl-32e.html
[13] S. Särkkä, *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013.
[14] F. Gustafsson, *Statistical Sensor Fusion*. Studentlitteratur AB, 2010.
[15] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 2, pp. 239–256, Feb. 1992. [Online]. Available: http://dx.doi.org/10.1109/34.121791
[16] A. Segal, D. Haehnel, and S. Thrun, "Generalized-icp," in *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.
[17] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.