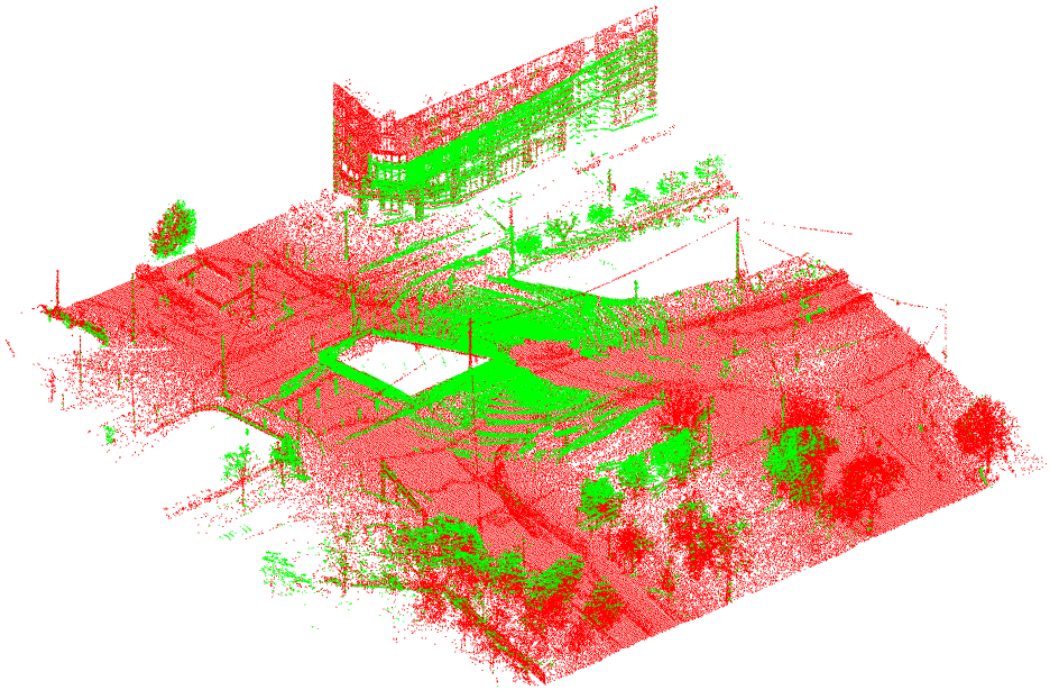
CHALMERS
UNIVERSITY OF TECHNOLOGY



Localization in a dense point cloud for ground truth estimation

Combination of LiDAR and image data for precise and accurate localization

Master's thesis in Systems, Control and Mechatronics

GUÐMUNDUR HJALMAR EGILSSON
ARNO ROLAND GUSTAFSSON

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019 - EENX30

Localization in a dense point cloud for ground truth estimation

Combination of LiDAR and image data for precise and accurate localization

GUÐMUNDUR HJALMAR EGILSSON
ARNO ROLAND GUSTAFSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

Localization in a dense point cloud for ground truth estimation
Combination of LiDAR and image data for precise and accurate localization
GUÐMUNDUR HJALMAR EGILSSON, ARNO ROLAND GUSTAFSSON

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

Master's Thesis 2019
Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A processed LiDAR point cloud from a test vehicle after it has been been passed through the algorithm (green) localized accurately in a dense reference point cloud (red).

Gothenburg, Sweden 2019

Localization in a dense point cloud for ground truth estimation
Combination of LiDAR and image data for precise and accurate localization
GUÐMUNDUR HJALMAR EGILSSON, ARNO ROLAND GUSTAFSSON
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The field of autonomous driving is a rapidly developing field which presents new challenges to vehicle control and its interaction with the current environment. Machine learning is an integral aspect of this field of research. In order to semi-automatically annotate data for machine learning, accurate localization is required. In urban environments a standard GPS system suffers a loss of accuracy due to signal propagation and rerouting of its communication signals to and from the GPS satellites. Improved localization can be obtained using on-board equipment such as LiDAR scanners and cameras to improve GPS measurements.

The main aim of this thesis is to accurately and robustly localize a vehicle in a dense 3D point cloud, by fusing on-board GPS data with LiDAR and camera data. The thesis attempts this by splitting this task into two mains sections, namely the preprocessing of the local LiDAR clouds from the test vehicle and then the alignment of these local clouds to a dense reference cloud.

The processing of point clouds is done by EGO-motion rectification, removal of ghost objects based on object detection in images and the merging of several local point clouds to create a denser representation of the current environment. These point clouds are then localized within a dense reference point cloud map, ranging between Chalmers Johanneberg and Lindholmen campuses. The localization is performed by the Generalized Iterative Closest Point (GICP) algorithm, with initial location estimates based on the GPS measurements. These results from the GICP alignment are classified according to their alignment rating and modelled as independent localization measurements. The GPS measurements are updated by smoothing the bias between the GPS measurements and the GICP outputs. Bad GICP alignments are then recomputed using a new initial location, based on the updated GPS trajectory. A final trajectory is then obtained by smoothing the bias between the combination of the two GICP passes and the GPS trajectory. This final trajectory is then evaluated against a ground truth estimate, constructed of manually verified alignments.

This method provides localization results which behave robustly in the presence of ghost objects and altered structural surroundings. In conditions similar to when the dense reference cloud was created, the algorithm is able to localize 74% of the final trajectory within 10cm in longitude, latitude and altitude at less than 1° angular deviation (*Good*). Additionally, 23% of instances fall within 50cm in position at less than 3° in angular deviation (*Ok*), leaving around 3% localized with an accuracy exceeding these boundaries (*Bad*). For a second test run at night in light rain, *Good* accuracy decreased approximately 7% for the first category whilst *Bad* localization increased by 3.5% with the remaining instances classified as *Ok*.

Keywords: Localization, Computer Vision, Machine Learning, Filtering, Thesis.

Acknowledgements

We would like to thank our supervisors Lars and Fredrik for their continuous support and guidance throughout the work. We also want to thank Arpit Karsolia and Christian Berger at Revere for their help and support with the data collection needed for the thesis. Further thanks go out to Måns Larsson for the semantic segmentation of the images with his neural network. Lastly, we want to thank Gunnar Bolmwall and Måns Östman for their assistance in the start of the project.

Guðmundur Hjalmar Egilsson, Arno Roland Gustafsson, Gothenburg, June 2019

Contents

List of Figures

List of Tables

Acronyms

FOV Field of View.
fS5 Fitness score output of the GICP, evaluated for points within a 5m range.

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

ICP Iterative Closest Point.
IMU Inertial measurement unit.

LiDAR Light detection and ranging scanner.

RANSAC Random Sample Consensus.
RTK Real-time Kinematic.
RTS Rauch-Tung-Striebel smoother.

1
Introduction

This chapter introduces the topic and the background of this thesis and includes the research questions. The chapter also gives an overview of the thesis scope and limitations and briefly discusses previous work.

1.1 Vehicle localization

Autonomous driving is currently a fast-growing field of innovation with various companies and academics in contest towards a vision of fully autonomous travel. The concept of fully autonomous vehicles consists of a network of interconnected vehicles, which navigate themselves without active human input or control. The advantages expected to be obtained include the number of road traffic accidents decreasing by eliminating the chance for human error, a lowering of vehicle emissions/energy usage due to the interaction between vehicles by optimizing vehicle routes and velocities, as well as freeing up time which an occupant would have to spend actively driving a vehicle.
Handing over the control of a complex system such as the traffic network to an artificial intelligence system requires a fault-free, accurate and robust system since human lives can be at stake. For autonomous driving, it is therefore crucial to be able to localize a self-navigating vehicle accurately and robustly in its current environment at all times. This precise knowledge of an autonomous vehicle's position is required for the system to interact correctly and safely with its surroundings, which involves constant decision making such as path planning, vehicle control, as well as anticipation and response to actions from other vehicles or pedestrians.
A vehicle's standard Global Navigation Satellite System (GNSS) measurements alone generally do not have sufficient accuracy and are susceptible to errors in urban environments due to satellite signal obstruction and multipath propagation. In order to obtain a more robust and accurate localization of a provided test vehicle, it is possible to use miscellaneous information sources such as a Real-time Kinematic (RTK) Global Positioning System (GPS) and camera images, LiDAR and GNSS coordinates from the vehicle itself. This information can be merged together using sensor fusion techniques and used in combination with a dense point cloud reference map to attempt to obtain an accurate and robust localization.
However, sensors only read information of the current state of their surroundings and are not, by themselves, able to distinguish varying/inconstant objects from stationary and more permanent objects. Difficulty therefore arises due to the fact that the perceived urban environment is constantly changing. This is due to moving

vehicles, people and animals present at a specific time instance. This can create *ghost objects* in the data, which are unreliable to be used for localization due to their non-stationary positioning.
In addition to this, seasonal changes have a substantial impact on the perceived surroundings. Not only is the light intensity a constantly varying factor, but also the shape and colour of trees or buildings due to time of year, temperature, or snowfall. All these factors increase the difficulty of localization based on the perceived environment.
This project will include the use of the aforementioned sensors, such as cameras and a GNSS unit along with an on-board LiDAR mounted on a test vehicle, to attempt to localize the vehicle. The goal is to achieve a more robust and consistently accurate localization within this constantly varying environment, given a previously obtained dense point cloud. Data will be collected with the vehicle under different conditions and processed in a way that should minimize the effects of ghost objects and environmental conditions.

1.2 Aim

The purpose of this project is to accurately and robustly localize a vehicle within a dense point cloud. An algorithm should be developed that receives various input data from LiDAR, a GNSS system and a camera and combines this data to output a more accurate position estimate of the vehicle. The computed best estimate of the algorithm will be aligned within the dense point cloud, and accuracy of the estimated position should be computed by a chosen metric to assess the alignment.

Research questions the thesis aims to answer:

- Is it possible to improve previous work done on the system so that the localization performs well for all parts of a specified route? This can be further split up into the following sub-topics:
  - Can the previous algorithm in [1] be improved by attempting different methods of EGO-motion rectification and alignment of multiple LiDAR point clouds?
  - Is it possible to detect and remove ghost objects in the data, for example using neural networks or geometry-based methods, allowing for more accurate localization?
- The scenery of the route is dependent on the season and the weather. Under different weather conditions the sensors capture different data from the same location, so matching features between the two point clouds becomes harder. How well can the devised algorithm localize the test vehicle under different conditions?

1.3 Scope and boundaries

Within the scope of the thesis is the following:

- Design and implement an algorithm that localizes the test vehicle within the reference point cloud.
- Usage of data collected from all sensors mounted on the test vehicle (camera, LiDAR, Inertial measurement unit (IMU), GPS).
- Driving the test vehicle and collecting data.

Parts that are outside of the scope:

- Any hardware design or sensor setup will be omitted from the thesis
- The algorithm will be run offline and therefore any evaluation of the algorithm's runtime is left out of this thesis.

1.4 Previous work on the system

Trying to localize the test vehicle in the reference point cloud was done in a previous Master's thesis by Måns Östman and Gunnar Blomwall [1]. Their study found that they were able to achieve fair accuracy, although such a system is very susceptible to noise which can be produced by moving objects etc.
Östman and Blomwall's algorithm tries to align a local LiDAR scan to the reference cloud, using an algorithm called the Generalized Iterative Closest Point (GICP). Their work and findings can be used as a basis for this thesis since they got a working algorithm which can be modified and extended in order to obtain better localization. Adding information from images captured by the test vehicle's camera, refining the LiDAR point clouds by means of pre-processing and exploring modifications of the GICP are some possible ways of improvement.

1.5 Structure of the thesis

The thesis is structured into nine main chapters. Following the *Introduction* chapter the second chapter is *System Description*, which explains the system setup and equipment used for data collection in the thesis. The next chapter is *Theory* which explains the underlying theoretical aspects of the different parts of the work. The fourth chapter, *Algorithm overview and preparation*, presents the structure of the implemented algorithm and how the algorithm is divided into two main parts. It further explains the implementation of external data sources, which are not part of the main algorithm but are used as inputs to the algorithm. Chapters five and six are then *Point Cloud Processing* and *Localization*, which explain in detail how the two major parts of the algorithm work. The methodology chapters are followed by *Validation of point cloud processing* where independent solutions in the thesis are verified. The last two chapters are *Results* and *Analysis and Conclusion* where the outcome of the localization is presented and evaluated.

2
System Description

This chapter describes the test vehicle used for data collection and the reference point cloud to localize within.

2.1 The test vehicle

This section briefly describes the sensor setup of the test vehicle and information about the sensors themselves.

Figure 2.1: REVERE test vehicle, Volvo XC90 2.0 T6 (Snowfox).

2.1.1 The test vehicle and sensor layout

Chalmers University of Technology has access to a vehicle which is used for research on autonomous driving. The vehicle to be localized in the thesis is of the type Volvo XC90 and can be seen in Figure 2.1.
Besides the vehicle's inbuilt sensor system, the test vehicle is equipped with a GPS/ IMU unit from Applanix, a Velodyne Light Detection And Ranging (LiDAR) as well as a set of cameras. The GPS/ IMU unit is an Applanix LV-220 and fuses GPS measurements with measurements from an inertial measurement unit (IMU), resulting in a more accurate estimate of the position. The LiDAR produces point clouds of the proximity of the test vehicle, where a single point cloud is much less dense than the dense reference map.
The data collection vehicle is also coupled to a real-time kinematic (RTK) positioning system which refines and updates the GPS measurements based on knowledge

of how the current atmospheric conditions affect the GPS signal. The test vehicle is further equipped with one Basler camera facing the driving direction.

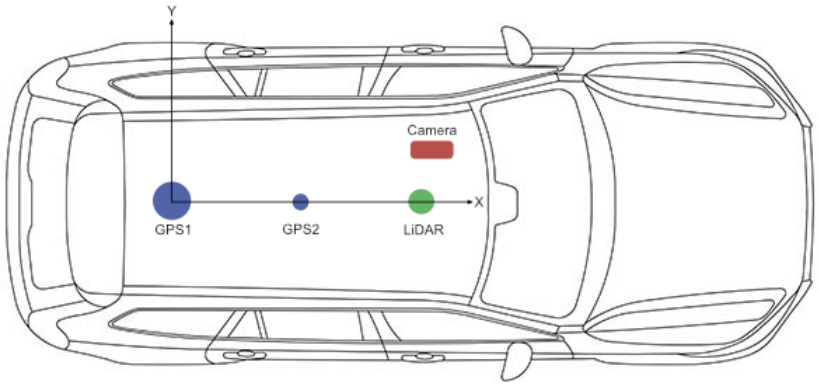


Figure 2.2: The test vehicle sensor layout is shown above. Two GPS antennas allow for accurate localization of position and orientation. The camera and LiDAR are placed towards the front of the vehicle to allow for data capture largely in the driving direction. The vehicle coordinate frame is also depicted in the figure.

2.1.2 Applanix LV-220 with RTK

The Applanix LV-220 is a navigation system integrating inertial measurements and values from two GPS sensors to obtain high accuracy positioning. The GNSS unit receives GPS updates at a frequency of 10 Hz and is updated in intervals of 100 Hz with IMU measurements. GPS1 is used for positioning of the system and the combination of GPS1 and GPS2 are used for azimuth ambiguity resolution. This results in positional accuracy of less than $10cm$ and a pose estimation accuracy of less than 0.2°. In addition to the base system, a Real Time Kinematic (RTK) system is used to obtain higher localization accuracy. An RTK system connects to a base station which corrects the GPS signals received by the Applanix unit, according to current factors such as the current atmospheric conditions. Both antennas for the system are mounted on top of the test vehicle, as shown in Figure 2.2.

2.1.3 Velodyne HDL-32E

The test vehicle is equipped with a Velodyne Light detection and ranging scanner (LiDAR) scanner, positioned towards the front of the vehicle, as shown in Figure 2.2. A LiDAR emits laser beams into its surroundings and measures information about the returning reflected beam. This allows a LiDAR to measure the location of an object with respect to itself. The Velodyne HDL-32E emits 32 layers of laser beams as it scans its environment over a 360° sweep to complete one scan. The LiDAR has a range of 70m and collects consecutive point clouds at a frequency of 10Hz. The density of layers and 360° range allows the LiDAR to record up to 72 000 points in one scan. Due to the LiDAR's mounting position on the test vehicle, a 270° forward facing Field of View can be obtained, due to some obstruction of the vehicle itself towards the rear.

2.1.4 Basler camera

The Basler acA2040-35gc is a compact camera fitted on top of the test vehicle, facing in the driving direction. It contains a $7.1mm \times 5.3mm$, 3.2 megapixel Sony CMOS sensor, which can capture images at a rate of 36 frames per second. It is also equipped with a feature set developed by Basler for in-camera image optimization.

2.2 Test route

The test route is set up in Gothenburg, Sweden, between Chalmers' Johanneberg campus and Chalmers' Lindholmen campus. The test route was selected as part of a larger research topic termed *Project-Copplar*. This route is driven in both directions, where the blue route highlights the trajectory from Johanneberg to Lindholmen, and green highlights the return path.

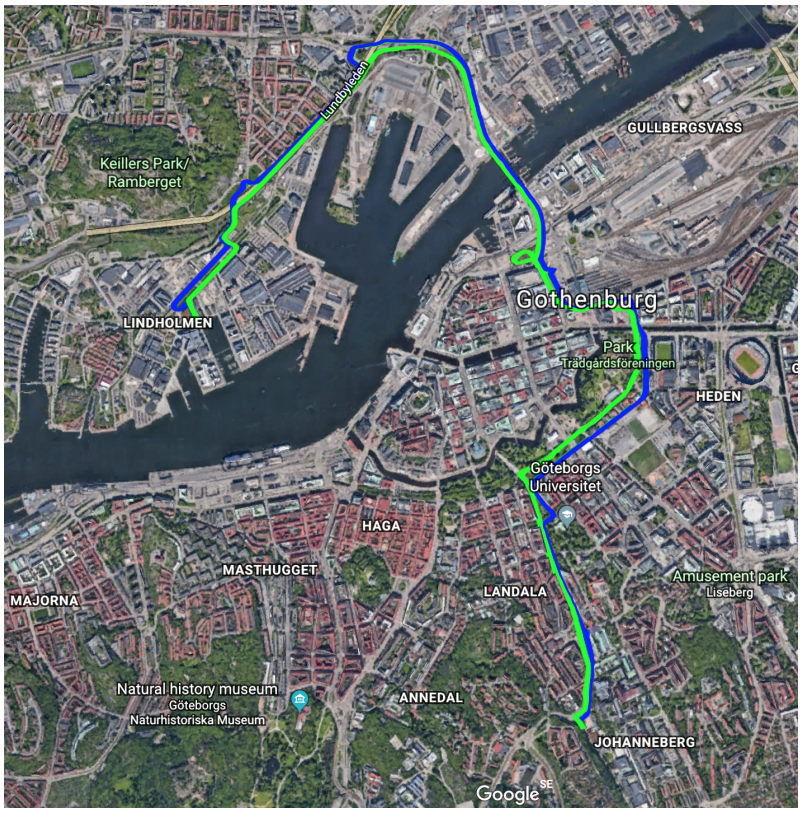


Figure 2.3: The Copplar test route indicated on a map of Gothenburg. The blue line represents the path from Chalmers Johanneberg to Chalmers Lindholmen, and the green line represents the return path. Photo courtesy of Google Earth.

2.3 The reference point cloud

In order to have a reference map in which to localize the test vehicle, a dense point cloud has been built of the urban route between Chalmers' two campuses in

Gothenburg. This reference point cloud has been created using data collected from several drives through the route by a vehicle equipped with an RTK GPS, LiDAR and 360°cameras. Information from the GPS, the LiDAR and the cameras is then combined to create the reference point cloud map, making the point cloud highly detailed and accurate. The point cloud contains information about every point's x, y, z global coordinates and each point is in RGB color. The point cloud also contains information about the LiDAR reflectivity of every point. The data used for creating the cloud was captured in February 2018, when there was no snow visible on the streets. The cloud is stored in multiple files, making it possible to load and view only selected parts of the cloud.

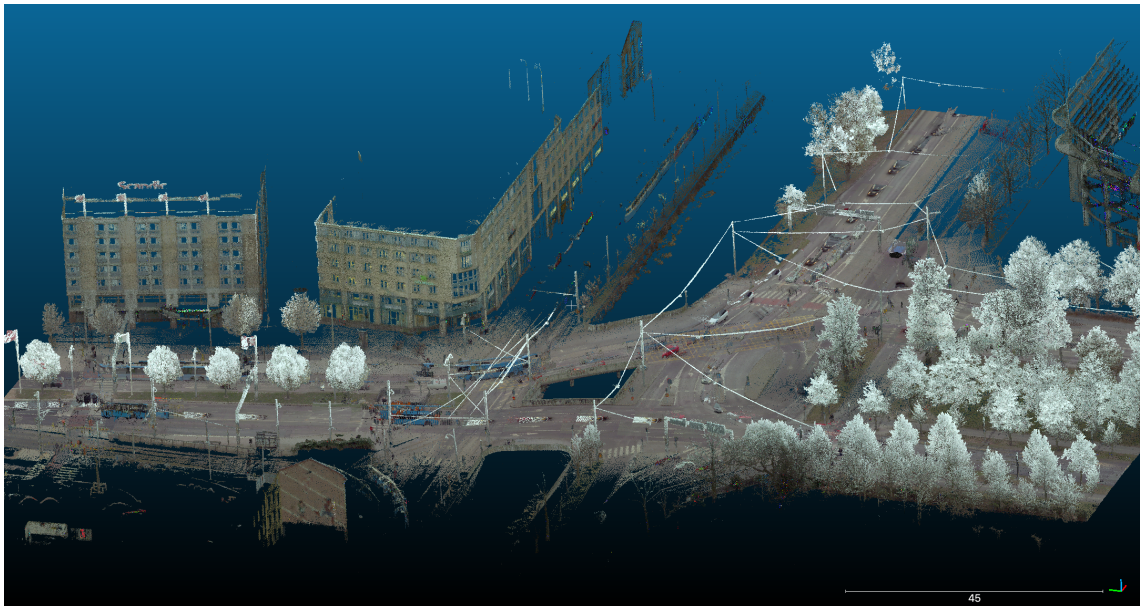


Figure 2.4: Image showing a section of the dense reference point cloud.

3

Theory

This section gives a background to the theoretical aspects of the thesis and explains the theory used in the work.

3.1 Pose of a rigid body

A rigid body in space can be fully described by its *position* and *orientation* (*pose*) with respect to a reference frame [2].

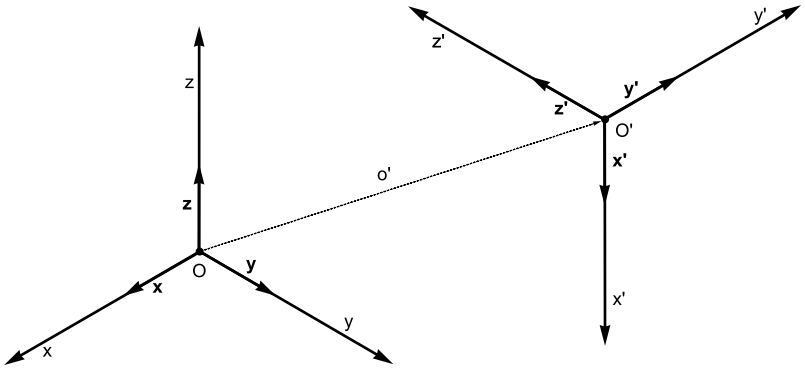


Figure 3.1: Position and orientation of a rigid body. The relative location from an objects body frame (with origin *O*') with respect to a reference frame (with origin *O*). **x, y, z** denote the unit vectors along axes x, y, z respectively for the world frame, and likewise do **x', y', z'** denote the unit vectors along axes x', y', z' in the body frame.

In the above Figure 3.1 the reference frame *O* contains the unit vectors **x, y, z** of the frame axes.
The position of the origin of the body frame *O'* of the rigid body can be denoted as

$$\mathbf{o}' = o'_x\mathbf{x} + o'_y\mathbf{y} + o'_z\mathbf{z} \tag{3.1}$$

Here o'_x, o'_y, o'_z represent the components of the vector $\mathbf{o}' \in \mathbb{R}^3$ along the frames of its axes. The position of the origin of the bodyframe O' can be written as

$$\mathbf{o}' = \begin{bmatrix} o'_x \\ o'_y \\ o'_z \end{bmatrix} \tag{3.2}$$

The rigid body orientation can be represented with use of an orthonormal frame attached to the reference frame and express its unit vectors with respect to the reference frame [2]. $O' - x'y'z'$ will denote such a frame with origin O' and unit vectors $\mathbf{x}$',$\mathbf{y}$',$\mathbf{z}$' along the frame axes. With respect to the reference frame, these vectors can then be expressed as

$$\begin{aligned} \mathbf{x}' &= x'_x\mathbf{x} + x'_y\mathbf{y} + x'_z\mathbf{z} \\ \mathbf{y}' &= y'_x\mathbf{x} + y'_y\mathbf{y} + y'_z\mathbf{z} \\ \mathbf{z}' &= z'_x\mathbf{x} + z'_y\mathbf{y} + z'_z\mathbf{z} \end{aligned} \tag{3.3}$$

3.1.1 Homogeneous Coordinates

Homogeneous (or projective) coordinates are a system of coordinates used in projective geometry. An advantaged gained when using homogeneous coordinates is that it is possible to represent points at infinity, which is not possible in Cartesian coordinates. Homogeneous coordinates also make it possible to represent vector operations such as rotation and translation as matrices, by which the vector is multiplied [3].

Points $p = (x, y)$ and $P = (x, y, z)$ in Cartesian coordinates are given by

$$p = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \qquad P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3.4}$$

in homogeneous coordinates. There is no information added or removed from the point by using either one of the coordinate systems and a point in homogeneous coordinates can be transformed to Cartesian coordinates by diving the positional elements by the last one.

Points at infinity are represented by a 0 in the last element of the point:

$$p_\infty = \begin{pmatrix} x \\ y \\ 0 \end{pmatrix} = (\frac{x}{0}, \frac{y}{0}) \to (\infty, \infty). \tag{3.5}$$

3.1.2 Euler angles

The Euler angles are a set of three angles by which any orientation of a body can be described with respect to a fixed coordinate system. A representation of an orientation can be performed with the three angles

$$\Phi = \begin{bmatrix} \phi & \psi & \theta \end{bmatrix}^T \tag{3.6}$$

These angles represent the angular difference from rotations about the X, Y and Z axes respectively.
In order to describe a rotation of an object, Euler's rotation theorem states that any rotation may be described by a combination of rotations about these three angles. The three rotations are sequentially applied to formulate the final orientation of an object. Common sequences of axes around which these rotations are performed are ZYZ or ZYX. An example of a ZYX rotation is depicted in Figure 3.2

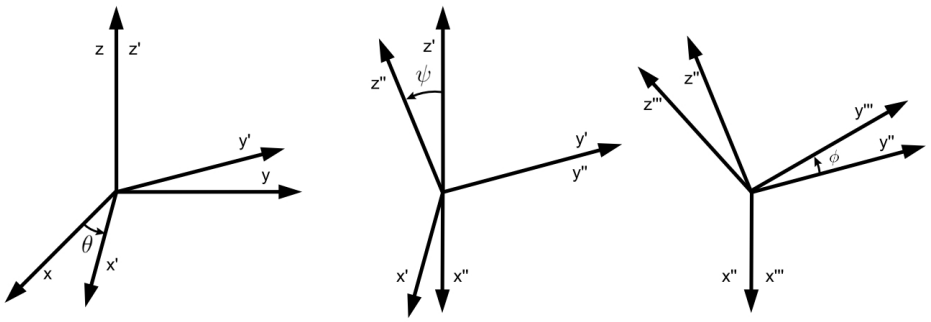


Figure 3.2: A combination of 3 Euler angle rotations in the ZYX order. The axis around which a rotation is made remains stationary, and the updated (new) axes are identified by an added ' to the axis name.

3.1.3 Rotation matrices

The body orientation with respect to a reference frame can be denoted in a more compact matrix form, termed a *rotation matrix*

$$\mathbf{R} = \begin{bmatrix} \mathbf{x}' & \mathbf{y}' & \mathbf{z}' \end{bmatrix} = \begin{bmatrix} x'_x & y'_x & z'_x \\ x'_y & y'_y & z'_y \\ x'_z & y'_z & z'_z \end{bmatrix} = \begin{bmatrix} \mathbf{x}'^T\mathbf{x} & \mathbf{y}'^T\mathbf{x} & \mathbf{z}'^T\mathbf{x} \\ \mathbf{x}'^T\mathbf{y} & \mathbf{y}'^T\mathbf{y} & \mathbf{z}'^T\mathbf{y} \\ \mathbf{x}'^T\mathbf{z} & \mathbf{y}'^T\mathbf{z} & \mathbf{z}'^T\mathbf{z} \end{bmatrix} \tag{3.7}$$

The column vectors of the matrix $\mathbf{R}$ represent the unit vectors of an orthonormal frame, and hence are themselves mutually orthogonal

$$\mathbf{x}'^T\mathbf{y}' = 0 \quad \mathbf{y}'^T\mathbf{z}' = 0 \quad \mathbf{z}'^T\mathbf{x}' = 0 \tag{3.8}$$

They also have the unit form [2]

$$\mathbf{x}'^T\mathbf{x}' = 1 \quad \mathbf{y}'^T\mathbf{y}' = 1 \quad \mathbf{z}'^T\mathbf{z}' = 1 \tag{3.9}$$

This results in the property of $\mathbf{R}$ being an orthogonal matrix

$$\mathbf{R}^T\mathbf{R} = \mathbf{I}_3 \tag{3.10}$$

where $\mathbf{I}_3$ is a 3×3 identity matrix.
A further property of a rotation matrix can be observed by postmultiplying both

sides of Equation (3.10) with the inverse matrix $\mathbf{R}^{-1}$. This yields the result of the transpose of $\mathbf{R}$ being equal to the inverse of $\mathbf{R}$

$$\mathbf{R}^T = \mathbf{R}^{-1} \tag{3.11}$$

Further, $\det(\mathbf{R}) = 1$ for a right-handed frame, and $\det(\mathbf{R}) = -1$ for a left-handed frame.

3.1.4 Affine and Euclidean transformations

In the field of projective transformations of transforming or mapping an object to a new representation of itself ($\mathcal{P}^n \mapsto \mathcal{P}^n$), there are several special cases contained, of which one such case is the affine transformation.
An affine transformation is a linear mapping, preserving points, straight lines and planes. Such transformations are commonly used to correct geometric distortions or deformations in images, but also to denote the translation and rotation of an object between two coordinate frames. An affine transformation is also able to represent a combination of translation, rotation, scaling, homotethy, shear mapping and composition. The order of combining, for instance, multiple rotations, is of importance, since the frame of rotation will vary between each one of these transformations, and will affect the starting point of the consecutive rotation.
When a transformation consists of only a translation and a rotation, it can be classified as a Euclidean/Rigid transformation. Euclidean transformations are a subspace of affine transformations, having all the properties of affine transformations and further preserving the Euclidean distances between points.
In order to denote the rotation $\mathbf{R}$ of an object, a rotation matrix can be applied, as described in the above Section 3.1.3. A translation of origins from the original coordinate system to the origin of the new location can be denoted by a translation vector $\mathbf{t}$,

$$\mathbf{t} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.12}$$

In order to describe a Euclidean transformation for a set of points $\mathbf{x}_a \in \{A\}$ to $\mathbf{x}_b \in \{B\}$, the rotation and translation are combined as

$$\mathbf{x}_b = \mathbf{R}\mathbf{x}_a + \mathbf{t} \tag{3.13}$$

Represented in homogeneous coordinates Equation 3.13 becomes

$$\begin{bmatrix} \mathbf{x}_b \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}_a \\ 1 \end{bmatrix} \tag{3.14}$$

where the Euclidian transformation is represented by $\mathbf{H}$

$$\mathbf{H} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{3.15}$$

The matrix $\mathbf{R}$ is an invertible $n \times n$ matrix, $\mathbf{t}$ is an $n \times 1$ vector, and $\mathbf{0}$ denotes an $1 \times n$ zero vector.

3.2 Numerical integration

Numerical integration is when an approximate solution is sought to the integral

$$\int_a^b f(x)dx \tag{3.16}$$

where $f(x)$ is the function to be integrated and $[a, b]$ is the interval which the function should be integrated over. The solution of the integral can be thought of as the area under the curve defined by f on the interval.
A simple and naive numerical approximation of the solution to Equation (3.16) is

$$S = [F(x)]_a^b = \int_a^b f(x)dx \approx f(a)(b-a) \tag{3.17}$$

where the function f is assumed to be constant over the interval and so the area S is approximated as the area of a square with height $f(a)$ and width $(b-a)$.

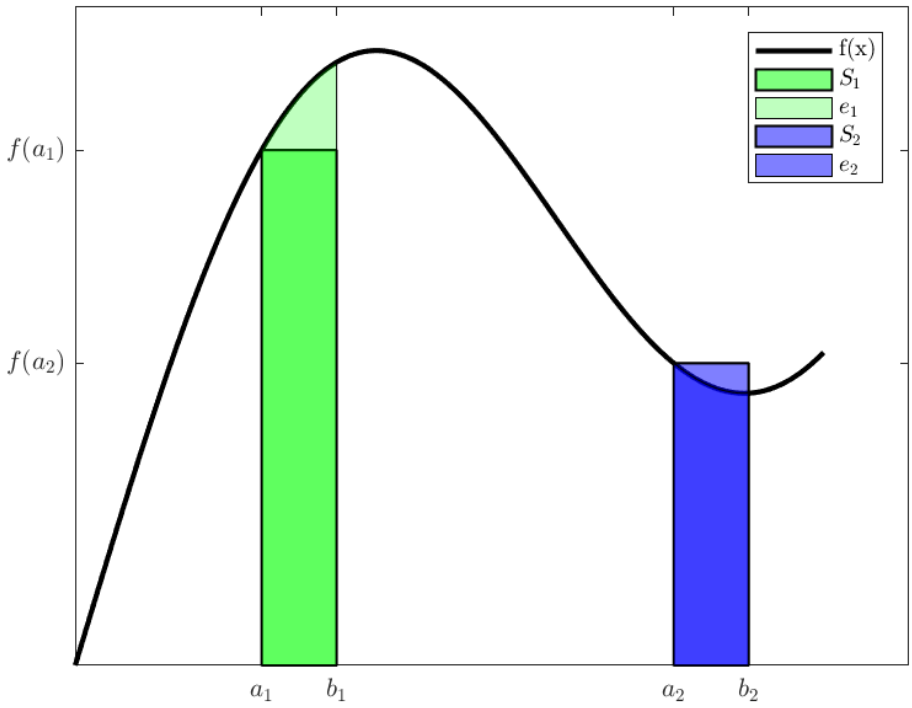


Figure 3.3: Numerical integration of $f(x)$ over the two intervals $[a_1, b_1]$ and $[a_2, b_2]$ using the approximation given by Equation (3.17). The errors in the estimations are visualized by e_1 and e_2.

As can be seen from Figure 3.3, the approximation used is quite crude. The area S_1 should be larger by as much as e_1 while S_2 should be smaller by as much as e_2.
The magnitude of the error in the approximation is affected by the size of the interval which is being integrated over, as well as the behavior of the function over

the interval. If the interval $[a, b]$ is large, the numerical approximation gets worse and the error increases. When the function f is constant or close to being constant over the interval $[a, b]$, the approximate solution is close to the true solution of the integral.

3.2.1 Integration drift

In some cases the use of a variable X is required when the only information available about X is it's derivative $\dot{X} = x$. In such cases it is possible to estimate X using numerical integration using the derivative x as f and an appropriate interval $[a, b]$. This integration of the derivative means that any errors in the values of x also get integrated and so the error in the approximation of X starts to drift off from the true value as more erroneous values of x get integrated. This is called *integration drift* and is an error term which arises from the errors in x, independent of the error in the numerical integration.

3.3 Camera equation

Cameras project 3D objects within their field of view on to a two-dimensional image plane. A simplified model of this projection is the pinhole camera, characterized by the so-called *camera equation*

$$\lambda x = PX \tag{3.18}$$

Equation (3.18) above describes the mapping of 3D points X onto 2D, x, through P. λ denotes the depth and the mapping P is usually referred to as the camera matrix, a 3×4 matrix describing the camera parameters. The camera Equation (3.18) above is linear and does not take into account any nonlinearities such as lens distortion [4].
P can be further split up into

$$P = K \begin{bmatrix} R & t \end{bmatrix} \tag{3.19}$$

where K is a 3×3 mapping which denotes the so-called intrinsic parameters, while the 3×4 matrix $\begin{bmatrix} R & t \end{bmatrix}$ denotes the extrinsic parameters, both of which are described below in further detail.

3.3.1 Intrinsic Parameters

Intrinsic camera parameters refer to the effects of the camera and lens itself imposed on the image. They are given by the matrix K and describe how the image plane in 3D is mapped to the image coordinate system in 2D in the unit pixels. The intrinsics are determined by the parameters of the lens such as principal point, focal length and skew on the following form:

$$K = \begin{bmatrix} \gamma f & sf & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.20}$$

f denotes the focal length of the camera and is the parameter which rescales the image coordinates into pixels. The principal point (x_0, y_0) is the center of the image in image coordinates and the center point of the 3D coordinate system gets mapped to the principal point. The parameter γ is the aspect ratio and handles the case when the pixels of the camera are not square so the re-scaling needs to be done differently in the x- and y-axes. The skew s corrects the pixels if they are not rectangular but tilted [4].

3.3.2 Extrinsic parameters

The extrinsics $\begin{bmatrix} R & t \end{bmatrix}$ describe the orientation and location of the camera with respect to an external reference system. I.e. the extrinsic parameters are a mapping from an external 3D coordination system to the 2D of the camera where R is a 3×3 rotation matrix and t is a 3×1 translation vector per the discussion above. A general position and orientation of the camera with respect to the LiDAR can be viewed in the sensor layout in Figure 2.2. The external coordination reference system can be thought of as the coordination system in which the LiDAR point cloud is measured and has the LiDAR as it's center.

3.4 Perspective n point and minimal solver

Perspective n point is the problem of obtaining the extrinsic parameters for a calibrated camera from n many known point correspondences between 2D image points and 3D world points [5]. The solution sought is the parameters R and t in the calibrated camera equation:

$$\lambda_i x_i = \begin{bmatrix} R & t \end{bmatrix} X_i \tag{3.21}$$

The equation has twelve unknowns but only six degrees of freedom since R needs to be orthogonal. Enforcing R to be orthogonal requires a delicate solution but Gao et al. [5] showed that by using only three points the problem could be simplified to the solution of two quadratic equations, yielding up to four viable solutions.

3.4.1 Deriving the minimal solver

To be able to solve the Perspective 3 point problem a set of three matching 2D and 3D points is needed. The image points $\left(x_1, x_2, x_3\right)$ need to be in homogeneous coordinates and normalized with the K matrix and a coordination change is needed for the 3D points $\left(X_1, X_2, X_3\right)$.

The origin of the three dimensional coordination system is moved to the coordinates of the first point so the 3D points become $\hat{X} = \left(\mathbf{0}, \hat{X}_2, \hat{X}_3\right)$. Next, two orthonormal vectors S_1 and S_2 are formed through the Gram-Schmidt process. S_1 is set as $\hat{X}_2$ scaled to unit length,

$$S_1 = \frac{\hat{X}_2}{||\hat{X}_2||} \tag{3.22}$$

S_2 is formed by subtracting from $\hat{X}_2$ the projection of $\hat{X}_2$ onto S_1:

$$S_2 = \hat{X}_2 - (\hat{X}_2{}^T S_1)S_1 \tag{3.23}$$

and then scaling to unit length,

$$S_2 := \frac{S_2}{||S_2||} \tag{3.24}$$

Now a transformation matrix can be formed

$$S = \Big(S_1, S_2, S_1 \times S_2\Big) \tag{3.25}$$

and the coordination change of the 3D points is possible:

$$\widetilde{X} = S^T \hat{X} = \Big(\mathbf{0}, \widetilde{X_2}, \widetilde{X_3}\Big) \tag{3.26}$$

The transformation S ensures that $\widetilde{X_2}$ and $\widetilde{X_3}$ are on the form $\widetilde{X_2} = \begin{pmatrix}\widetilde{x_{2,1}} & 0 & 0\end{pmatrix}^T$ and $\widetilde{X_3} = \begin{pmatrix}\widetilde{x_{3,1}} & \widetilde{x_{3,2}} & 0\end{pmatrix}^T$.

As shown in Section 3.1.3, the matrix R can be written as

$$R = \begin{bmatrix}\mathbf{a} & \mathbf{b} & \mathbf{c}\end{bmatrix} \tag{3.27}$$

where $\mathbf{a}$ and $\mathbf{b}$ are such that

$$\mathbf{a}^T\mathbf{b} = 0 \tag{3.28}$$

and

$$\mathbf{a} \times \mathbf{b} = \mathbf{c} \tag{3.29}$$

Inserting R and the transformed point $\widetilde{X_1}$ into Equation (3.21) gives

$$t = \lambda_1 x_1 \tag{3.30}$$

Using this and inserting $\widetilde{X_2}$ and $\widetilde{X_3}$ into Equation (3.21) yields a linear system in $\mathbf{a}, \mathbf{b}$ and λ_2, λ_3:

$$\begin{pmatrix}
\widetilde{X_{2,1}} & 0 & 0 & 0 & 0 & 0 & -x_{2,1} & 0 \\
0 & \widetilde{X_{2,1}} & 0 & 0 & 0 & 0 & -x_{2,2} & 0 \\
0 & 0 & \widetilde{X_{2,1}} & 0 & 0 & 0 & -x_{2,3} & 0 \\
\widetilde{X_{3,1}} & 0 & 0 & \widetilde{X_{3,2}} & 0 & 0 & 0 & -x_{3,1} \\
0 & \widetilde{X_{3,1}} & 0 & 0 & \widetilde{X_{3,2}} & 0 & 0 & -x_{3,2} \\
0 & 0 & \widetilde{X_{3,1}} & 0 & 0 & \widetilde{X_{3,2}} & 0 & -x_{3,3}
\end{pmatrix}
\begin{pmatrix}\mathbf{a} \\ \mathbf{b} \\ \lambda_2 \\ \lambda_3\end{pmatrix} =
\begin{pmatrix}-x_{1,1} \\ -x_{1,2} \\ -x_{1,3} \\ -x_{1,1} \\ -x_{1,2} \\ -x_{1,3}\end{pmatrix} \tag{3.31}$$

The linear system presented in Equation (3.31) has 8 unknowns but only 6 equations meaning that explicit solution can not be obtained for $\mathbf{a}$ and $\mathbf{b}$. It is however possible to get a solution for λ_2 and λ_3 in terms of $\mathbf{a}$ and $\mathbf{b}$. Combining that solution with the orthogonal constraints from Equations (3.28) and (3.29) yields the two aforementioned quadratic equations needed for solving the problem.

Solving the two quadratic equations gives the two sought orthogonal vectors $\mathbf{a}$ and $\mathbf{b}$. This can give up to four different solutions.

Now the rotation matrix can be formed:

$$R = \frac{1}{||\mathbf{a}||}\begin{bmatrix}\mathbf{a} & \mathbf{b} & \frac{1}{||\mathbf{a}||}(\mathbf{a}\times\mathbf{b})\end{bmatrix} \tag{3.32}$$

and the translation t needs to be modified accordingly:

$$t = \frac{1}{||\mathbf{a}||}\lambda_1 x_1 \tag{3.33}$$

Now the only thing left is to transform the solutions back into the original coordinate system:

$$\begin{aligned} P &= \begin{bmatrix} RS^T & -RS^T X_1 + t\end{bmatrix} \\ &:= \begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix} \end{aligned} \tag{3.34}$$

3.5 Local optimization and bundle adjustment

It is possible to refine and improve a calibrated camera solution $P = \begin{bmatrix}R & t\end{bmatrix}$ by solving an optimization problem. The goal of the optimization is to minimize the *reprojection error* of the camera, which is the distance between the true 2D point x and the projection of the corresponding 3D point X.

The optimization is formulated as

$$min \quad \sum_{j=1}^{m}\left|\left|\left(x_{j,1} - \frac{\mathbf{a}X_j + t_1}{\mathbf{c}X_j + t_3}, \quad x_{j,2} - \frac{\mathbf{b}X_j + t_2}{\mathbf{c}X_j + t_3}\right)\right|\right|^2 \tag{3.35}$$

where $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are the columns of R and m is the number of point correspondences. Here the assumption is made that the noise in the measured points is Gaussian and the formulation in Equation (3.35) is called *non-linear least squares*. The formulation has no closed form solution so the optimization is therefore a local optimization. The non-linear least squares minimization of the reprojection errors is referred to as *bundle adjustment* in Computer Vision literature [3].

3.5.1 Gradient descent and Levenberg-Marquardt

A general least squares minimization problem has the form

$$\min_v ||f(v)||^2 \tag{3.36}$$

where the objective is to minimize f with respect to the variables v. This is solved by iteratively taking a step in the steepest direction of the function f, starting at a point v_0. The directional derivative is

$$f'_d(v_0) = \nabla f(v_0)^T d \tag{3.37}$$

where d is a vector of unit length that ensures that the step is taking in the steepest direction of the gradient

$$d = \frac{-\nabla f(v_0)}{|\nabla f(v_0)|} \tag{3.38}$$

The next point is then simply selected as

$$v_1 = v_0 + \lambda d \tag{3.39}$$

where λ is selected so $f(v_1) < f(v_0)$. This procedure is then repeated until a local minimum is found.

This simple gradient descent approach can run into problems with convergence speed and stability at points far off from the minimum. Another method for selecting the direction d is Levenberg-Marquardt, an optimization method which counters the problems faced by the steepest descent method by selecting the update step in a more complex manner [4].

The update step in Levenberg-Marquardt is chosen as the solution to the minimization problem

$$\min_d ||r(v_0) + J(v_0)d||^2 + \lambda ||d||^2 \tag{3.40}$$

where $J(v_0)$ is the Jacobian of the function r evaluated at v_0. After solving for d, Equation (3.40) yields

$$d = (J(v_0)^T J(v_0) + \lambda I)^{-1} J(v_0)^T r(v_0) \tag{3.41}$$

The update rule in Levenberg-Marquardt is then $v_1 = v_0 + d$ and λ in equations (3.40) and (3.41) is a free parameter that can be adjusted between iterations.

3.5.2 Linearization of reprojection error

The Levenberg-Marquardt method suits well for the bundle adjustment problem presented in Equation (3.35). Doing so requires the set of residuals r_i to be minimized in a least squares manner

$$\min_v \sum_i r_i(v)^2 \tag{3.42}$$

Stacking all residuals together in a single vector *r(v)* allows for the notation:

$$\min_v ||r(v)||^2 \tag{3.43}$$

which is the same form as the general problem presented in Equation (3.36).

To be able to use Levenberg-Marquardt in Bundle Adjustment the rotations must be linearized, i.e. it's necessary to linearize terms such as

$$\underbrace{\frac{\mathbf{a}X_j + t_1}{\mathbf{c}X_j + t_3}}_{f_1} \tag{3.44}$$

It is possible to write any rotation R as a multiplication of a given rotation R_0 with the exponential map of a skew-symmetric matrix [4]

$$R = \exp \begin{pmatrix} 0 & -a_1 & -a_2 \\ a_1 & 0 & -a_3 \\ a_2 & a_3 & 0 \end{pmatrix} R_0 \tag{3.45}$$

The skew-symmetric matrix can be factorized into

$$a_1 \underbrace{\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}}_{S_1} + a_2 \underbrace{\begin{pmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}}_{S_2} + a_3 \underbrace{\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}}_{S_3} \tag{3.46}$$

The exponential mapping can be defined as the power series:

$$\exp(A) = \sum_{k=0}^{\infty} \frac{1}{k!} A^k = I + A + \frac{1}{2}A^2 + \frac{1}{6}A^3 + \dots \tag{3.47}$$

Therefore it is possible to express the mapping R from Equation (3.45) as

$$R = (I + a_1 S_1 + a_2 S_2 + a_3 S_3))R_0 \tag{3.48}$$

in the neighbourhood of R_0. This is linear parametrization of R around R_0 which makes it possible to linearize and differentiate terms such as the one presented in Equation (3.44) above. Differentiating Equation (3.44) with respect to the parameters a_1, a_2, a_3, t_1, t_2 and t_3 yields

$$\frac{\delta f_1}{\delta a_1} = \frac{S_1^1 R_0 X_0}{R_0^3 X_0 + t_0^3} - \frac{R_0^1 X_0 + t_0^1}{(R_0^3 X_0 + t_0^3)^2} S_1^3 R_0 X_0 \tag{3.49}$$

$$\frac{\delta f_1}{\delta a_2} = \frac{S_2^1 R_0 X_0}{R_0^3 X_0 + t_0^3} - \frac{R_0^1 X_0 + t_0^2}{(R_0^3 X_0 + t_0^3)^2} S_2^3 R_0 X_0 \tag{3.50}$$

$$\frac{\delta f_1}{\delta a_3} = \frac{S_3^1 R_0 X_0}{R_0^3 X_0 + t_0^3} - \frac{R_0^1 X_0 + t_0^1}{(R_0^3 X_0 + t_0^3)^2} S_3^3 R_0 X_0 \tag{3.51}$$

$$\frac{\delta f_1}{\delta t_1} = \frac{1}{(R_0^3 X_0 + t_0^3)} \tag{3.52}$$

$$\frac{\delta f_1}{\delta t_2} = 0 \tag{3.53}$$

$$\frac{\delta f_1}{\delta t_3} = -\frac{R_0^1 X_0 + t_0^1}{(R_0^3 X_0 + t_0^3)^2} \tag{3.54}$$

where S_i^j is the jth column of S_i, R_0^j is the jth column of R_0 and t_0^i is the ith element of t_0. The derivation of the second reprojection $f_2 = \dfrac{\mathbf{b}X_j + t_2}{\mathbf{c}X_j + t_3}$ is done in the same way.

At this stage, it is possible to form the Jacobian of the reprojection error and obtain the update step d from Equation (3.41). The starting point for the optimization is the initial pose estimate

$$v_0 = \begin{bmatrix} R_0 & t_0 \end{bmatrix} \tag{3.55}$$

and finally the update rule in bundle adjustment is:

$$v_1 = \begin{bmatrix} R_1 & t_1 \end{bmatrix} = \begin{bmatrix} \exp \begin{pmatrix} 0 & -d_1 & -d_2 \\ d_1 & 0 & -d_3 \\ d_2 & d_3 & 0 \end{pmatrix} R_0 & t_0 + d_{4,5,6} \end{bmatrix} \tag{3.56}$$

where d_i is the ith element of d.

3.6 RANSAC

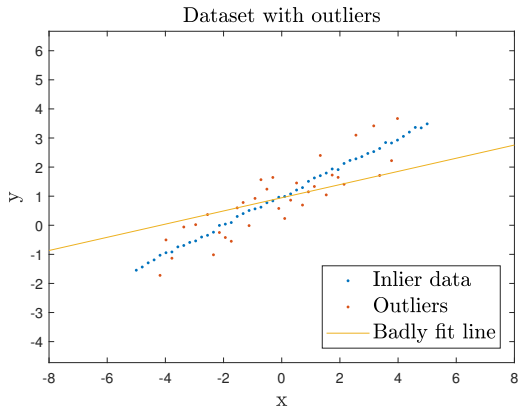
The Random Sample Consensus (RANSAC) is an algorithm to estimate and fit the parameters of a mathematical model to a dataset suffering from outliers. The algorithm iteratively estimates the parameters by randomly selecting a subset from the dataset containing the minimum number of points needed for the estimation and returns the parameters that yield the best results [6].

Given a dataset with points distributed by some set of model parameters, an outlier is a point in the dataset which deviates from the modelled distribution by more than a predefined threshold ϵ. An inlier is a point that falls within the threshold of the model distribution. The objective of the RANSAC algorithm is to estimate this set of model parameters and minimizing the effects of the outliers on the estimated parameters. This is done by selecting a subset from the dataset which contains the minimum number of randomly selected points n needed for the parameter estimation. Using the random points, the parameters are estimated and the estimation is evaluated on how many inliers it gets, i.e. the number of points that fall within the threshold of the model. This procedure is repeated often enough for the model parameters to be estimated by inlier points only. The outcome of the RANSAC is the set of parameters which yielded the most inliers on the dataset.

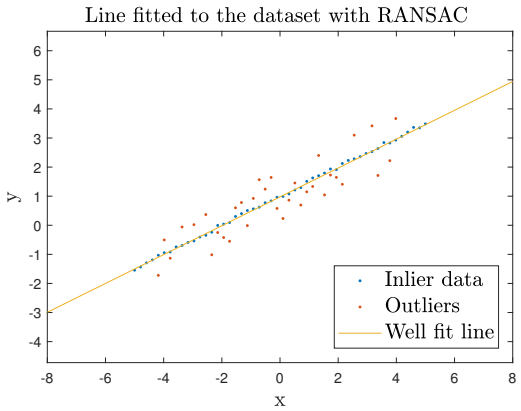
It is possible to determine the expected minimum number of iterations needed for RANSAC to do the parameter estimation with inlier data only. The number of iterations k is given by

$$k = \frac{log(1 - P(\text{success}))}{log(1 - w^n)} \tag{3.57}$$

where w is the probability of a point selected from the set of points being an inlier. The probability of success, $P(\text{success})$, in Equation (3.57), is a free parameter that can be tuned to affect the certainty of RANSAC since a desired high probability of success leads to more iterations which in turn should yield a better outcome.



(a) Dataset to fit a line to. The figure shows an example of one RANSAC iteration where a line has been fitted to the dataset using two randomly selected points. As can be seen, the line is affected by the outliers and does not represent the inliers well.



(b) The results of line fitting with RANSAC. The resulting line aligns well to the dataset and the outliers have no effect on the line.

Figure 3.4: The procedure of fitting a line to a dataset using RANSAC.

Figure 3.4 above shows how well RANSAC is able to fit a line to a dataset that suffers from outliers. The resulting line fits well to the inliers and is not affected by the outliers.

3.7 Deep machine learning

Machine learning is a fast-growing field of artificial intelligence, applied to many practical areas of life to teach computers to perform complex tasks of solving problems. In deep machine learning a neural network is deployed, which is structured and trained in order to learn how to solve these problems without human input.
A neural network consists of an input and output layer and a certain number of hidden layers in between these. An example of a basic neural network structure is depicted in Figure 3.5. Especially the hidden layers are constructed as a combination of multiple interconnected layers of varying functions and computations. Each layer consists of a number of nodes which can each be activated or not, based on the relevance of an input for the network's prediction. The activation of the nodes is done by mathematical equations, known as activation functions, which are attached to each node in a layer. The network processes input data through the layers according to the different weights assigned to connections between nodes in consecutive layers, which allows a network to compute and output a prediction of the best likely course of action to take.
In order to teach a network to correctly compute its output probabilities, it needs to be trained. This can be done by passing large amounts of annotated data through the network, which it will process and evaluate its performance based on the predefined annotations of the data. Individual weights for various layers and nodes will

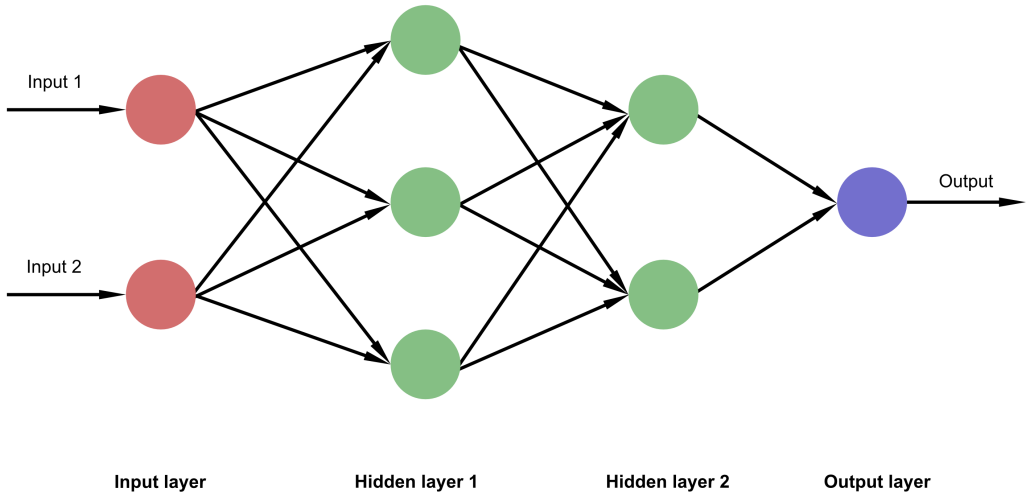


Figure 3.5: Example of a neural network structure. The inputs are passed into one or several input layers. The input layers are interconnected with the next layers, which can be a combination of various hidden layers which finally pass their output to one or more output layers. Each layer can contain several nodes, which passes its output to nodes in the following layer.

hence be updated in a backward manner of passing the result from the output to the front of the network, updating these weight one step at a time. This process is called *backpropagation* which iteratively and recursively updates the weights. This is done by following a gradient descent approach which computes the partial differentials of the activation functions between connected nodes. Thereby, the network is reinforced for positive detections, and penalized for false/negative detections, which updates the network to better perform its assigned task.
Once a network's weights have been trained well, the network can be used to process new, never before seen data and is be able to solve the problem at hand with similar accuracy.

3.7.1 Object detection

One application of deep machine learning is using a network for object detection given an input such as an image or point cloud. An object detection model is trained to be able to identify different predefined object categories within an image. The detected subject is captured within a bounding box, denoting by which image coordinates this object is bounded. It is to be noted that the object does however not necessarily fill the entire bounding box.
Such a model is trained with annotated data containing information about the predefined objects. In this manner, the network is able to learn about certain features or forms which compose a certain object and is able to detect the presence of similar objects in never before seen data sets. Various models have been created to per-

form this task, attempting to solve the object detection problem faster and more accurately.

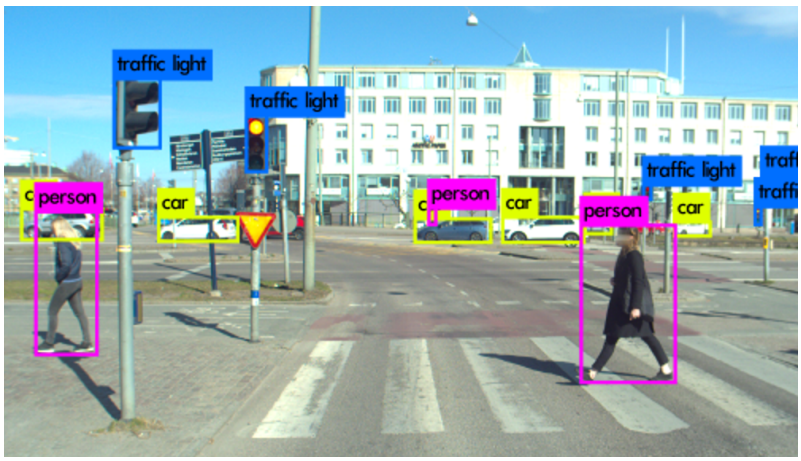


Figure 3.6: Example of object detection on an image from a traffic scene.

3.7.2 Semantic segmentation

Semantic segmentation performs a similar task to object detection in that it classifies categories of objects found in a certain image. However, instead of containing a detected object within a bounding box, semantic segmentation assigns a label to each pixel in the image, labelling it as either one of the prescribed classes or depending on the application, a class of lesser relevance (for e.g. unspecified/background). Since this assignment is done pixel-wise, it allows for a tighter outlining of an object.

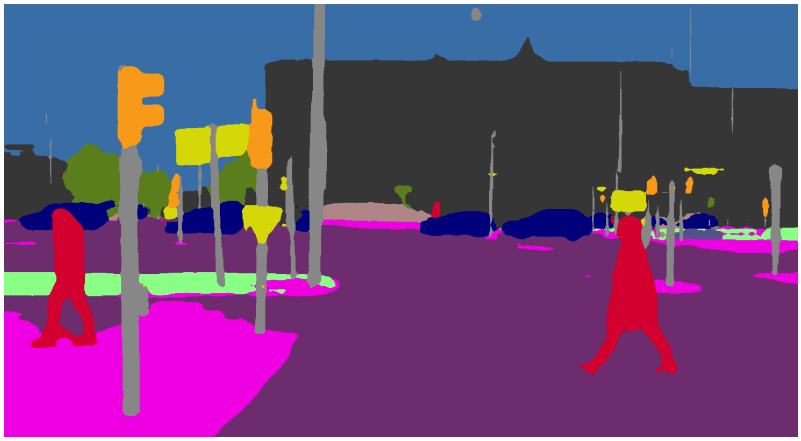


Figure 3.7: Example of semantic segmentation on an image from a traffic scene.

3.8 Point cloud registration

Registration is the process of aligning two point clouds from misaligned initial positions. A registration method needs to be able to finetune the adjustment of the major planes and objects within a point cloud, whilst not being too sensitive to noise or outlier objects.

3.8.1 Iterative Closest Point

The Iterative Closest Point (ICP) algorithm [7] is a registration method which aligns a source point cloud $A = \{a_i\}_{i=1...N}$ to a target point cloud $B = \{b_i\}_{i=1...N}$ given a prior estimate of alignment.
In order to align the two point clouds from differing initial positions, a transformation $\mathbf{T}$ is computed which aligns the source cloud to the target cloud

$$B = \mathbf{T}A \tag{3.58}$$

This alignment is performed in an iterative manner as outlined by the algorithm below:

Algorithm 1: Iterative Closest Point

Initialize $\mathbf{T}_0$;
while *end condition not met* **do**
- Source cloud transformed by transformation $\mathbf{T}$;
- Correspondences computed by finding nearest neighbour for each point;
- Transformation $\mathbf{T}$ computed which minimizes the distance between corresponding points;

end

The initial transformation $\mathbf{T}_0$ is an alignment estimate based on prior information of the rotational and translational difference between the source and target cloud. The first step in the iterational loop transforms the source cloud with the transformation $\mathbf{T}$ to obtain the source cloud in its new estimated position.
Next, the correspondences between the two point clouds can be computed. Since ICP is a point-to-point alignment method, the closest point in cloud B is computed for all points in set A, which yields proposed corresponding pairs $a_i = b_i$. An inlier threshold d_{max} is applied to the point correspondences to discard matches which are too far away from one another, as defined by

$$||b_i - \mathbf{T}a_i|| < d_{max} \tag{3.59}$$

This threshold is added since the two clouds might not always overlap fully, and hence false correspondences can easily be created.
With the remaining point correspondences it is possible to now compute the transformation $\mathbf{T}$, which minimizes the sum of the squared error

$$\mathbf{T} = \underset{argmin}{\mathbf{T}} \frac{1}{N_P} \sum_{i=1}^{N_P} ||b_i - \mathbf{T}a_i||^2 \tag{3.60}$$

$\mathbf{T}$ consists of a rotation matrix $\mathbf{R}$ and a translation matrix $\mathbf{t}$ $\mathbf{T} = [\mathbf{Rt}]$, a_i and b_i are the point correspondences and N_P is the number of points.

3.8.2 Generalized Iterative Closest Point

The Generalized Iterative Closest Point (GICP) method as presented by [8] is a combination of the regular ICP method and point to plane matching, in order to

obtain a more robust point cloud alignment. The difference to the regular ICP method is in the minimization function, while the rest of the algorithm remains intact.
As for ICP, the GICP method uses point clouds A and B and computed their respective point correspondences between these two sets $A = \{a_i\}_{i=1...N}$ and $B = \{b_i\}_{i=1...N}$. As before, all correspondences spaced too far apart are removed.
Within A and B the existence of an underlying set of points $\hat{A} = \{\hat{a}_i\}$ and $\hat{B} = \{\hat{b}_i\}$ is assumed, which generates A and B according to the modelled Gaussian distributions

$$\begin{aligned} a_i &\sim \mathcal{N}(\hat{a}_i, C_i^A) \\ b_i &\sim \mathcal{N}(\hat{b}_i, C_i^B) \end{aligned} \tag{3.61}$$

where C_i^A and C_i^B denote the covariance matrices corresponding to the measured points. In modelling the surface of these clouds, they are assumed to be locally planar. Differently now to ICP, point correspondences are not assumed to match directly to a specific point but rather to be located on the same plane. To model this, the covariance matrix is specified to contain a small covariance ϵ in the surface normal direction, while the covariances in the planar directions are set to 1:

$$\mathbf{C} = \begin{bmatrix} \epsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.62}$$

Next, ϵ has to be aligned to the surface normal vector since it must represent the covariance measured at this point. This is done by rotating the covariance matrix $\mathbf{C}$ with the rotation matrix $\mathbf{R}_{v_i}$ which corresponds to the orientation of the surface normal vector v_i

$$C_i = \mathbf{R}_{v_i} \mathbf{C} \mathbf{R}_{v_i}^T \tag{3.63}$$

Once aligned, [8] makes use of Principle Component Analysis (PCA) to estimate the surface normals. This is done by computing the covariance matrix of the k closest points to i, and from the resulting matrix the eigenvector can be found which corresponds to the smallest eigenvalue.
Assuming perfect correspondences which are geometrically consistent and without sampling errors, the transformation $\mathbf{T}$ represents

$$\hat{b}_i = \mathbf{T}^* \hat{a}_i \tag{3.64}$$

For a rigid transformation $\mathbf{T}$, $d_i^{(T)} = b_i - \mathbf{T}a_i$ is defined and the distribution from which $d_i^{T^*}$ is drawn as

$$\begin{aligned} d_i^{T^*} &\sim \mathcal{N}(\hat{b}_i - (\mathbf{T}^*)\hat{a}_i, C_i^B + (\mathbf{T}^*)C_i^A(\mathbf{T}^*)^T) \\ &= \mathcal{N}(0, C_i^B + (\mathbf{T}^*)C_i^A(\mathbf{T}^*)^T) \end{aligned} \tag{3.65}$$

The Maximum Likelihood Estimator MLE can be used to find the transformation $\mathbf{T}$

$$\begin{aligned} \mathbf{T} &= \underset{T}{argmax} \prod_i p(d_i^{\mathbf{T}}) \\ &= \underset{T}{argmax} \sum_i log(p(d_i^{\mathbf{T}})) \end{aligned} \tag{3.66}$$

Equation (3.66) can then be simplified as

$$\mathbf{T} = \underset{T}{argmax} \sum_i d^{(\mathbf{T})^T} (C_i^B + \mathbf{T} C_i^A \mathbf{T}^T)^{-1} d^{(\mathbf{T})} \tag{3.67}$$

which replaces the minimization step in the regular ICP algorithm.

3.9 Bayesian filtering

Bayesian filtering is an approach for estimating the state of a time-varying system, relying on noisy measurements and a process model [9]. These process and measurement models consist of a sequence of probabilistic distributions, where the model for the state x_k is

$$x_k \sim p(x_k | x_{1:k-1}, y_{1:k-1}) \tag{3.68}$$

where k is the current timestep. The process model is used to represent the dynamics of a given system. In similar manner, the measurement y_k can be described as

$$y_k \sim p(y_k | x_{1:k}, y_{1:k-1}) \tag{3.69}$$

which describes the distribution of the measurements given a certain state. In order to greatly reduce complexity and computation of calculating the full posterior distributions every time a new measurement is obtained, the models are assumed *Markovian*. A valuable property gained by this assumption is that the current state x_k, given a previous state x_{k-1}, can be fully described, independent without the knowledge of the entire sequence of previous states prior to x_{k-1}. Likewise, the measurement y_k give x_k is independent of previous states and measurements. This allows for simplification of the process and measurement models as

$$\begin{aligned} p(x_k | x_{1:k-1}, y_{1:k-1}) &= p(x_k | x_{k-1}) \\ p(y_k | x_{1:k}, y_{1:k-1}) &= p(y_k | x_k) \end{aligned} \tag{3.70}$$

Adding prior information about the initial distribution $p(x_0)$, a general probabilistic state space model can be written as

$$\begin{aligned} x_0 &\sim p(x_0) \\ x_k &\sim p(x_k | x_{k-1}) \\ y_k &\sim p(y_k | x_k) \end{aligned} \tag{3.71}$$

The aim of Bayesian filter in to compute the *marginal posterior distribution* of a state x_k given the measurement history $y_{1:k}$ up to the current time step. This computation starts by *initialization* of the prior, and is performed by recursively by calculating a *prediction* and *update* step for each time instance.
The prediction of state (x_k) at timestep k is given by the *Chapman-Kolmogorov* equation

$$p(x_k | y_{1:k-1}) = \int p(x_k | x_{k-1}) p(x_{k-1} | y_{1:k-1}) dx_{k-1} \tag{3.72}$$

Given the measurement y_k at time instance k, the posterior distribution of x_k can be computed with Bayes' rule:

$$p(x_k | y_{1:k}) = \frac{1}{Z_k} p(y_k | x_k) p(x_k | y_{1:k-1}) \tag{3.73}$$

where the normalization constant Z_k is given as

$$Z_k = \int p(y_k|x_k)p(x_k|y_{1:k-1})dx_k \tag{3.74}$$

3.9.1 Kalman filter

For linear Gaussian process and measurement models, the closed-form solution to the Bayesian filtering equations is the *Kalman filter* [9], described as

$$\begin{aligned} x_k &= A_{k-1}x_{k-1} + q_{k-1} \\ y_k &= H_{k_k} + r_k \end{aligned} \tag{3.75}$$

where the process noise q_{k-1} and measurement noise r_k are

$$\begin{aligned} q_{k-1} &\sim \mathcal{N}(0, Q_{k-1}) \\ r_k &\sim \mathcal{N}(0, R_k) \end{aligned} \tag{3.76}$$

and the prior distribution is Gaussian $x_0 \sim \mathcal{N}(m_0, P_0)$. In the process model, the matrix a_{k-1} is the transition matrix, and in the measurement model H_k is the measurement model matrix. The posterior distribution can be represented by its mean $\hat{x}_{k|k}$ and covariance $P_{k|k}$, since it will also be Gaussian under the stated assumptions. Starting with prior mean x_0 and covariance P_0, the posterior distribution can be computed by the prediction and update steps.
The prediction step consists of

$$\begin{aligned} \hat{x_{k|k-1}} &= A_{k-1}\hat{x}_{k-1|k-1} \\ P_{k|k-1} &= A_{k-1}P_{k-1|k-1}A_{k-1}^T + Q_{k-1} \end{aligned} \tag{3.77}$$

The update step is performed by evaluating the innovation v_k, innovation covariance S_k and Kalman gain K_k

$$\begin{aligned} v_k &= y_k - H_k\hat{x}_{k|k-1} \\ S_k &= H_kP_{k|k-1}H_k^T + R_k \\ K_k &= P_{k|k-1}H_k^TS_k^{-1} \end{aligned} \tag{3.78}$$

and then the final updated mean and covariance can be computed as

$$\begin{aligned} \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_kv_k \\ P_{k|k} &= P_{k-1|k-1} - K_kS_kK_k^T \end{aligned} \tag{3.79}$$

3.9.2 Rauch-Tung-Striebel smoother

The closed-loop smoothing solution, $p(x_k|y_{1:N})$, to the linear model (3.75) can be computed using the Rauch-Tung-Striebel smoother (RTS), also known as the Kalman smoother. The smoother differs from the Kalman filter in that it is conditional on the whole measurement data $y_{1:N}$, whereas a filter only has access to the

measurements prior to and including the current time step. The backward recursion of the Rauch-Tung-Striebel smoother is given as

$$\begin{aligned}
\hat{x}_{k+1|k} &= A_k \hat{x}_{k|k} \\
P_{k+1|k} &= A_k P_{k|k} A_k^T + Q_k \\
G_k &= P_{k|k} A_k^T P_{k+1|k}^{-1} \\
\hat{x}_{k|N} &= \hat{x}_{k|k} + G_k(\hat{x}_{k+1|N} - \hat{x}_{k+1|k}) \\
P_{k|N} &= P_{k|k} - G_k(P_{k+1|k} - P_{k+1|N}) G_k^T
\end{aligned} \tag{3.80}$$

where $\hat{x}_{k|k}$ and $P_{k|k}$ are the mean and covariance respectively, computed by the Kalman filter. The smoothing recursion begins at the final time step N, with $\hat{x}_{N|N} = \hat{x}_{N|k}$ and $P_{N|N} = P_{N|k}$, and iteratively works itself to the beginning of the sequence.

4

Algorithm overview and preparation

This chapter outlines the general structure of the algorithm, including the data flow of external information sources. Following this, the method of implementation of the external sources of *Vehicle data* and *Camera calibration* are explained.

4.1 Algorithm overview

The general structure and information flow of the algorithm is presented in Figure 4.1 below, to provide an overview of the sequence and dependency of certain tasks on others.

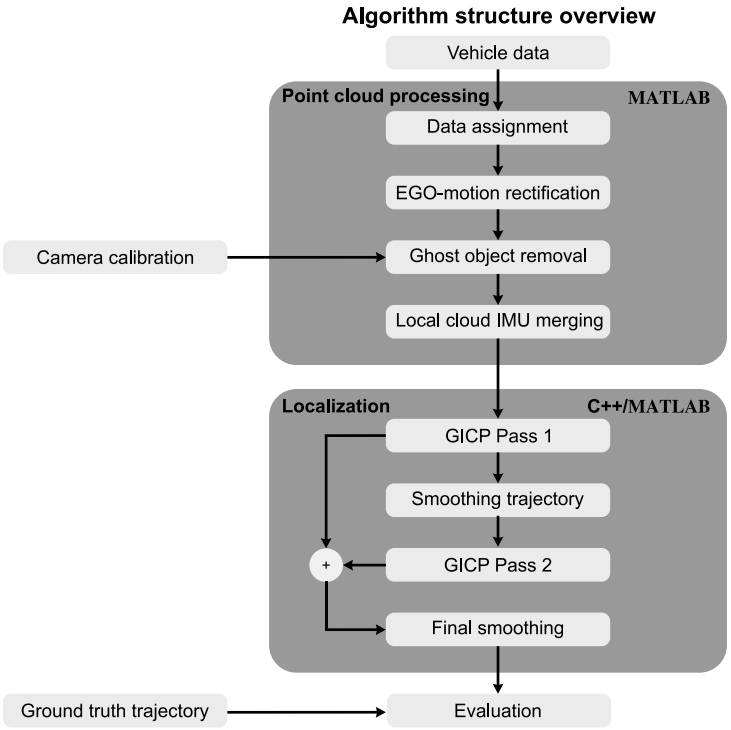


Figure 4.1: A general overview of the main algorithm components. The two main sections are Point cloud processing and Localization, containing their respective subsections. External data into the algorithm is also depicted, showing the flow and combination of information where required.

The collected vehicle data is initially downloaded from REVERE's server, where it is stored and saved to a local drive. Then it is read by the algorithm, being processed and structured so it can be accessed by the subsequent functions. EGO-motion rectification is then the first processing step of the point clouds, to remove all motion distortions. Once the clouds have been rectified, dynamic objects can be removed, before several of these local clouds are merged to form a denser representation of the vehicle's surroundings. Some final preprocessing and downsampling is performed to prepare the data for alignment with the dense reference point cloud, before the GICP is deployed to perform this task. A new trajectory of the vehicle's path through the dense point cloud is then estimated and smoothed. Instances with seemingly bad results are passed through the GICP with a new initial alignment. The results from the first and second GICP passes are combined to keep only the best alignment results and this trajectory is then smoothed again.

4.2 Association of data from different sensor systems

Two structs are created at the start of the algorithm; a Velodyne struct with an element corresponding to each LiDAR cloud and a trajectory struct with an element for each time instance with a trajectory measurement. Each Velodyne element gets assigned the trajectory element with the GPS measurement which is closest to the timestamp of the cloud. Similarly, an image from the Basler camera is associated to every point cloud. This is of high importance as the ghost object removal is removing 3D points in the point clouds based on 2D detections in the images, so the images and clouds need to be from the same time instance. There is a small delay between timestamps of LiDAR clouds and the images from the Basler camera so the image with a timestamp 0.4 seconds later than the LiDAR cloud's timestamp is assigned to the cloud. No image gets assigned to the cloud if there is more than 0.1 second interval between the image and cloud. To do this a binary search algorithm is deployed which assigns an image to a cloud based on the smallest time difference between the image and point cloud timestamps. When merging local clouds the position, angular pose and timestamp of the middle cloud are used with the whole merged cloud.

4.3 Camera calibration and alignment to LiDAR

Camera calibration was performed to obtain the intrinsic and extrinsic parameters of the camera. This is necessary to match to LiDAR scan with the Field of View (FOV) of the camera so that objects in the camera image correspond to their 3D point cloud representations. Camera calibration is only needed to obtain the camera parameters for the camera matrix and is therefore not included in the main parts of the algorithm. Once the camera matrix is determined it can be used for multiple data sets, given that the camera lens has not been altered from when the calibration was run and that the position of LiDAR and camera have not been altered.

4.3.1 Intrinsic calibration

Intrinsic parameters of a camera refer to the inner parameters of the camera as described in Section 3.3.1.
In order to find these parameters, images need to be taken of a large pattern with known dimensions, in order to calculate the variation of the real physical pattern and the perceived image of the pattern by the camera. An 8×11 black and white checkerboard was printed out and mounted onto a planar metal shield. Each checker was of a 43×43mm size.
Multiple images need to be taken of the checkerboard placed throughout the field of view of the camera, filling the image plane. This also needs to be done for various depths from the camera and for various rotations of the board.
The images are then uploaded to the **MATLAB** *camera calibration* tool, which computes the intrinsic parameters, reprojection errors and further constants of the camera, given details about the physical checkerboard size.
The intrinsic parameters so computed are:

$$K = \begin{bmatrix} 1775.02 & 1.15 & 1036.66 \\ 0 & 1773.25 & 690.34 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

The mean reprojection error for this calibration is 0.07879 pixels.

4.3.2 Extrinsic calibration

Extrinsic calibration, in this case, is the computation of the transformation matrix from the camera center to the LiDAR center, as explained in Chapter 3.3.2. This is necessary to align the field of view of the camera to the field of view of the LiDAR, in order to select LiDAR points corresponding to the same object in the camera image.
Initial processing needs to be performed on the image in order to remove the effects of the camera intrinsics on the images. This means undistorting individual images. This is done in **MATLAB** using the *undistortimage* function which reads an image and the intrinsic matrix computed before and outputs the undistorted image.
Once the preprocessing of the images is completed, point correspondences between the LiDAR cloud and the undistorted image need to be selected. These point correspondences representing the same object or position in both data formats need to be selected with high accuracy, in order to avoid misalignment in the computation of the transformation matrix due to poorly selected points.
To further reduce the risk of relying on poor point selections, multiple RANSAC iterations are run in order to randomly select various 3 points and compute the transformation matrix for each set of points, as described in Section 3.6.
Within the RANSAC loop, the minimal solver derived in the theory chapter computes the transformation matrix based on the point selection from RANSAC. The RANSAC loop is iterated typically a couple of thousand times to get a good estimate of the extrinsics.
Given the estimate resulting from the RANSAC loop with the minimal solver, the transformation matrix is refined by a local optimization algorithm, namely the

Levenberg-Marquard method. The RANSAC estimate is used for the initial point in the Levenberg-Marquard. Since the RANSAC estimate is already quite close to the local minimum, the Levenberg Marquard only needs to be iterated around 10 times for it to converge.
Finally, the computed extrinsic parameters obtained are

$$\begin{bmatrix} R & t \end{bmatrix} = \begin{bmatrix} 0.01737 & -0.99985 & 0.00088 & 0.22208 \\ -0.08995 & -0.00244 & -0.99594 & -0.18690 \\ 0.99579 & 0.01723 & -0.08998 & -0.21325 \end{bmatrix} \tag{4.2}$$

5

Point cloud processing

This chapter discusses the processing done on the LiDAR clouds before they are aligned to the dense cloud. Every step of this part of the algorithm is implemented in **MATLAB**.

5.1 EGO-motion rectification

All LiDAR measurements are recorded in a local coordinate frame fixed to the optical center of the LiDAR. Under stationary conditions, all point measurements collected during one LiDAR scan correspond to the same position and orientation of a fixed origin, hence recreating an undistorted 3D representation of its current environment. If, however, the LiDAR's position and orientation varies due to motion of the scanner itself, the collected LiDAR point cloud becomes distorted since the origin of the point cloud is moving over the 0.1 second duration for one scan.

This distortion needs to be accounted for and rectified if the geometry and location of the measured objects are to be relied on for localization purposes. Merriaux et. al present a method [10] for LiDAR point cloud correction, taking into consideration linear motion as well as angular velocity. Since, in their study, roll and pitch rates proved to be minor in comparison to the yaw rate, motion compensation was performed based on planar vehicle motion. The approach presented below is based largely on [10], but modified and adjusted according to the available output data from the vehicle sensors.

Plotting the roll, pitch and heading of the car in Figure 5.1 during a drive through the test route, indicates how well a planar motion model would fit the route.

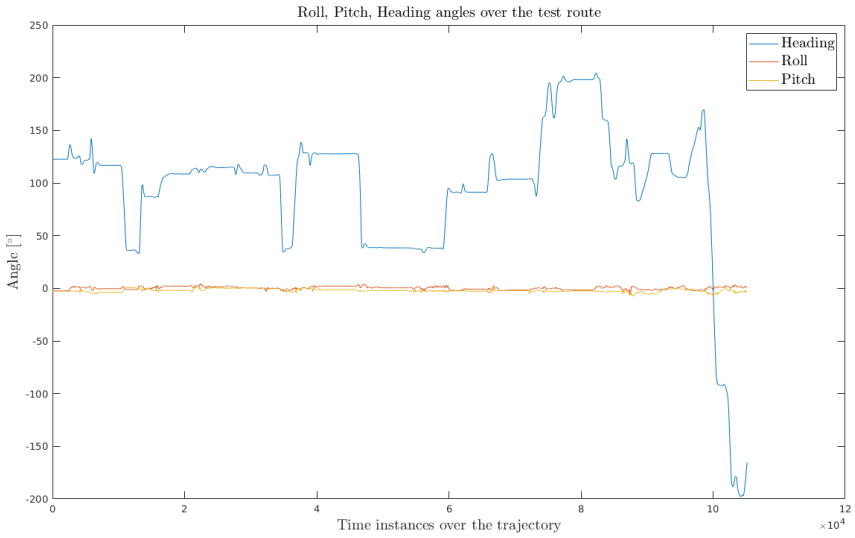


Figure 5.1: Plot of the heading, roll and pitch angles of the car during a drive on the route between Johanneberg and Lindholmen. The heading angle is the only angle that changes notably throughout the route which justifies the use of a planar model for EGO-motion rectification over the route.

As can be seen from examining Figure 5.1, the heading angle is the only angle that changes notably throughout the route while the roll and pitch angles are centered around 0 with some small deviations. This corresponds with the vehicle's movement being mostly planar throughout the route and justifies the use of a planar model for EGO-motion rectification over the route. In the planar model the reference frame of the object is two dimensional and the object can freely move in the reference plane and rotate around its own z axis.

The rectification performed on the point cloud is thus not only a linear shift of the points in the driving direction (along the positive x-axis), but rather a compensation in both the x and y-direction. The difference between the original and rectified cloud will increase as linear and angular velocity increase, meaning that EGO-motion rectification becomes even more important under those driving conditions, further enforcing the suitability of the planar model.

5.1.1 Model of the vehicle's movement during a scan

The vehicle's motion can be defined according to two coordinate systems, the global (world) coordinate system and the local (vehicle) coordinate system. The vehicle is assumed to be travelling in planar motion, parallel to the road surface, within the X-Y plane in the world coordinate system. This simplifies the modelling to consider only the angular change of the *yaw (heading)* angle. A visualization of the planar motion is presented in Figure 5.2.

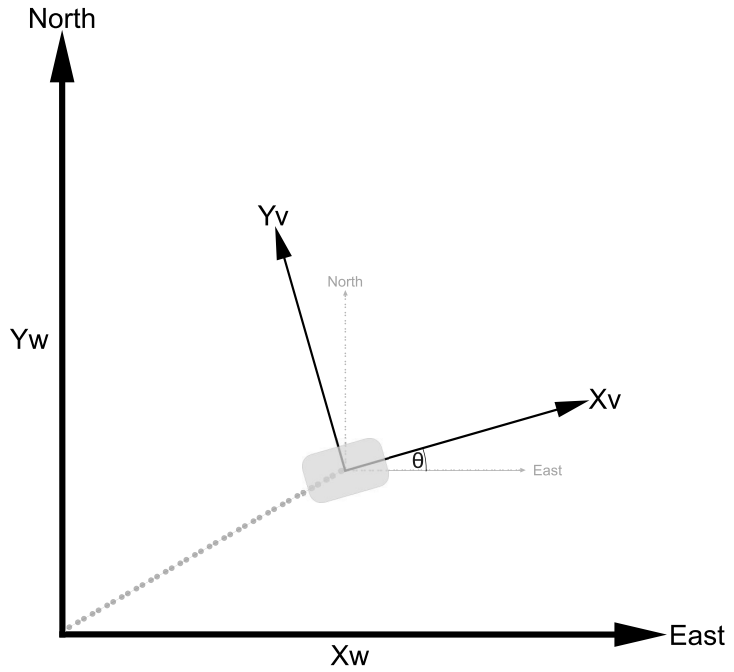


Figure 5.2: Orientation of world frame (X_W, Y_W) and vehicle frame (X_V, Y_V) as seen from above along the Z axes. Angle θ is the heading of the vehicle

The world frame is defined according to the Geographic coordinate system, meaning position is described according to coordinates in the North and East directions.
The trajectory travelled by the vehicle during the 0.1 second scan duration needs to be accounted for, since the origin of the LiDAR cloud is assumed stationary in its local frame, but in actuality moves throughout the scan duration within the world coordinate system. The distance Δx is the change in location measured at the beginning and the end of the scan. However, this might not necessarily be a straight line.
The body-frame of the vehicle is depicted in Figure 5.2. Its coordinate system consists of a Cartesian 3D system, of X_V, Y_V and Z_V directions. The X_V-axis is fixed to the vehicle facing forwards in the driving direction, the Y_V-axis denotes motion perpendicular to the driving direction while the Z_V-axis denotes any changes in elevation with respect to the road, which is considered constant since motion in this direction is small compared to planar motion in the $X_V - Y_V$ plane.
The velocity in the world X-Y plane is computed as the Euclidian norm between the velocityEast and velocityNorth measurement from the IMU/GPS

$$v = \sqrt{v_{East}^2 + v_{North}^2} \tag{5.1}$$

The Linear motion component Δx is then estimated as:

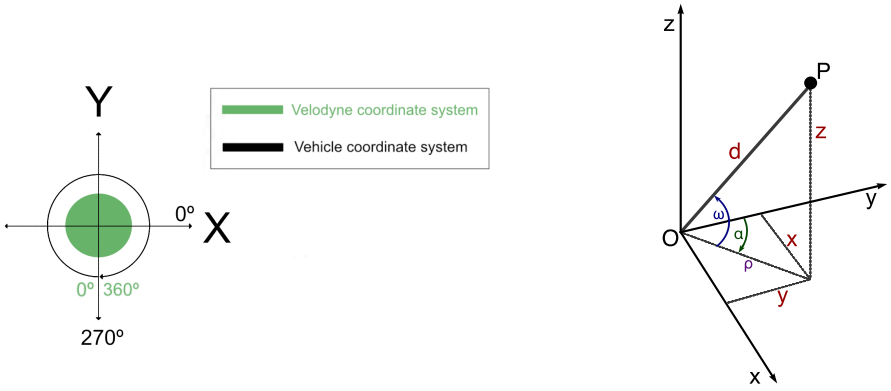
$$\Delta x = v \cdot t \tag{5.2}$$

where v is the speed measured vehicle velocity in the vehicle's x-direction (driving direction) during the scan. t is the time duration of the interval. The angular velocity

$\dot{\theta}$ was recorded as the IMU measured angular velocity around the downward z-axis, and is used to compensate for the non-linear trajectory section to model turns. Since the z-axis in the IMU is positive downwards, it means that a positive turn around the negative z axis is clockwise. The z-axis in the EGO-motion model is positive upwards which requires the measurements to be negated to represent a counterclockwise positive angular velocity around the upward z axis.

5.1.2 Correcting the measurements (update)

The LiDAR axes align with the vehicle coordinate frame, although a LiDAR scan itself begins at 270° (or −90°), as shown in Figure 5.3(a). Based on the motion of the vehicle during a scan, as described in the above section, each point in the point cloud can be shifted to the actual location it represents with respect to the beginning of a single scan. This requires a rotational shift in the EGO-motion processing, since a LiDAR cloud should be rectified in the same sequence as it was recorded, adjusting the points captured last the most. The Velodyne HDL-32E LiDAR measures 32 scanlines near instantaneously, each spaced at differing inclinations, as it sweeps through a 360° rotation to complete one scan.



(a) Alignment of the LiDAR within the vehicle frame. The starting and ending position of the LiDAR scanner is at 270° within the vehicle frame.

(b) Coordinate system of a single LiDAR point P as measured from the optical center of the LiDAR, denoted as O. Angle α denotes the azimuth angle of the LiDAR ray and ω denoted the inclination.

Figure 5.3: Alignment of the LiDAR coordinate system of the vehicle coordinate system (a), and representation of a point within the LiDAR coordinate system.

Each individual point measurement contains an X,Y and Z coordinate with respect to the optical center of the scanner in Figure 5.3(b), a value of the reflectivity of the measured object as well as the scan line azimuth α and inclination ω. This allows for direct application of this data for EGO-motion rectification, without the need

of any geometric preprocessing of individual points to access the aforementioned angles.

Conversely to [10], the LiDAR scan will be corrected from its starting position at $\alpha_{LiDAR} = 0^{\circ}$. Hence, the corrected LiDAR location must be updated with respect to the LiDAR azimuth α and the vehicle odometry for every point. The vehicle's displacement and change in heading at an azimuth α are denoted Δx_α and $\Delta\theta_\alpha$ where Δx_α is the fraction of the total linear displacement moved for each point, according to its azimuth position. Similarly, $\Delta\theta_\alpha$ denotes the fraction of the angular position of the vehicle at the point when α has a specific value. The updated position and orientation of the vehicle at the time instance a point was captured are then:

$$P_\alpha = \begin{bmatrix} X_\alpha \\ Y_\alpha \\ \theta_\alpha \end{bmatrix} = \begin{bmatrix} \Delta x_\alpha cos(\Delta\theta_\alpha) \\ \Delta x_\alpha sin(\theta_\alpha) \\ \Delta\theta_\alpha \end{bmatrix} \tag{5.3}$$

Since the change in LiDAR position has now been computed, the LiDAR point cloud can now be reconstructed to compensate for motion:

$$E_c = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} X_\alpha \\ Y_\alpha \\ 0 \end{bmatrix} + \begin{bmatrix} cos(\theta_\alpha) & -sin(\theta_\alpha) & 0 \\ sin(\theta_\alpha) & cos(\theta_\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{5.4}$$

The rotation matrices and angles must be updated for each new α and θ combination of the LiDAR scan.

5.1.3 Division of a LiDAR scan into time segments using GNSS data

The Applanix GNSS system operates at 100 Hz so during a LiDAR scan it is possible to account for varying speeds instead of a constant one. This allows for better estimation of the EGO-motion by integrating the vehicle trajectory over smaller time instances.

The GNSS measurement which is the closest in time to the LiDAR timestamp can lie at a time instance before or after the start of the Velodyne scan. In practice, the number of measurements obtained from the GNSS system which overlap with the Velodyne scan varies, so the update of the LiDAR clouds cannot be split up into equally large time segments but need to be adjusted for dynamically.

The main idea behind the implementation of the compensation is that a LiDAR point captured early in the scan needs less modification than a point captured late in the scan. The procedure iterates therefore through every point in the point cloud and the point gets shifted appropriately based on the time instance it was captured.

The total linear motion and the total heading at the time a point was captured can be estimated by adding up the linear motion and heading for every time interval. This is described by the following equations

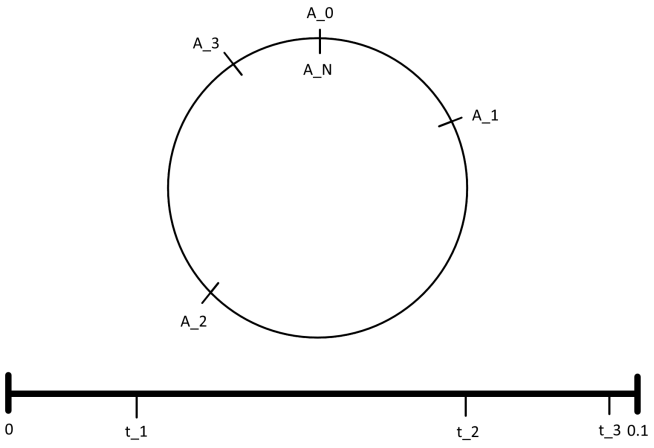


Figure 5.4: The time interval of one LiDAR scan split up into segments. The azimuth angles of the LiDAR corresponding to the start of each interval are denoted $A_1, A_2, ..., A_{N-1}$. Angles A_0 and A_N are always 0 and 2π.

$$\begin{aligned} \Delta x &= \sum_{i=1}^{n} v_i \Delta t_i \\ \Delta \theta &= \sum_{i=1}^{n} \dot{\theta}_i \Delta t_i \end{aligned} \tag{5.5}$$

where Δt_i is the duration of the interval and n is the total number of intervals. This division of a scan into time segments is presented visually in Figure 5.4.

The time intervals can be of varying length (usually around 0.01 seconds each) and when a point is captured within an interval, one needs to compute the fraction of the time spent within this interval instead of using the full length of the interval. Equation (5.5) is thus modified to

$$\begin{aligned} \Delta x_\alpha &= \sum_{i=1}^{n-1} v_i \Delta t_i + f v_n \Delta t_n \\ \Delta \theta_\alpha &= \sum_{i=1}^{n-1} \dot{\theta}_i \Delta t_i + f \dot{\theta}_n \Delta t_n \end{aligned} \tag{5.6}$$

where the fraction f is calculated as the following:

$$f = \frac{\alpha - A_{\Delta,j-1}}{A_{\Delta,j} - A_{\Delta,j-1}} \tag{5.7}$$

where α is the azimuth angle for the point and $A_{\Delta,j}$ is the azimuth angle of the LiDAR at the end of time interval j.

5.2 Object detection and removal

Ghost objects in a LiDAR scan can occur due to recording dynamic objects which are never consistently present at an exact location. Since these objects might not be present at a later stage, problems can arise in matching the two different clouds with the Generalized ICP algorithm. The algorithm will compute false correspondences and attempt to align objects which do not correspond to one another, which causes overall misalignment as the GICP algorithm is likely to converge to a local minimum. In order to mitigate the convergence to local minima due to ghost objects, it is attempted to detect and remove non-stationary object within the local LiDAR point cloud. With fewer objects present in the cloud, the GICP algorithm cannot attempt to align these objects falsely, which in theory results in a more consistent and stable alignment.

5.2.1 Dynamic object selection in images

In order to attempt to remove dynamic objects, an object detection algorithm is implemented to detect these objects in camera images captured at a similar time instance to the point cloud. Since object detection in point clouds is at this time not as thoroughly developed as object detection in images, a detection algorithm for RGB images was used.
The Basler camera is pointing in the driving direction and provides images at a rate of 10Hz, as this is the current limit for stable operation on the test vehicle. These images are passed through the neural network, which was set up and trained to detect a certain number of objects commonly found in an urban traffic environment.

Object detection

The initial approach to detecting ghost objects within images was the use of an object detection algorithm. The *yolov3* [11] network with pretrained network weights was applied for this task. General performance was considered good, with accurate and fast detections made of the image scenes. The fact that object detection does not select an object pixel-wise, but rather contains an object within a predicted bounding box, was not an issue since this simply meant a larger removal of points. In certain instances of fast-moving objects, a larger boundary for point removal can even be beneficial to compensate for the misalignment in the image time stamp and the point cloud timestamp. A larger issue, however, was the fact that the pretrained weights did not account for buses or trams, which appear as large planar objects in a point cloud and therefore can more easily cause planar misalignment.
Due to the limited time period of this project, it was decided against training the network to detect further classes and rather to apply a different network, as described in the Section below.

Semantic segmentation

The semantic segmentation network used in the thesis was chosen because it can also detect buses and trams. In traffic situations, these vehicles cause large planar

surfaces to be detected in the 3D space which can cause the GICP to converge to a local minimum. The network used was implemented by Måns Larsson [12], based on the PSPNet [13] network structure. The network was trained on the Cityscapes [14] and Mapillary Vistas [15] data set, as well as the CMU and RobotCar cross-season correspondence data sets, as mentioned by Larsson et.al. The categories contained in the network are common traffic orientated categories, such as *person, car, bus, bicycle, on-rails* etc. The category ID assigned to each pixel by the network allows for easy addressing of areas within an image, by assessing the pixel ID individually. This creates a mask with a tighter boundary of detecting and classifying individual objects than regular bounding boxes would.

5.2.2 Removal of 3D points based on image detections

With results from the camera calibration, the 3D points can be projected into the image plane. This is done by applying the camera equation 3.18, so the 2D projections of the 3D LiDAR points can be computed. The position of a 2D projections in the image plane now corresponds to the location of the objects depicted in the images.
Since each pixel in an image is categorized, these values can be used to construct a pixel map of where objects have been detected in the image. This is done by firstly constructing a vector with all desired object categories, which should be detected for removal in an image. Then a mask of detected objects is created, which covers the entire image. A '1' corresponds to a detected object and a '0' otherwise. This procedure is visualized later in the Validation Section 7.2 (Figure 7.4).
For the point cloud, first a Field of View (FOV) section of the LiDAR Cloud is selected, which corresponds to the image FOV. This section ranges from azimuth angles of 210° to 305°. These 3D points are then projected into the image plane are then compared to the pixel mask. If a projected LiDAR point lies over, or within a radius of 5 pixels of a dynamic object in the mask, this point is indexed for removal. A pixel radius of 5 pixels is chosen in order to account for minor misalignment in the camera calibration as well as the difference in time between an image and the point cloud. It is beneficial to rather remove a few points too many than to have segments of the ghost object remain in the point cloud. The 2D projections which have been selected for removal have their point indices extracted and applied to the 3D FOV cloud. The selected indices are then discarded from the point cloud before the FOV section is rejoined with the rest of the point cloud. Dynamic objects in the front facing direction have now been removed, whilst objects outside of the FOV remain unaffected.

5.3 Alignment and Merging of local clouds

In order to reinforce the planes and edges, and in general build up a denser representation of surfaces of the structures captured in the local point clouds, several local clouds are merged together. This requires alignment according to the difference in position and orientation between individual local scans, which then merges neighbouring point clouds into a denser 3D representation.

5.3.1 Structural preprocessing for registration

Merging local point clouds is the last point cloud preprocessing step before GICP registration to the dense point cloud is performed. Due to the high computational cost of the GICP algorithm, the number of dense point clouds created in this step are spaced at a distance of 4m apart. This spacing still sufficiently covers the entire test trajectory, yet reduces computational cost greatly due to the reduction of the number of local dense clouds outputted. By selecting clouds every 4 meters, no (near) duplicate clouds are created when stationary, which is a common occurrence in a traffic scenario.
The next step of preprocessing is the assignment of the nearest neighbouring clouds. It was chosen to create point clouds consisting of $N = 9$ local scans, which are structured around one center cloud (the outputted, dense representation assigned to its respective GPS location) merged with $\frac{N-1}{2} = 4$ clouds positioned before and after the center cloud. Nine clouds were chosen as this represents a satisfactory build-up of detail in the merged clouds, but is not overly dense which would only add computational cost the GICP algorithm. The time difference between consecutive neighbouring clouds is assessed, and if this value falls outside 0.1 ± 0.05 seconds, the clouds are discarded and a new center cloud is chosen. This assures a dense 3D build and reduces the risk of outlier clouds which might have faulty trajectory data assigned to them due to a delay in the sensor output.

5.3.2 Registration based on IMU data

Registration based on IMU data does not rely on optimization or iterative convergence of aligning two or more point clouds like the ICP algorithm, but combines several LiDAR clouds based on their respective position and orientation obtained from the Applanix GNSS and IMU system. As mentioned above, first a center cloud is chosen which is the basis of the registration since it determines the timestamp of the registered clouds, and hence defines the exact position and orientation associated to the output. The specified number of neighbouring clouds are selected, and one at a time is aligned to the main cloud.
This alignment is done by compensating first for the rotational difference between the two clouds and then for location shift. The rotational difference $\Delta\theta$ in heading is computed as

$$\Delta\theta = -(\theta_{main} - \theta_{neighbour}) \tag{5.8}$$

Using $\Delta\theta$ the neighbouring cloud $P_{neighbour}$ can aligned to the center cloud by applying a rotation matrix R_z around the positive z axis. The rotation matrix R_z is defined as

$$R_z = \begin{bmatrix} \cos\Delta\theta & -\sin\Delta\theta & 0 \\ \sin\Delta\theta & \cos\Delta\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.9}$$

and the 3D points are transformed by

$$P_{rot} = R_z \times P_{neighbour} \tag{5.10}$$

It is to be noted that the planar model is used once again, which for this case is a more coarse approximation compared to using the planar model in EGO-motion

rectification. This is due to the fact that the distances considered in merging several clouds are much larger, and it becomes more likely that the road surface inclination can change. Going back to Figure 5.1 one can see that the heading and pitch angles do not change drastically but could still affect the outcome of the registration. Some minor discrepancies can be removed by compensating for all three angles. This should be a simple adjustment, as only the rotation matrix R_z needs to be altered to incorporate a full rotation compensation in all directions.

To perform the location compensation, the total distance travelled for each cloud needs to be computed. To obtain distance travelled, the sum of the timesteps between the neighbouring cloud and the main cloud is computed by multiplying the velocity over the time intervals with the time interval itself. Although the resulting velocity in the driving direction is required, the Applanix system records velocity in the North and East direction. The distance D travelled in these respective directions is then first computed

$$\begin{aligned} D_N &= [\Delta t] \times [v_N]' \\ D_E &= [\Delta t] \times [v_E]' \end{aligned} \tag{5.11}$$

where Δt is an array containing the difference between consecutive timestamps, and v_N and v_E are the velocities over these time intervals in the North and East direction respectively. To compensate for the clouds furthest away from the center cloud, larger time intervals need to be multiplied with their respective velocities, which increases the possibility for numeric integration drift, as tiny errors and approximations of the velocities sum together to form a larger error term. This is another reason why it was chosen to limit the number of local clouds to be merged together, in order to not compromise the accuracy of aligned surfaces.
The total distance travelled in the driving direction can then be computed by taking the norm

$$D = \sqrt{D_N^2 + D_E^2} \tag{5.12}$$

Now the compensation in the x, y and z directions can be obtained with the use of the change in heading $\Delta\theta$ computed earlier. A distinction in compensation needs to be made whether the neighbouring cloud is located before or after the main cloud. For neighbouring clouds before the main cloud, the change in location is calculated as

$$\begin{aligned} \Delta x &= -D \cos \Delta\theta \\ \Delta y &= -D \sin \Delta\theta \\ \Delta z &= 0 \end{aligned} \tag{5.13}$$

and for neighbouring clouds after the main cloud

$$\begin{aligned} \Delta x &= D \cos \Delta\theta \\ \Delta y &= D \sin \Delta\theta \\ \Delta z &= 0 \end{aligned} \tag{5.14}$$

Note that a simplification is made by assuming that the z coordinate has not drastically shifted in height.

Finally, the position compensation can be added to the already rotated point cloud P_{rot} to obtain the final aligned neighbouring cloud

$$P_{aligned} = P_{rot} + \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{5.15}$$

Now that a neighbouring cloud has been aligned to the main cloud, it can simply be concatenated to the main cloud, which, after performed for all 9 clouds, yields the final denser, merged output cloud.

6
Localization

At this stage in the algorithm, the local LiDAR clouds have been processed and aligned and are ready for registration by the GICP. The GICP algorithm used is a C++ implementation from the open source *Point Cloud Library* package, the code for loading and inputting the clouds into the GICP algorithm was written by Gunnar Bolmwall and Måns Östman as a part of their work in [1]. After the local clouds have been registered to the dense cloud with the GICP the bias between the GICP outputs and the GPS trajectory are smoothed with an RTS smoother. The smoothing process is performed in **MATLAB**.

The aim of the localization is to accurately estimate the full six-dimensional *pose* of the vehicle which is the three-dimensional position in x, y, z and the Euler angles of roll, pitch and yaw who are measured in the body frame of the car. The state space formulation for the pose is therefore

$$\mathbf{x} = \begin{bmatrix} x & y & z & \phi & \psi & \theta \end{bmatrix}^T \tag{6.1}$$

Every local cloud has an attached pose $\mathbf{x}_{GPS}$ which the algorithm seeks to improve.

6.1 Downsampling of point clouds and selection of corresponding dense clouds

The GICP algorithm is computationally expensive and scales in relation to the number of points sent into the algorithm. This is why both the dense reference point cloud, as well as the merged local clouds, are downsampled before being processed by the GICP algorithm. The downsampling is done on a voxel-based method, which divides the point cloud into $\delta \times \delta \times \delta$ sized grid boxes (voxels). The δ is a parameter chosen relative to the point being downsampled. Points that fall within a grid box are merged by averaging the point's positions, resulting in one point for each grid box. This allows for even distribution of points without compromising loss of features represented in the point cloud. This is performed in **MATLAB** using $\delta = 10cm$ for the local clouds and $\delta = 25cm$ for the dense cloud. The local clouds don't have as clear a structure as the dense cloud which is why the delta is chosen lower for the local clouds than the dense one.
The parts of the dense cloud which correspond to each local cloud are selected from the pose of the Velodyne sensor at the time instance associated to the local cloud.

6.2 Initial alignment

The origin of the source cloud coordinate system is the LiDAR centre, while the coordinate system of the dense target cloud corresponds to GPS coordinates. To get an initial alignment of the two clouds the target cloud is translated to the GPS coordinates of the source cloud. Next, the source cloud is aligned to the world frame by a rotating it around the roll, pitch and yaw angles corresponding to the cloud. This aligns the two clouds accurate up to the precision of the Applanix.

6.3 Denoising operations

The PCL library offers some methods to further reduce the number of points in the point clouds. Firstly all points in the source cloud that are closer than 1.5 meters to the origin or further away than 60 meters are removed. Points within a 1.5 meter radius of the origin correspond to points on, or inside, the vehicle and are therefore most likely faulty measurements. The upper limit of 60 meters is chosen due to the fact that far away from the center the points become sparse and have little or unclear structure, adding little value to the point cloud. Secondly, points in the target cloud further than 70 meters away from the origin are discarded. This is simply due to the fact that points above 60 meters are removed from the source cloud so the points removed from the target cloud have a small chance of corresponding to any remaining points in the source cloud. The source cloud is filtered with a *k-nearest neighbour* filter to denoise it even further. This filtering operation removes points which have less than 20 neighbouring points within a 1 meter radius. These thresholds are chosen somewhat arbitrarily but seem to remove unexpected clusters of points that tend to appear below ground or in thin air. Different from the previous work in [1], no height filtering is performed on the target cloud.

6.4 GICP

The fully processed source and target clouds are sent to the GICP which calculates the transformation aligning the source to the target. The GICP does not subsample the clouds in a RANSAC manner but runs an inner loop of optimization, starting from the initial alignment. The GICP runs the optimization until it has converged or reaches 50 iterations.
The transformation matrix which the GICP outputs is an affine 4×4 transformation, denoted T_{GICP}, in the global frame that describes the rotation and translation needed to align the local cloud from it's initial alignment to the dense one.
Apart from the transformation, the algorithm outputs *fitness score*, which is an estimate of how well the resulting transformation aligns the source to the target. The fitness score is the squared Euclidean distance of each point in the source cloud to its nearest neighbour in the target:

$$f = \sum_{i=1}^{n}(||(p_{\text{i, source}} - p_{\text{i, target}}||)^2 \tag{6.2}$$

The fitness score calculation in Equation (6.2) gives only an estimate of the quality of the transformation but can be used to distinguish good alignments from bad ones as the accuracy in the GICP alignment is somewhat proportional to the fitness score. Alignments where the fitness score is low, are more likely to be better than alignments with high fitness score. As can be seen from Equation (6.2), the fitness score computation can be highly reliant on the number of points within the source point cloud. This is not an ideal trait of the fitness score, but it did not prove to be a considerable factor in this work since the number of points within the merged local point clouds does not deviate too much. How this can affect the fitness score itself and how it is handled will be described in the section below.

6.5 Updating the position measurements with GICP outputs

The GICP output contains information about $\mathbf{x}_{GICP}$, the position to which the GICP algorithm converged for each individual point cloud. In addition to this, a fitness score is associated with each GICP output which provides an assumption of how close the GICP was able to align the source and target clouds.
An outlier threshold can be added to the fitness score calculations in equation (6.2), so that points in the source cloud with no neighbouring target cloud points within a certain radius are left out of the score. A fitness score radius of 5m was selected, in order to minimize the weight which outlier points add to the overall metric, as points with no nearest neighbour within 5m are discarded. This is helpful to alleviate the effects from scanned areas or larger objects present in the source cloud but not present in the dense point cloud. An example of this is the construction site of the new Götaälvbron not present in the dense cloud but captured by the LiDAR driving across the bridge. A Fitness score output of the GICP, evaluated for points within a 5m range (fS5) is outputted as the alignment evaluation metric.
By manual inspection boundary values on the fS5 were selected to classify the GICP alignment as *Good, Ok* or *Bad*. The selected boundaries for the fS5 fitness score categorization are shown in Table 6.1 below

Fitness score fS5 boundaries for GICP	
Rating	Score range
Good	0<fS5<0.33
Ok	0.33<fS5<0.6
Bad	fS5>0.6

Table 6.1: Fitness score boundaries into which the fitness scores are categorized for *Good, Ok* and *Bad* performance.

The section over Götaälvsbron is set with an alignment value of *Good*, regardless of its fS5 score. The reason for this is that the initial position estimate is very accurate in the open area, and the GICP aligns extremely well, although buildings captured beside the bridge are not always present in the dense cloud, raising the fS5 score wrongly.

Based on the fS5 categorization, standard deviations are assigned to each fitness score in an increasing manner. Each section (*Good, Ok, Bad*) contains such an assignment, whose value grows linearly as the fitness score increases. This defines how sure the algorithm is about a specific GICP alignment being correct. The formula describing the standard deviation σ assignment for each section is

$$\sigma = \sigma_{min} + (\sigma_{max} - \sigma_{min})\frac{fS5}{\Delta} \tag{6.3}$$

where the Δ is the upper limit of the fitness score sections, 0.33, 0.6 and 1 for the last section.
The standard deviation ranges are shown in Table 6.2,

Standard deviation limit assignment		
Rating	σ_{min}	σ_{max}
Good	$\begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}^T$	$\begin{bmatrix}0.1 & 0.1 & 0.05 & 0.1 & 0.1 & 0.1\end{bmatrix}^T$
Ok	$\begin{bmatrix}0.5 & 0.5 & 0.05 & 0 & 0 & 0\end{bmatrix}^T$	$\begin{bmatrix}1.5 & 1.5 & 0.1 & 0.1 & 0.1 & 0.1\end{bmatrix}^T$
Bad	$\begin{bmatrix}1.5 & 1.5 & 0.05 & 0 & 0 & 0\end{bmatrix}^T$	$\begin{bmatrix}3 & 3 & 0.05 & 0.1 & 0.1 & 0.1\end{bmatrix}^T$

Table 6.2: Standard deviation limits into which the fS5 fitness score is categorized. Within each of these ranges a linear assignment of covariances is performed according to the fitness score. The units for the standard deviation terms of elements 1-3 are in meters, whilst elements 4-6 are in degrees, corresponding to the definition of the states $\mathbf{x}$.

The standard deviation values are then squared, and shaped into a diagonal matrix, to obtain the covariance matrix R_k representing the uncertainty about a specific alignment. The clouds have been aligned to the center of the source cloud and the Euler angles are measured in the car's body frame, thus the GICP results are transformed into the correct coordinate systems by

$$T = t_0 T_{GICP} R_0 \tag{6.4}$$

where t_0 is a translation back to the GPS coordinates of the source cloud and R_0 the rotation matrix aligning the source cloud to the world frame. This yields an estimate of where the GICP algorithm thinks the vehicle is located at each instance of a LiDAR scan as the resulting GICP pose $\mathbf{x}_{GICP}$ is the translation of the matrix T and the Euler angles are obtained from the rotation matrix in $T_{1:3,1:3}$.
By these, computations a measurement set is constructed based on the GICP output containing state estimates for the vehicle's orientation and location with measurement covariances assigned to it.

6.6 Trajectory smoothing

In order to update the GPS measurements with the GICP 'measurement' estimates, the bias between the GICP states and GPS states is smoothed with a linear RTS

smoother. This bias $\mathbf{Y}$ is constructed as the difference between states

$$\mathbf{Y} = \mathbf{x}_{GICP} - \mathbf{x}_{GPS} \tag{6.5}$$

This is done for the instances of the trajectory where a GICP output was obtained and the rest of the trajectory is discarded. The reason the bias $\mathbf{Y}$ is smoothed instead of the trajectory is to compute the difference between the two trajectories (bias) and not to distort or adjust the trajectories themselves too much. Once the smoothed bias has been computed, it is added to the GPS trajectory,

$$\mathbf{x}_{smoothed} = \mathbf{x}_{GPS} + \mathbf{Y} \tag{6.6}$$

The RTS smoother is a linear smoother whose structure is described in the Theory Section 3.9. The random walk model was chosen to present the prediction step since this ensures the bias to change gradually over time, uncorrelated with the vehicle's motion. Due to the random walk model deployed, for the smoothing equations 3.78 the process model A_k is a 6×6 identity matrix. The process noise Q_k is computed as

$$Q_k = \begin{bmatrix} r \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix} r^T & \begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix} \\ \begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix} & \begin{matrix} 0.001 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0.001 \end{matrix} \end{bmatrix} \tag{6.7}$$

where r is a rotation matrix indicating the vehicle pose, specific to a trajectory instance. The covariances are thereby applied to account for uncertainty in the respective x, y and z values. The rotation by r should not, in theory, result in any difference, but the structure was kept since initially the variance in z was assigned a lower value. The update step is performed with a measurement model H being a 6×6 identity matrix (due to the Random Walk model), and the noise covariance R_k being constructed manually as described above in Table 6.2. The prior state for the smoother is selected as the first bias term between $\mathbf{x}_{GICP} - \mathbf{x}_{GPS}$, whilst the prior covariance is the assigned according to the initial state's fitness vector, as described above.
The smoother returns the state estimate $\mathbf{x}_s$ and covariance $\mathbf{P}_s$ for each step in the trajectory. These values are used to construct a smoothed trajectory which serves as input to the second pass of the GICP algorithm.

6.7 Run GICP on smoothed path

The smoothed trajectory has estimated the bias in the GPS measurements and updated the trajectory with new position and orientation updates. These represent a more accurate estimation of the vehicle's path, which will be used as the input states to the second GICP smoother. Since the performance of the GICP algorithm is strongly affected by the initial alignment, an improved starting position is expected to mitigate convergence to local minima and instead find the true alignment. For

the initial GICP computation, some instances failed to converge correctly, whilst some managed to find the true alignment to the reference cloud. For computational purposes therefore, only the point clouds with a fitness vector of $fS5 > 0.33$ are passed through the GICP again, with updated starting positions. The improvement in alignment after the first GICP to the second deployment of it, as approximated by fS5 fitness scores, is shown in Figure 6.1

Figure 6.1: The resulting alignment between the first and second GICP passes. The blue dots represent the fitness scores for an alignment at a specific time instance for the initial GICP pass. The trajectory is then smoothed to obtain better initial positions and these are used as inputs for the second GICP pass (orange). The alignment in some sections improves drastically as the smoothing estimates the bias well and compensates for it.

The GICP output files are read for the second pass, and its values are stored in a data structure. The same assignment of covariances are used as for the first smoothing iteration, the only difference being that the updated position should have better state accuracy and thus get assigned lower covariances. In order to avoid any worsening of GICP localizations, the fitness score between the first and second pass of the GICP are compared for each location, and only the best result is kept.

6.8 Final trajectory estimate

After having read the localization results obtained from the second pass of the GICP a final trajectory gets estimated by smoothing the bias between a combined GICP trajectory and the GPS measurements. This combined GICP trajectory is formed by updating the unsmoothed GICP measurements from the first run along with the assigned covariances and updating them with the results of the second GICP pass. This is done in order to get a new estimate of the bias without any effects of the previous smoothing. The bias is then smoothed in the same manner as before, explained in Chapter 6.6. The smoothed bias is then added to the GPS trajectory, yielding the final output estimate for the vehicle states over the trajectory.

7

Validation of point cloud processing

This chapter documents the verification of the preprocessing subsections of the overall algorithm, namely EGO-motion rectification, camera calibration, ghost object removal and local cloud registration. Validation for these subsections is of importance since their individual performance will affect the final accuracy of localizing within the reference point cloud. Further, if a subsection of the algorithm performs inadequately or incorrectly, not only will the final localization be worsened, but accurate analysis of the overall performance becomes distorted, leading to faulty conclusions and recommendations about the project itself.

7.1 Validation of EGO-motion rectification

The first major point cloud processing subsection in the algorithm is EGO-motion rectification. An undistorted 3D representation of the current environment is a crucial starting point for all consequent cloud processing actions. Additionally, a well performing EGO-motion rectification addresses one of the aims of this thesis prescribed in Section 1.2, by addressing poor localization in corners due to an oversimplified EGO-motion model, as suggested by [1].

7.1.1 Test set up

Specific data was collected in order to validate the implemented algorithm for the EGO-motion rectification. To see how accurate and correct the algorithm is, data was collected by driving the car fast and in spinning circles around a fixed point in a closed-off environment. The data collected consisted of both clockwise and counterclockwise turns around the same point. The environment of the test set up contained a lot of fixed objects such as blocks and poles that would be easy to detect in the resulting LiDAR point clouds and help with the verification of the algorithm.

7.1.2 Counterclockwise turns

The first case for validation will analyze the performance of the algorithm for the counterclockwise turns. The initial and transformed cloud can be viewed in a Birds Eye View (BEV) representation in Figure 7.1.

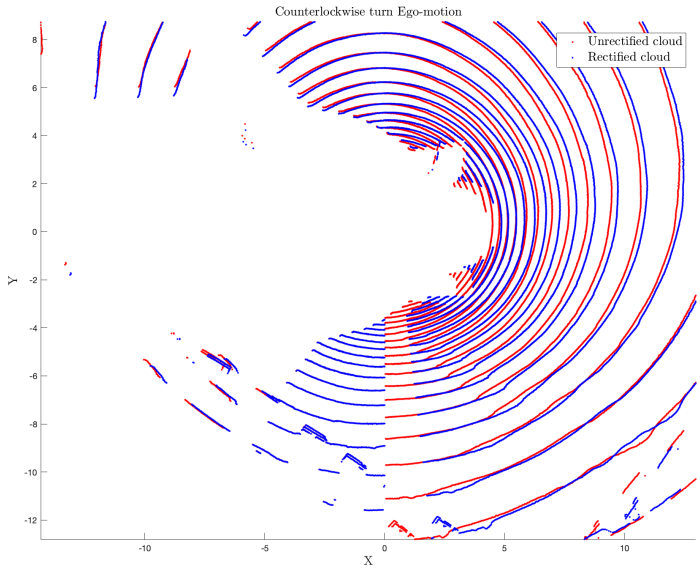


Figure 7.1: Validation of EGO motion during a counterclockwise turn. The car turns with a mean angular rate of -50.2°/s. Due to the counterclockwise motion of the car, the LiDAR is not able to do a full scan of its surroundings, even though it believes it has, represented by the red points. The EGO motion rectification moves points back in the LiDAR angle azimuth, causing the triangular shape in the scan line where no blue (rectified) points are present.

From the analysis of the counterclockwise turn, no certain proof of validation can be drawn, except the analysis of rotational direction. As expected, a counterclockwise vehicle rotation during a LiDAR scan results in a 3D representation which covers less than a full 360° rotation, as shown in Figure 7.1. This occurrence is due to the opposing rotational directions of the LiDAR scanner and the vehicle in this scenario, which verifies that points are incrementally rectified in the correct direction with the correct starting and ending angles within a scan. The counterclockwise example, however, fails to verify whether the amount by which points are being adjusted is correct. For this reason, a clockwise vehicle rotation has to be considered in order to fully validate the EGO-motion rectification.

7.1.3 Clockwise rotations

Figure 7.2 and Figure 7.3 depict the original and rectified clouds for a clockwise vehicle rotation, in a BEV and 3D representation respectively.
As expected, conversely to the counterclockwise turn, the clockwise turn overscans an area and covers more than 360°. This means a single LiDAR scan could contain the same object twice (scanned at 0° and again 360°), and points should be rectified to align these two representations into one. If this can be achieved, the magnitude of the point adjustment can also be verified, which is performed in Figure 7.3 with duplication of a flag pole:

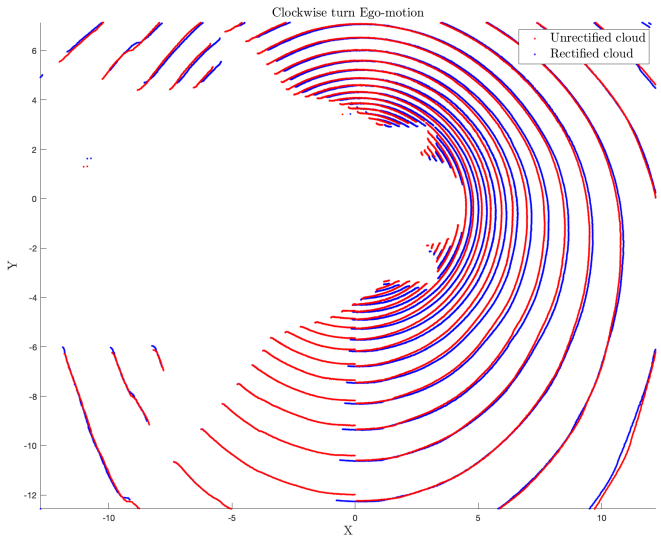


Figure 7.2: Validation of EGO motion during a clockwise turn. The car turns with a mean angular rate of 47.0°/s. The clockwise turn over-scans an area (red), yet saves this data in a 360 ° format. To rectify this scenario, the blue scan (rectified) needs to be adjusted as shown above.

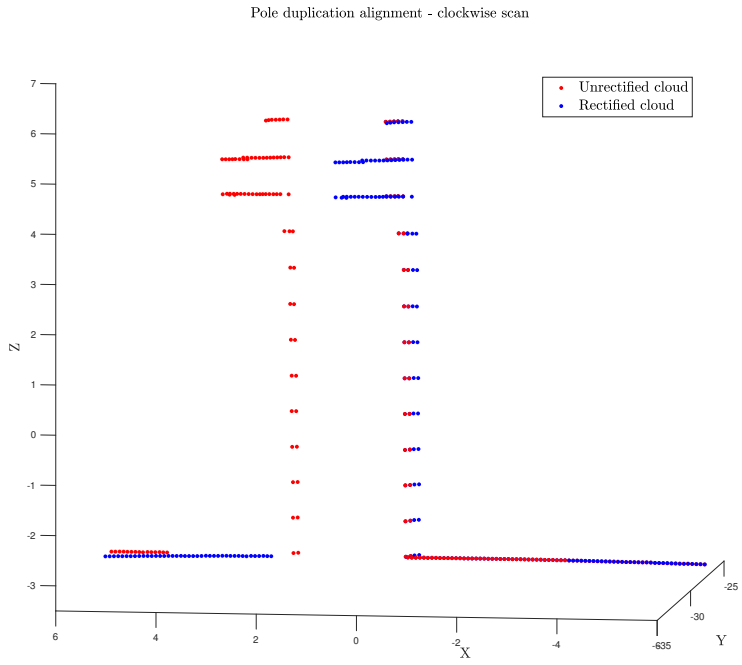


Figure 7.3: Validation of EGO motion. The car turns with a mean angular rate of 47.0°/s. Inspection of the three poles on top of the image verifies the correctness of the EGO motion. In the raw LiDAR cloud there are two poles shown in red. The reason for that being that the LiDAR captures the pole twice due to the circular motion of the car. The rectified cloud shown in blue only includes the pole once since it compensates for the motion of the car and thus removes the double detection by merging the two red poles into one.

The two red poles in Figure 7.3 correspond to the same pole in the test environment that was captured at the beginning and at the end of the LiDAR scan. The EGO-motion algorithm shifts the points captured of the pole at the end of the scan to align with the points of the pole from the start of the scan. The faulty points get shifted on average 2.4 meters in the x direction and 5 centimeters on average in the y direction. After being corrected, the shifted points align well and form the single blue pole which is correctly placed in the cloud.

7.1.4 Conclusion

Since the directions of rectification and also the magnitude of point adjustment have been proven to be implemented correctly, the EGO-motion section is validated.

There are few published papers on EGO-motion rectification so there are no benchmarking tests to compare against here. The algorithm presented in this thesis is based on the work done by Merriaux et al. in [10], as mentioned earlier. The implementation presented here differs from what the one presented in the paper and the sensor data available used in this project differs from the one available in [10]. Due to the lack of common evaluation metric and results it's hard to tell whether the implemented algorithm outperforms the one by Merriaux et al. but there is nothing that indicates that the implemented algorithm performs worse than the one in [10].

7.2 Validation of Ghost removal

Validation of ghost object removal verifies whether the algorithm is able to detect and remove the correct LiDAR points corresponding to dynamic objects. As a starting point, this relies on accurate camera calibration in order to project and align the 3D points of the environment onto the image plane. For accurate camera calibration, the objects found in an image can then be compared to the projected 2D points, to find any points falling within the created object masks. A dynamic traffic scene was chosen to represent the alignment of stationary and moving objects, whilst the ego-vehicle itself is moving. The complexity of the scene is meant to replicate realistic conditions under which the algorithm should perform.

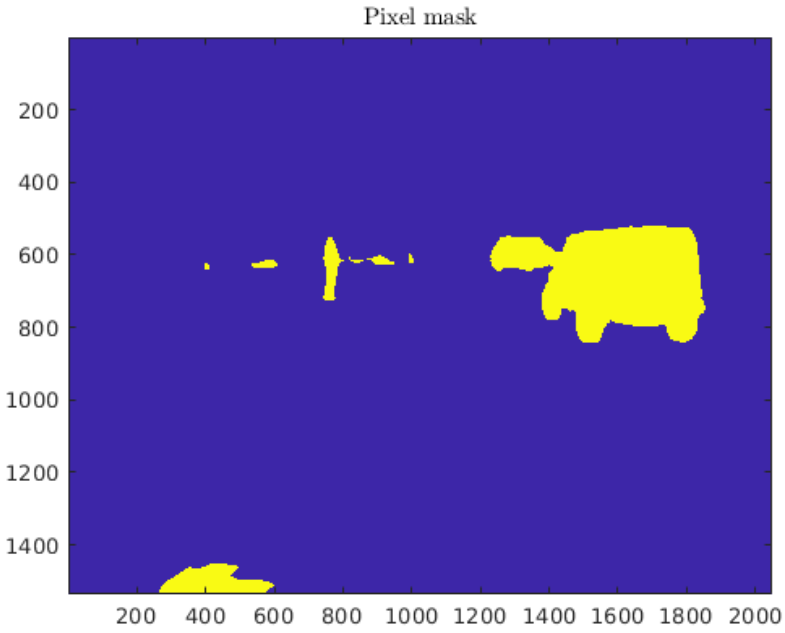


Figure 7.4: A pixel mask of an urban traffic environment is shown in the Figure. Yellow represents objects which fall into any of the semantic segmentation categories, and purple represents the rest of the image. Any projected points falling into the yellow pixel mask will be removed.

According to the pixel mask created from a semantically segmented image, the yellow areas highlight detected objects in the scene whilst purple denotes the remaining pixels. As described in Section 5.2.2, the corresponding LiDAR point cloud is then projected into an image scene, as shown in Figure 7.5

From the projected points it can be seen that the camera calibration and projection align the detected objects fairly close to the outlines of the images. Some slight misalignment is expected due to the small time delay between the capture of the image and the 3D LiDAR cloud. In this delay, discrepancies arise due to the test vehicle's movement as well as the motion of other objects. Adding to this, tiny misalignments in camera calibration can cause the edges not to align perfectly for all images, which is why a 5 pixel boundary is added around each pixel mask, as described in Section 5.2.2. Based on the pixel mask, the projected points which overlay objects are selected for removal, as shown in Figure 7.6.

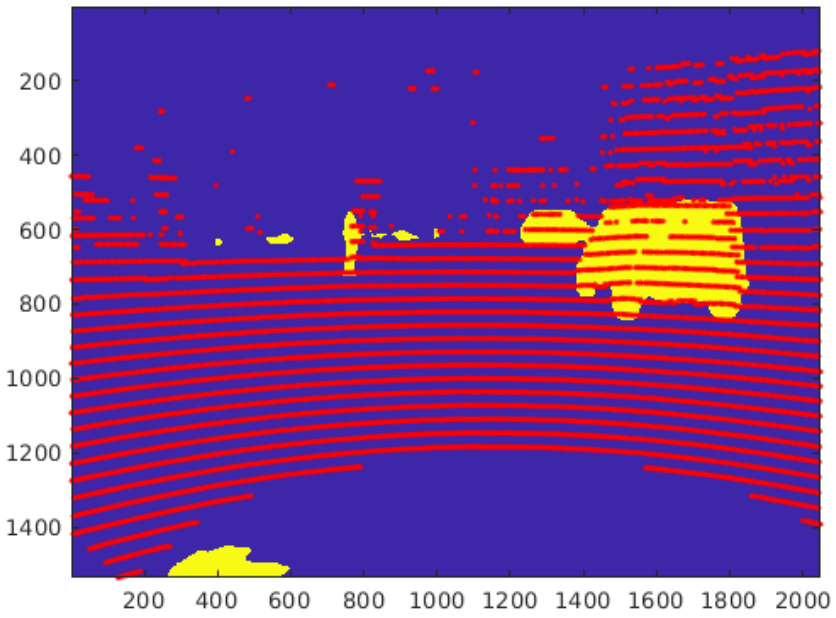


Figure 7.5: An example of a 3D LiDAR point cloud projected onto an image. The outlines of dynamic objects can be seen from their yellow pixel masks, which align to the projected point boundaries within a reasonable margin.

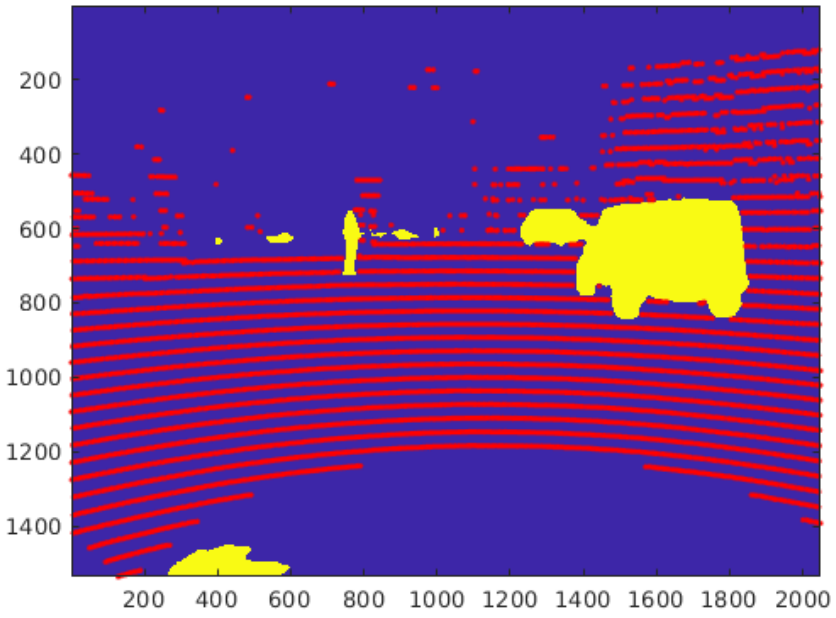


Figure 7.6: By assessment of the pixel mask, the dynamic objects can be selected for removal in the 2D space.

Following a selection of 2D points, their indices need to be transferred and removed

from the 3D point cloud, which should result in a clean and accurate removal of any dynamic objects in the scene. Figure 7.7 shows the detected objects back-projected into 3D, which are set to be removed.

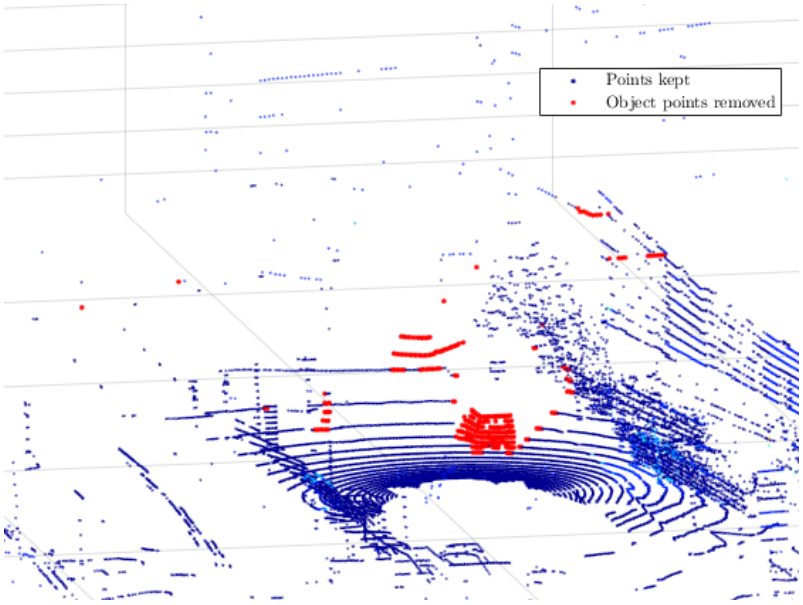


Figure 7.7: Transferring the indices of the selected 2D points back into 3D allows for detection of dynamic objects in 3D. The objects are selected accurately without remaining residuals, whilst some background points are also selected due to small misalignments. This does not, however, pose a problem, as it is better to remove more points than fail to remove ghost object sections.

As can be seen in the above Figure 7.7, the objects (vehicles and person) are at large captured entirely, up to the accuracy of the semantic segmentation detection. This is an important factor since dynamic objects should be removed completely, as it is better to remove too many points than too few. It can be seen that some selected points fall beyond the dynamic objects, as the pixel mask alignment is not perfect. This is no problem however, as the removal of these background points is minimal and does not greatly affect the structure of the shape of the surroundings.

The selected points are then deleted, which results in the final output cloud, depicted in Figure 7.8

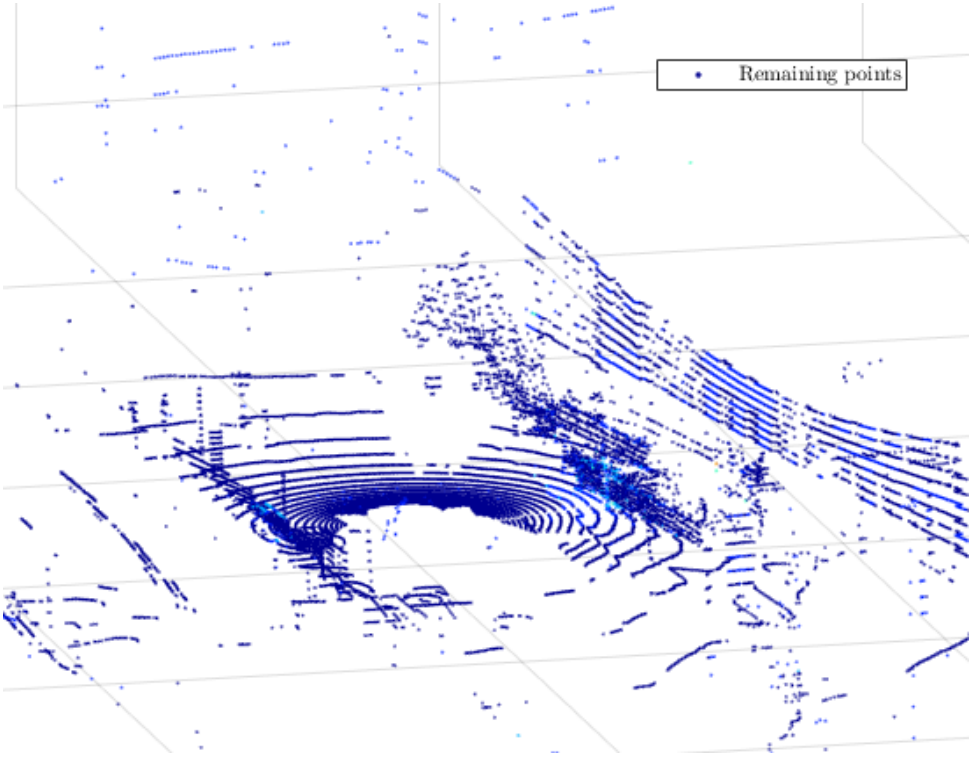


Figure 7.8: The Figure depicts the final output cloud from the object removal algorithm, with ghost objects in the FOV removed.

With ghost objects in the FOV removed, the point clouds are ready to be merged into a denser local point cloud. All dynamic objects which have not been removed (such as behind the vehicle), will likewise be built up into a more prominent 3D object, which is why a clean removal of as many dynamic objects as possible is desired. As was shown in this section, this can be performed with good accuracy, even with slight misalignment in the projection of 3D points to a 2D plane.

Instances where the alignment and removal of dynamic objects might suffer, is at higher speeds. Scenarios susceptible to such misalignment occur when the difference of velocity (and its direction) is large between the EGO-vehicle and the dynamic object. Examples for this could be when the EGO-vehicle is driving past a stationary dynamic object at high speed, or the ego vehicle is stationary and a dynamic object moves past it in a perpendicular of approaching direction. This is due to the detection of dynamic objects being performed in the image plane, so large offsets to pixel positions will cause a poor selection of points to be removed, created by the time delay between image and point cloud capture. An example of such an occurrence is presented in Figure 7.9

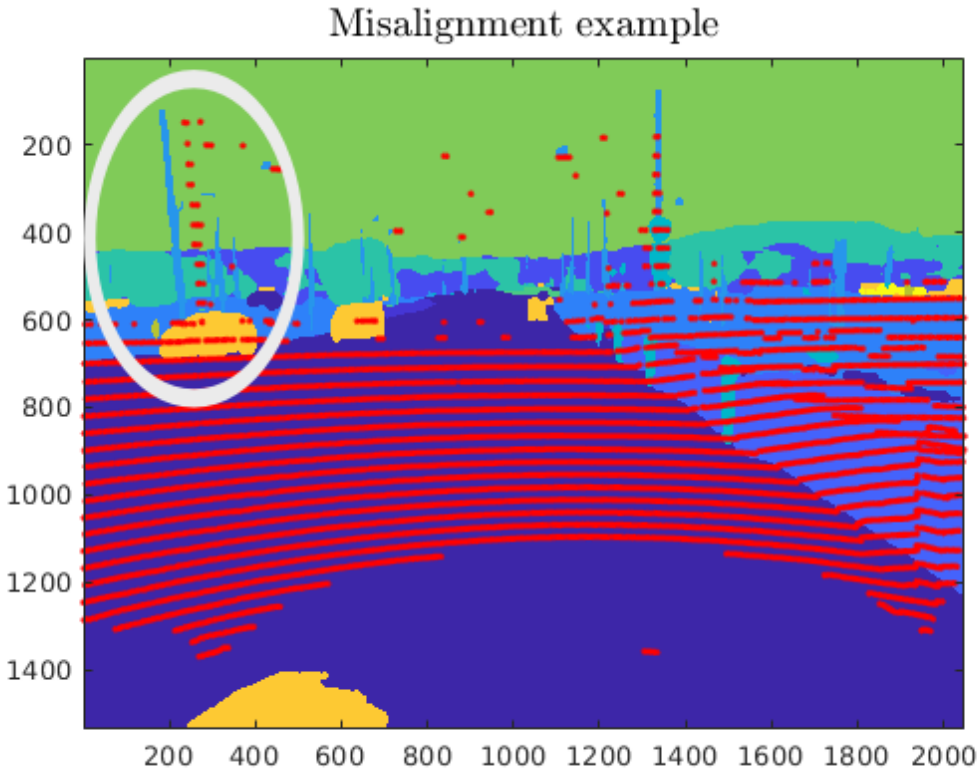


Figure 7.9: A misalignment example is shown of the LiDAR points being projected into the nearest image. As can be seen from the pole misalignment, encircled with a white loop, the possibility for misalignment between the LiDAR and camera data rises as the velocity of the vehicle increases.

7.3 Validation of local cloud registration

The registration of several local clouds aims to create a denser, more defined representation of the features in the current environment. By doing this, surface and edge features are reinforced to improve the GICP performance during the final localization. Due to the accuracy of the GPS/IMU this can be done with fairly high accuracy.

A comparison between a single point cloud and a representation of 9 merged point clouds is shown below. Figure 7.10 depicts the unmerged base cloud, and an example of the IMU merging is presented in Figure 7.11

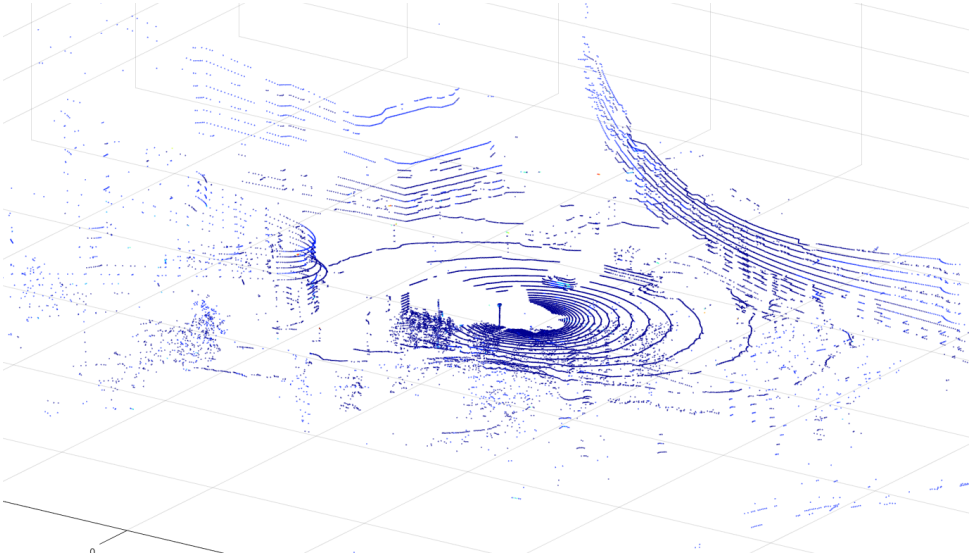


Figure 7.10: A single 32 layer point cloud, which is used as the main cloud for merging multiple clouds. Features and structures are distinguishable, though not densely built up.

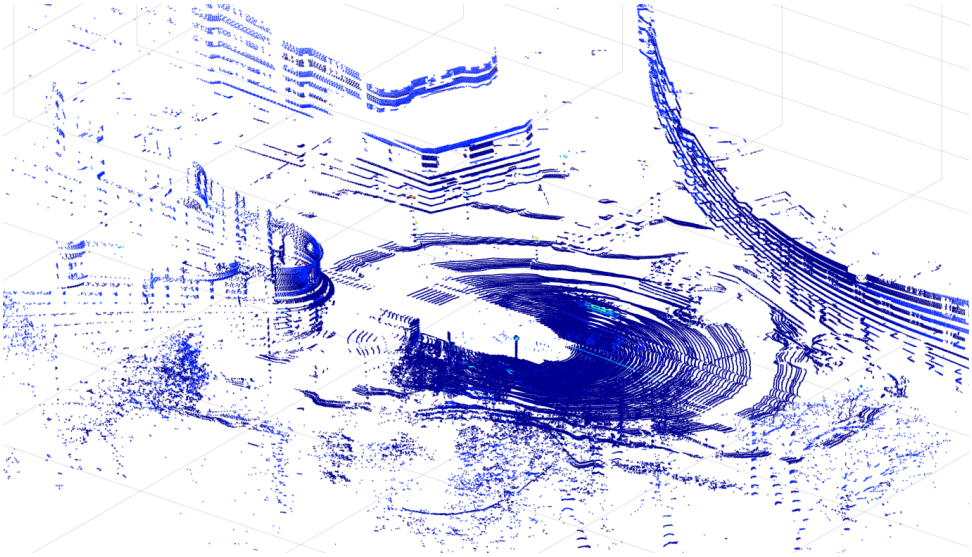


Figure 7.11: An overview depiction of a point cloud reconstruction from 9 single clouds, during a straight driving direction. Surface features of the road, buildings and other objects are reinforced by multiple scan-lines.

From inspection of the single cloud, it can be seen that features and buildings are distinguishable, and a reasonable density of objects can be constructed close to the test vehicle. However, objects further away are sparsely represented, mostly in only a linear manner along a scan line. This is due to the geometry of the angular spacing between the 32 layers of the Velodyne LiDAR. The merged point cloud, by contrast, is a denser reinforcement of surface features, due to the added scan line layers. As mentioned above, this is expected to be of value in the GICP localization.

In the example in Figure 7.11 the vehicle is travelling in a straight line, which results in a simpler, merely linear shift in alignment when merging multiple clouds. In order to inspect the accuracy of the IMU alignment under linear driving conditions, a close-up view of a scene at Vasaplatsen is shown in Figure 7.12, which contains finer features such as poles, traffic signs and a bus stop to be aligned. By inspection of the image, it can be seen that the 9 point clouds are aligned with high accuracy in this case. The various colours in the point cloud represent the intensity of an object's reflectivity, which aids to distinguish the accuracy of the aligned traffic sign, for example.

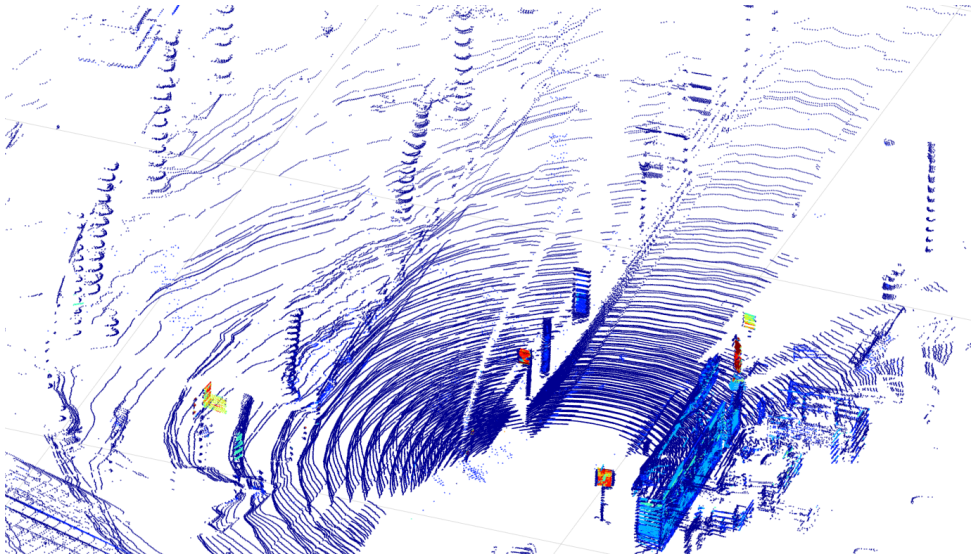
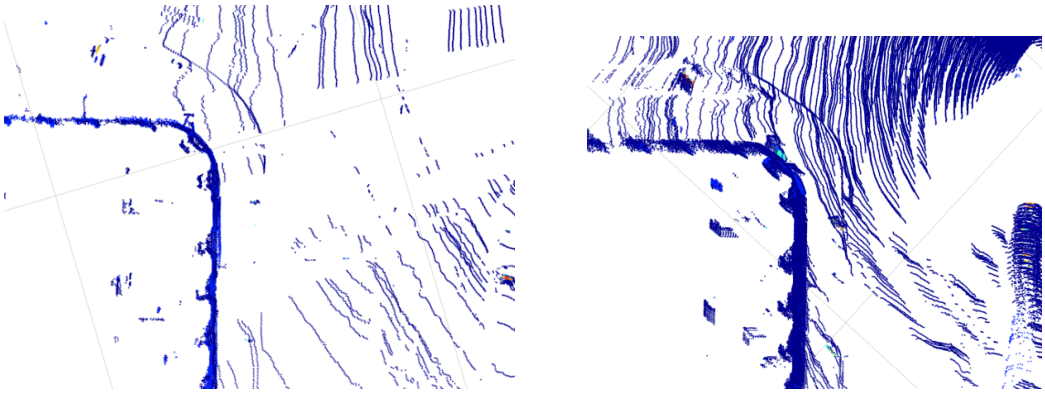


Figure 7.12: A zoomed in scene of a dense point cloud construction of 9 clouds, under a linear driving direction. By inspecting individual objects the accuracy of the alignment can be evaluated, which performs very well yielding clean and dense reconstructions.

Accuracy does, however, suffer slightly during fast turns. Since the angular and linear differences between a center cloud to an edge cloud are computed as a summation of all individual measurements between them, noise in these readings can accumulate and cause slight misalignment. As the distance of objects from the center of the vehicle increases, angular noise is amplified the further away an object is. A comparison of the same building edge is depicted twice in Figure 7.13, where (a) represents an alignment during linear motion and (b) represents an alignment during a turn. Even in the noisy alignment, the merging of 9 point clouds into a single structure still yields distinct features, but is no longer represented by clean surface lines, due to the noise.



(a) Good vertical edge alignment during linear driving conditions.

(b) Noisier vertical edge alignment of alignment during a turn.

Figure 7.13: A comparison of two merged point clouds (from 9 single clouds) of the same building. (a) is captured during linear motion of the test vehicle, which results in simpler alignment and clean surfaces during the merge. (b) was constructed whilst the vehicle was turning, which results in a noisier reconstruction of vertical features, such as the building, and a more dense representation of planar surfaces in the x-y plane, such as the road.

Another observation to be noted is that due to the shift in scan-lines during a turn, a much denser representation of the environment can be built up, as the scan lines are spread out to cover a larger area. This can be advantageous in covering a larger area within the point cloud and should construct surfaces within the x-y plane well (such as the road).
Other noise influences can occur when merging a large number of clouds where the boundary clouds are far from the center cloud, as this alignment also accumulate a small degree of noise. This is partially due to the integration drift and small noise in the sensor readings which accumulates over time and partially due to the timestamp which was used for the Velodyne scan.
A *sampleTimestamp* is saved by the sensor system on-board the test vehicle, which should be the most accurate timestamp available, but due to an error on the sensor system the *sampleTimestamps* are wrongly assigned to the point clouds, mixing up the order of the measurements. For this reason the *sentTimestamps* of the sensor cluster had to be used, which is also fractionally noisier compared to the *sampleTimestamps*, but outputs data in the correct order to be processed. This is an issue on the hardware development side of the test vehicle, and outside the scope of the thesis to be addressed.
Finally, to be taken into consideration is that the merging of several clouds only yields an advantage when the vehicle is moving. During fully stationary conditions the neighbouring clouds are all captured by the same scan-lines in the same places. This means no new data of the environment is captured between scans.

8
Results

In this chapter, the results of the localization are presented and evaluated against a ground truth. The ground truth is explained and the results are analyzed with respect to each of the six states of the pose.

8.1 Ground truth estimation

In order to create a metric to evaluate the performance of the localization, a ground truth value needs to be established. Since no concrete ground truth for the vehicle location exists, an estimate needs to be created based on only the most accurate localizations obtained, discarding all locations of even slight misalignment. A new ground truth needs to be established for each trajectory which the algorithm is to be evaluated for. The procedure for constructing the Ground Truth trajectory is almost identical to the method described above, running the GICP and smoother twice in an interwoven manner. The Velodyne clouds used in the ground truth are chosen so that the ground truth consists of poses obtained from clouds that are not part of the results. After having run the localization algorithm on this new sub-trajectoy, the quality of them is verified by manual inspection. All pose estimates which are not perfectly aligned to the reference cloud are picked out. If the trajectory consisting of the good poses is not dense enough, the bad poses can be passed through GICP again or smoothed in order to increase their accuracy. These refined poses are then manually verified again and all localizations which are not perfectly aligned are discarded. The remaining states have absolute certainty of their position and orientation and are used for ground truth. The ground truth can now be used as a reference to comparing the performance of a full data set which is being localized. It is important to note that fully independent data samples need to be used for ground truth construction. No corresponding point clouds can be present in both the ground truth and the test set, as this might compromise the comparison to a ground truth if the identical value exists in both sets.

8.2 Evaluation metric

The overall localization performance evaluation is based on the accuracy of the algorithm's output trajectory, compared to the ground truth trajectory. The two trajectories are compared at the same time instance, meaning that the value of the output trajectory at a certain timestamp is compared to the ground truth trajectory at the same time instance. Therefore the ground truth needs to be splined as it is

obtained using clouds that are not used for the algorithm results. The splining of the ground truth is done by linearly interpolating the bias of the ground truth and adding that to the original GPS trajectory. That way the splined values of the ground truth are as close as possible to the ground truth.
The positional difference between the validation and Ground truth trajectory is computed as the distance between the respective Longitude (X), Latitude (Y) and Altitude (Z) position estimates. The angular misalignment is computed as the difference in roll, pitch and heading angles between the validation instances and the nearest ground truth instance.
The bias differences to the ground truth trajectory are classified as follows:

Alignment classification categories			
State	Good	Ok	Bad
Latitude	0.1m	0.5m	> 0.5m
Longitude	0.1m	0.5m	> 0.5m
Altitude	0.1m	0.5m	> 0.5m
Heading	1°	3°	$> 3°$
Pitch	1°	3°	$> 3°$
Roll	1°	3°	$> 3°$

Table 8.1: Categorization of the localization accuracy of an instance in the test trajectory against the same instance in the ground truth trajectory.

8.3 Results on a day run in April

The first evaluation of the algorithm's performance is run on a data set collected on the 17th of April in 2019. That day the weather conditions were sunny and a real clear sky with no signs of rain. The foliage on the trees neighbouring the route was similar at that time as it was when the dense map was collected. (To be noted is that the final section of the route is not evaluated, due to no data being collected over the final section. This section of the trajectory is however present in the night run, which is discussed hereafter.)
The Ground Truth consists of 430 state instances, whilst the validation set consists of 317 instances. The Ground truth data set was interpolated to fit a denser representation of points between each state instance, in order to reduce the computed distance of the state comparison. The ground truth trajectory is a smoothed estimate of only perfect point cloud alignment instances, as the best estimate of the location and pose at an instance. Unfortunately, the ground truth construction is not completely dense, so some turns and corners are interpolated roughly, causing some approximations in the trajectory itself.
In order to obtain numerical accuracy estimations, a smoothed trajectory of data points will be compared to a ground truth trajectory estimation. As described in Section 8.1, the ground truth estimation is a smoothed trajectory constructed from only accurate GICP alignment positions. The classification result of the localization compared to the ground truth trajectory is shown in Figure 8.1.

Figure 8.1: Day run: Performance evaluation of the classification of alignment instances over the trajectory. Green indicates *Good* performance, blue indicates *Ok* and red indicates *Bad* alignment. A section of the trajectory is greyed out, which is not evaluated on localization performance due to the lack of a concrete ground in this area.

Results	
Classification	Inlier %
Good	74.31%
Ok	22.57%
Bad	3.12%

Table 8.2: Results for the day run classified as Good, Bad or Ok

Figure 8.1 uses green markers to represent the *Good* localization instances, blue to represent *Ok* localization and red markers indicate *Bad localization*. A section of the trajectory is greyed out, which is a part of the route where ground truth could not be obtained with absolute certainty. This is due to the effects of scaffolding on the building facades, which are not present in the dense point cloud and cause the GICP algorithm to fail. Due to the lack of a concrete ground truth, the algorithm will not be evaluated over this section. To further provide an insight into the localization accuracy, the distribution differences between each of the 6 states and the ground truth states are plotted in histograms in Figure 8.2 to Figure 8.7. The locations are evaluated in their global Latitude and Longitude measurements, unrelated to the vehicles current orientation. The angular deviations are measured in the vehicle's body frame (roll, pitch, yaw) as this yields an intuitive insight into the differing scale of these various offsets.

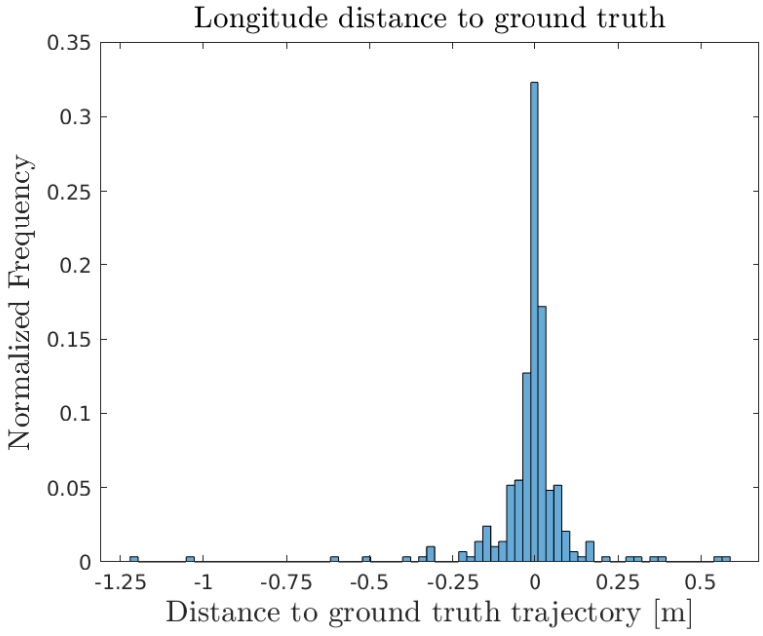


Figure 8.2: Day run: Distribution of misalignment distance [m] in longitude.

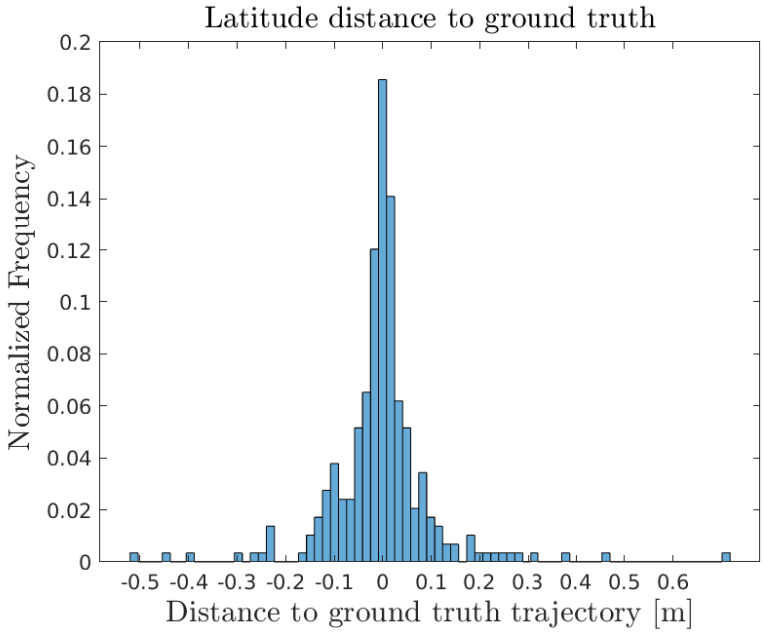


Figure 8.3: Day run: Distribution of misalignment distance [m] in latitude.

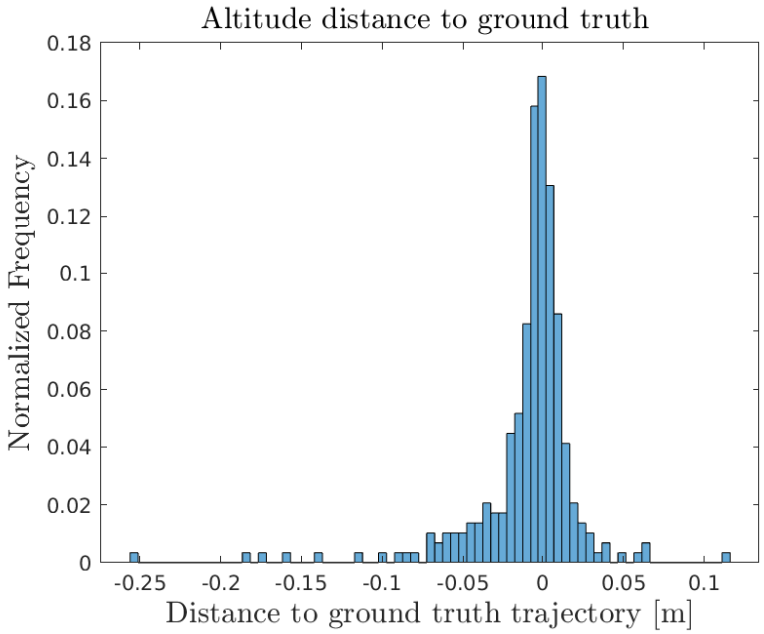


Figure 8.4: Day run: Distribution of misalignment distance [m] in altitude.

The positional deviations from the ground truth all approximately follow a normal distribution. The majority of localizations vary less than 25cm in longitude and of latitude. The altitude is expected to deviate less due to its fixed height of the vehicle deviating only in suspension travel. This means that the altitude readings have a much closer starting alignment to the ground truth than the longitude and latitude values, which however need to be shifted considerably more by the GICP algorithm.

Below are

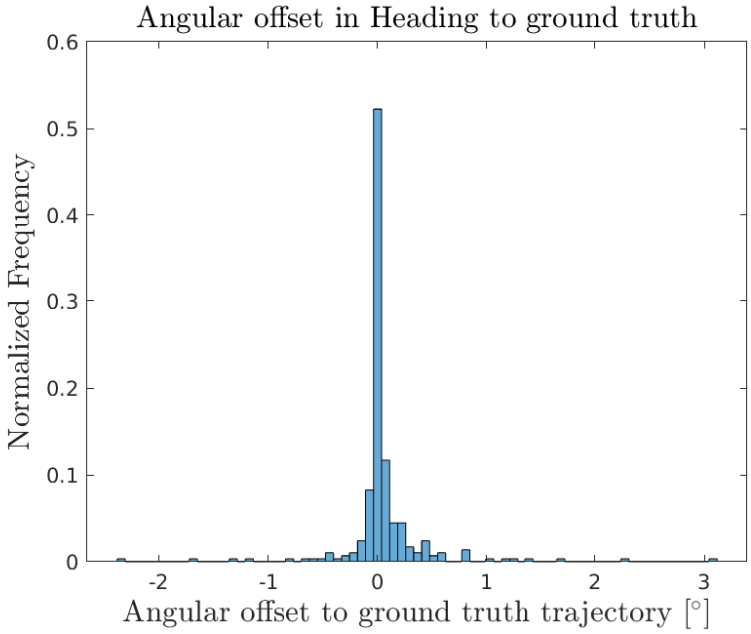


Figure 8.5: Day run: Distribution of angular misalignment [°] in heading.

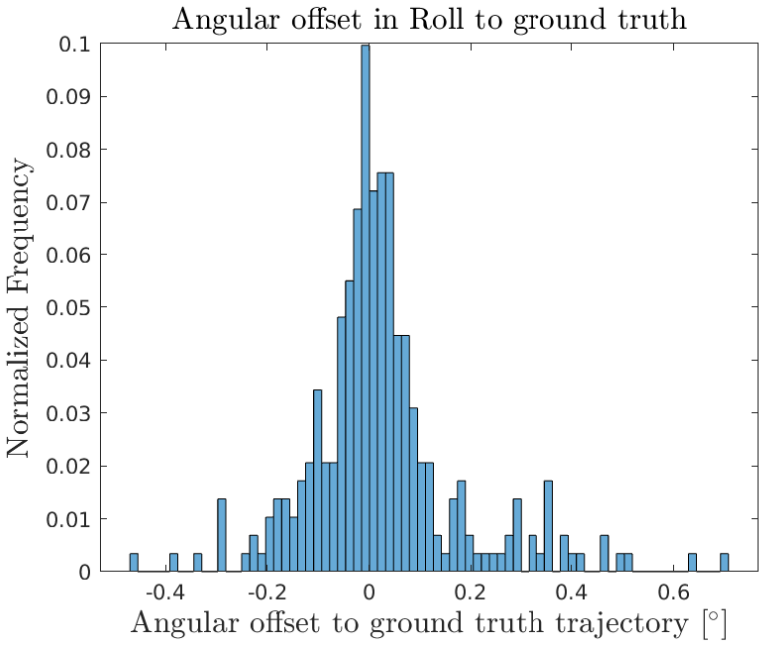


Figure 8.6: Day run: Distribution of angular misalignment [°] in roll.

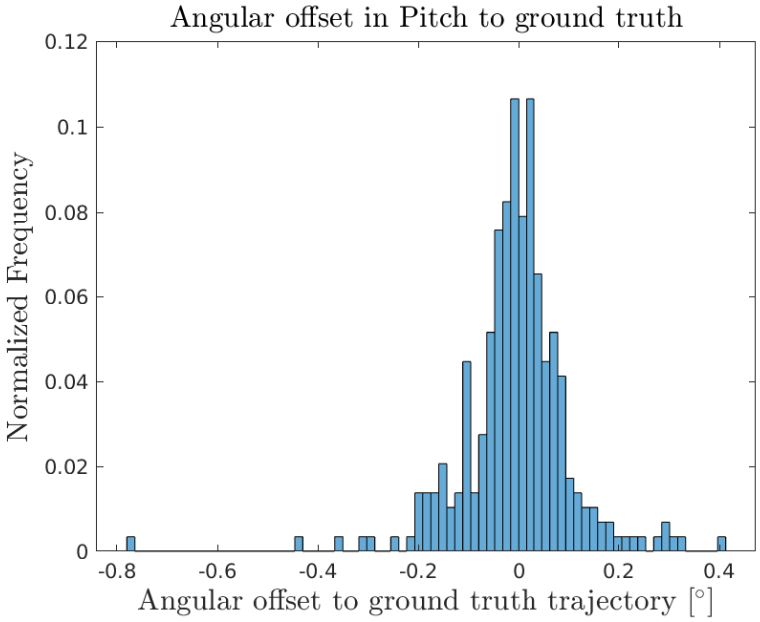


Figure 8.7: Day run: Distribution of angular misalignment [°] in pitch.

The angular deviations likewise represent approximately a normal distribution, though one which has differing covariances for its angles. The heading angle has a relatively low covariance, with the vast majority of cases (above 52%) aligned within ±0.04°, with some outliers outside of the 1° mark. The roll and pitch angles are distributed more broadly with a higher covariance but within much tighter angular deviation boundaries. This is expected since changes in roll and pitch can occur due to suspension travel in the test vehicle, but they never vary as drastically as the heading does.

8.4 Results on a night run in May

An additional test run was performed at night on the 17th of May in 2019. This test was constructed to evaluate localization performance in a dark scenario, and light rain was falling during the test as well. By now, the trees neighbouring the route had grown larger foliage due to the spring season.

The Ground Truth for this case consists of 400 instances, whilst the validation set consists of 457 instances. The Ground truth data set was interpolated and preprocessed in the same manner as for the day run. This trajectory spans the full path between Johanneberg and Lindholmen, and so adds new evaluation scenarios to the test. Due to the time of day, the traffic was much more sparse and fewer causes of ghost objects in its surroundings. The differing outdoor conditions also provide interesting insight into the localization performance under varied conditions. The classification result of the localization compared to the ground truth trajectory is shown in Figure 8.8 and categorized in Table 8.3.

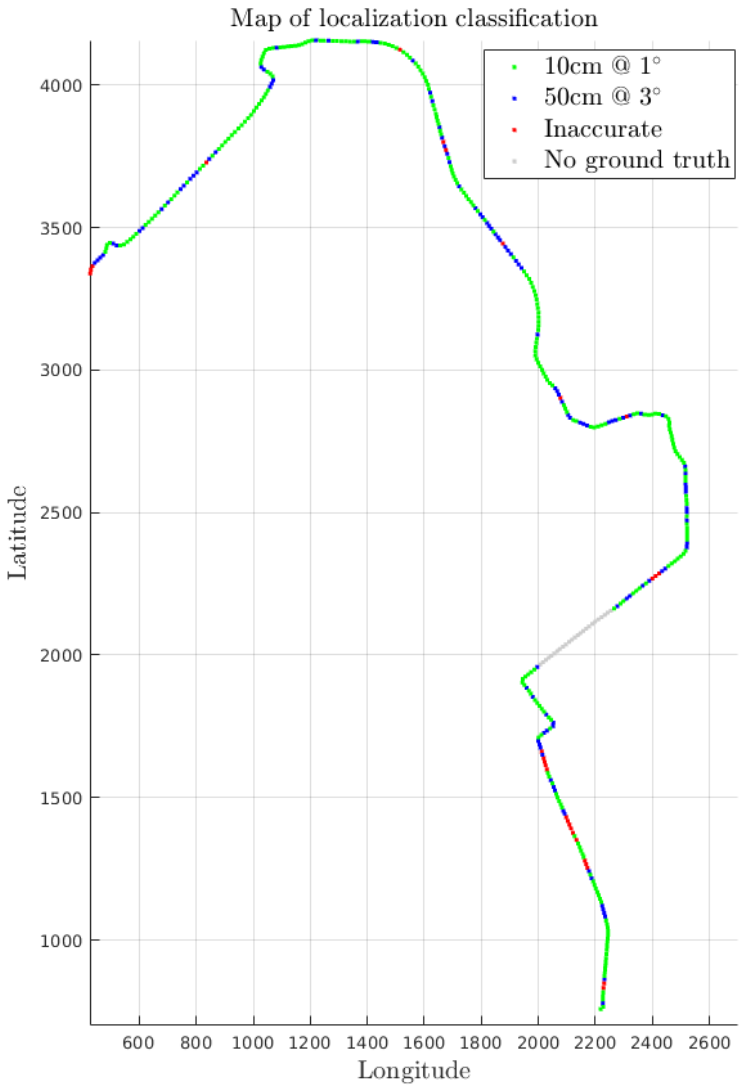


Figure 8.8: Night run: Performance evaluation of the classification of alignment instances over the trajectory. Green indicates *Good* performance, blue indicates *Ok* and red indicates *Bad* alignment. A section of the trajectory is greyed out, which is not evaluated on localization performance due to the lack of a concrete ground in this area.

Results	
Classification	Inlier %
Good	67.13%
Ok	26.11%
Bad	6.76%

Table 8.3: Results for the night run classified as Good, Bad or Ok

As can be seen from Figure 8.8, the localization still classifies the majority of instances as *Good*, but lacks accuracy in comparison to the day run. For one, the percentage of *Bad* localization instances increased considerably and they occur mostly along straight road sections. This is a surprising result, compared to the day run. Overall the performance has decreased considerably for the night run, only being able to localize 67.13% of instances accurately. To further inspect the results, the distribution of the bias in location and pose will once again be considered.

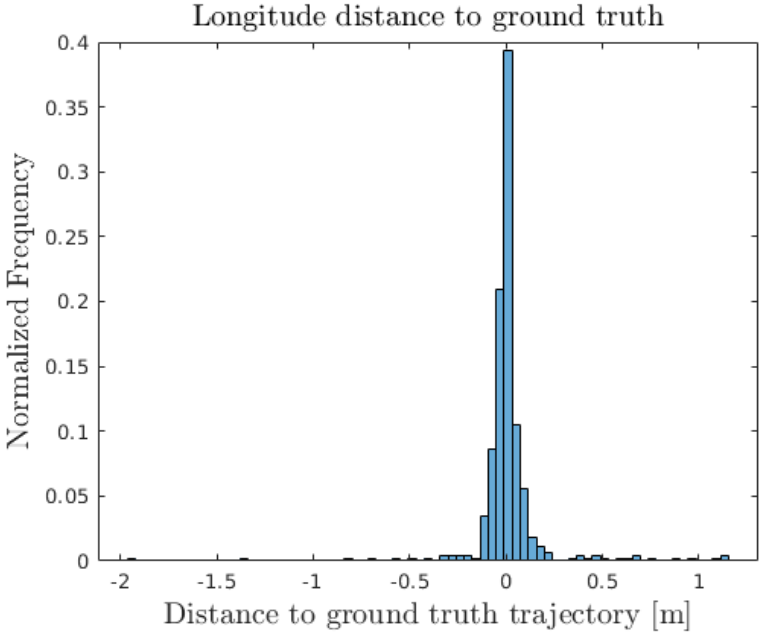


Figure 8.9: Night run: Distribution of misalignment distance [m] in longitude.

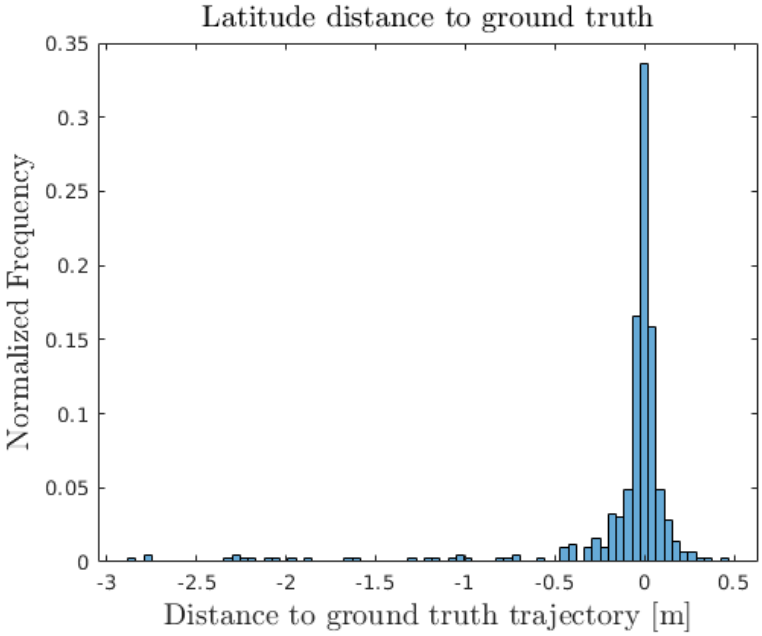


Figure 8.10: Night run: Distribution of misalignment distance [m] in latitude.

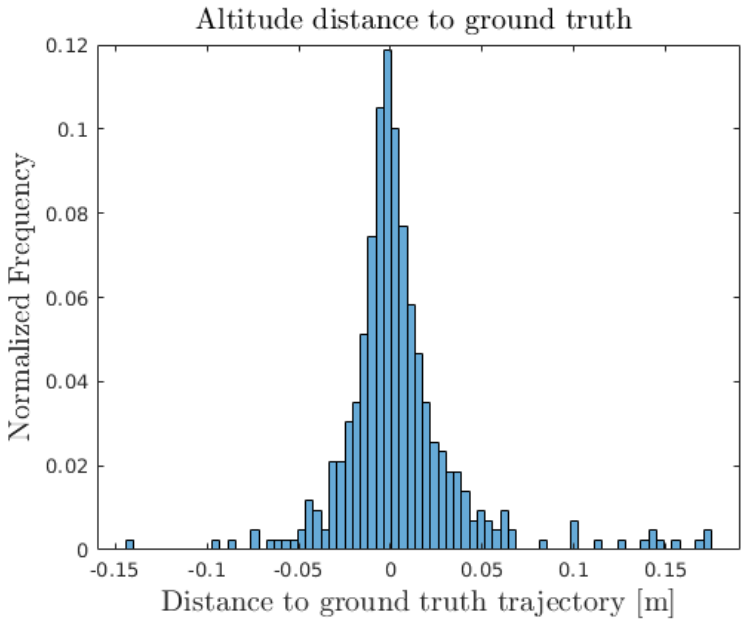


Figure 8.11: Night run: Distribution of misalignment distance [m] in altitude.

The night run distribution of errors in latitude and longitude follows a similar pattern as for the day run. To be noted is that there are larger and more outliers in the night run, ranging beyond 1.5 meters and in some cases even up to 3 meters away from the ground truth. A phenomenon to be observed is that a majority of the outliers contain a negative bias, meaning that they are lagging behind the ground truth locations. Linking the poor classifications of the night run map to the outlier distributions, one can see that these poor localizations mainly occur over the straight sections of the trajectory. This was also manually verified by plotting these point cloud alignments.

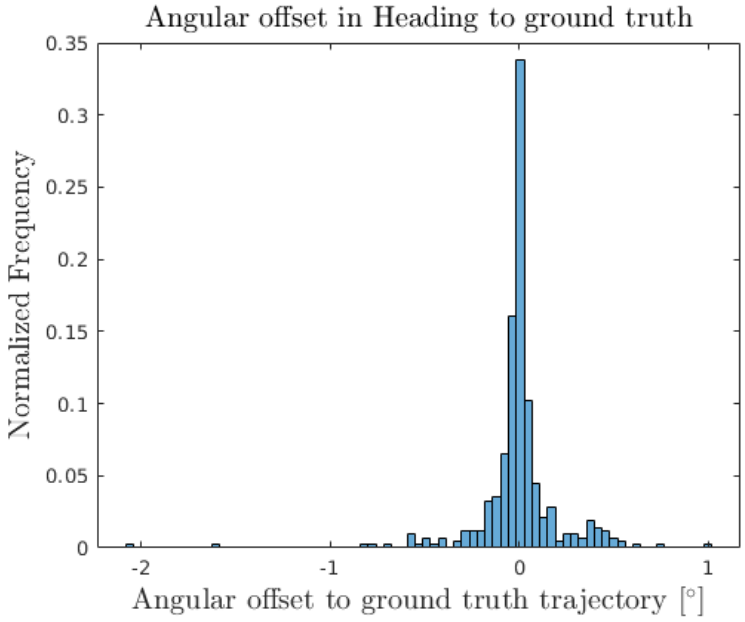


Figure 8.12: Night run: Distribution of angular misalignment [°] in heading.

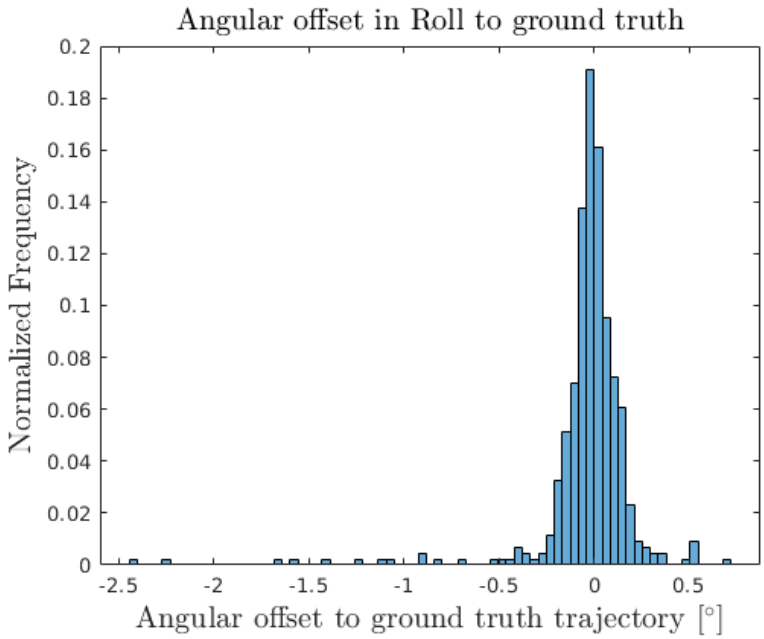


Figure 8.13: Night run: Distribution of angular misalignment [°] in roll.

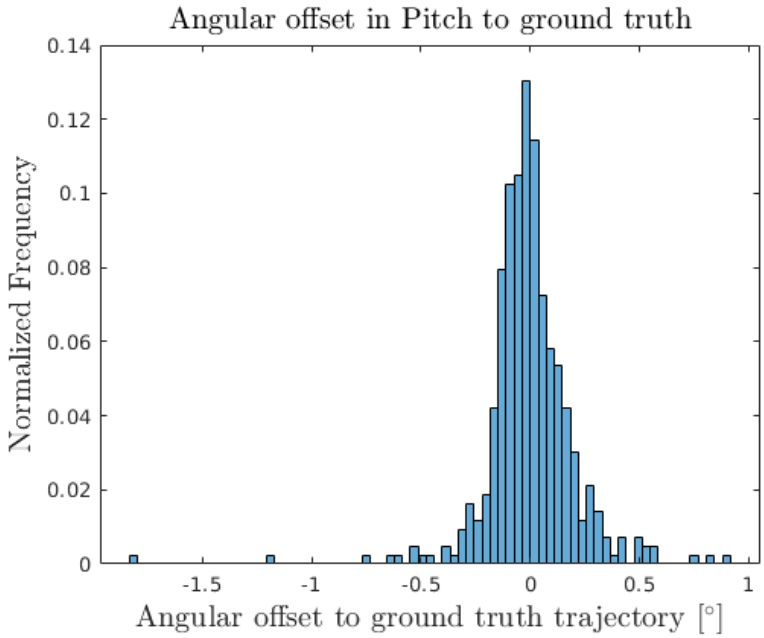


Figure 8.14: Night run: Distribution of angular misalignment [°] in pitch.

The accuracy of the heading improved for the night run, which could be explained by the increased density of the trajectory used for this test. The increase in density captures angular change better between individual instances. This means denser estimations by the smoother to predict accurate orientations of the vehicle. Conversely however, the roll and pitch angles, which are expected to be fairly small, have higher covariances and larger outliers than for the day run but are still small in comparison to the heading.

9
Analysis and Conclusion

This chapter serves as an evaluation of the thesis in the larger scope of its aims and performance. It includes an analysis and discussion about different factors affecting the performance of the algorithm and a conclusion to the thesis as a whole.

9.1 Analysis

The discussion section offers an analysis of the overall performance and the main contributing factors. Subsections of the point cloud processing are also reviewed to discuss their contribution, necessity and need for further development.

9.1.1 Main localization performance factors

The localization itself is performed by the GICP algorithm and the RTS smoother, which are the main localization performance factors. Good results are obtained by interlacing the GICP passes with the smoother, due to these two aspects complimenting each other well where the other lacks in performance. Since the smoother is estimating the trajectory bias with the GICP outputs, it is dependent on the measurements obtained by the GICP alignment. The second pass of the GICP, in turn, benefits from the improved initial positions as estimated by the smoother. The covariance assignment for the smoother is performed in a manner allowing for slight refinement of good GICP outputs, and larger potential improvements for bad outputs. An accurate GICP alignment is dependent on a close initial location estimate as input, as well as the quality of the input point cloud which preferably should contain well-defined structures and planes. Therefore, overall alignment performance is also influenced by the individual point cloud processing sections, which will be discussed in more detail in the following sections.

9.1.2 RTK GPS

The RTK GPS offers a definite advantage to the initial position and orientation estimates. Its accuracy is slightly hampered in urban, built-up areas (hence the need for this project), but offers reliable accuracy in open areas. It is an asset to have at one's disposal, as it provides the first GICP pass with better estimates than it would obtain without the Real Time Kinematic updates. The advantage of the RTK updates of the GPS measurements is evident on the Götaälvbron. There the previous work [1] found that due to bad GPS measurements (without RTK updates)

and lack of lateral features, their algorithm performed poorly. The RTK updates give extremely good initial position estimates for the section over the Götaälvbron, leading to little refinement needed by the GICP, which results in good localization on this particular part of the route

9.1.3 EGO-motion rectification

EGO-motion rectification is an integral part of not only this project, but most point cloud processing operation where the LiDAR is moving. As shown in the Validation Section 7.1, the distortions introduced into a point cloud under linear and angular motion can result in distortion of several meters, even at moderately low speeds. Planar EGO-motion rectification performed sufficiently well, and no examples of misalignment were found.
The reason for the complex EGO-motion implementation in this thesis, compared to the previous work, is that [1] found that their algorithm performed poorly in turns and argued that this was due to EGO-motion rectification only being applied linearly in the driving direction. As has been validated in Chapter 7.1, the EGO-motion presented in this thesis is able to correct distortions in LiDAR clouds in sections where the vehicle is turning. However, the localization still performs worse in turns than on straight stretches. The cause of this seems to rather be the sparseness of the trajectory which is inputted to the algorithm, instead of imperfect EGO-motion rectification.

9.1.4 Ghost object removal

The ghost object removal from the LiDAR point clouds performs consistently well, removing dynamic objects within the FOV in a clean manner. The effects which the ghost object removal has on the overall localization is not fully evaluated, since no full alignment was performed without removing ghost objects. However, the alignment is certainly still affected by ghost objects in the data, and it would be beneficial to remove all. An example of this is where a tram was recorded behind the FOV and the GICP fails to converge globally, since the tram is represented as a dense planar structure mimicking the wall of a building. This effect only gets reinforced by the merging of local clouds afterwards. This can cause the GICP to get stuck in a local minimum if the starting initial position is not closely aligned to the true solution. This point cloud was however localized correctly after the second GICP pass, which had improved starting positions from the smoother. Therefore it could be beneficial to remove dynamic objects behind the FOV as well. Since the field of view is limited to the front of the car, it was considered to only build up smaller maps using the FOV from the local clouds. This does not seem to be a too promising solution in all scenarios since almost all data can be lost in dense traffic situations and in the presence of parked cars. To remove all dynamic objects in a scene, cameras covering a 360° view can be used with the same techniques applied in this paper. Alternatively, a 3D object removal algorithm could be used as well. The manner of accomplishing the object removal here is subordinate to the end goal of being able to remove all objects in a LiDAR scan.

9.1.5 Merging of local point clouds

The merging of several local point clouds is a critical aspect to consistent, good performance because it performs well in the absence of many vertical features, where the previous thesis by [1] struggled. By merging several local clouds, a more complete and detailed representation of the current environment is obtained, which yields more features to be registered within the dense point cloud. Potentially this alignment could be done more accurately with better timestamps from the test vehicle, but this development is outside of the scope of the thesis and is handled by the development team for the test vehicle.

9.1.6 GICP performance

As described above, the fitness score yields an evaluation metric based on the Euclidean squared distance to the nearest neighbour assigned to each point in the source cloud. If the local cloud and the dense reference cloud represent the same space in every aspect of the point cloud, then the distances between each point match will be constant and yield an accurate representation of the overall alignment. However, due to several factors this is not the case. These factors include ghost objects remaining in the local and dense clouds, extreme misalignment of initial positions and also missing sections of side-roads, etc in the dense reference cloud. These inconsistencies increase the difficulty of evaluation based on the fitness score since some uncertainty can be added to the overall fitness score based on such outlier scenarios.

The fitness score itself is not reliable enough to be used as an accurate evaluation tool, but it can be used as an indicator of general performance, as is done in the algorithm. According to the fS5 fitness score, covariances were assigned according to the certainty of accuracy of a GICP alignment, which are used in the RTS smoother to estimate the trajectory. This deployment of the fS5 proved to be successful, as the smoothed trajectory was able to correct the initial GICP estimate by quite some margin.

Visual inspection of instances with low fS5 score has lead to the conclusion that the GICP is in some cases able to correctly register source clouds to the target cloud with exceptional accuracy. In these cases, the initial position has been close to the true pose and the local clouds minimally affected by ghost objects. This leads to GICP alignment which is close to being perfect which in turn contributes to the low fitness score.

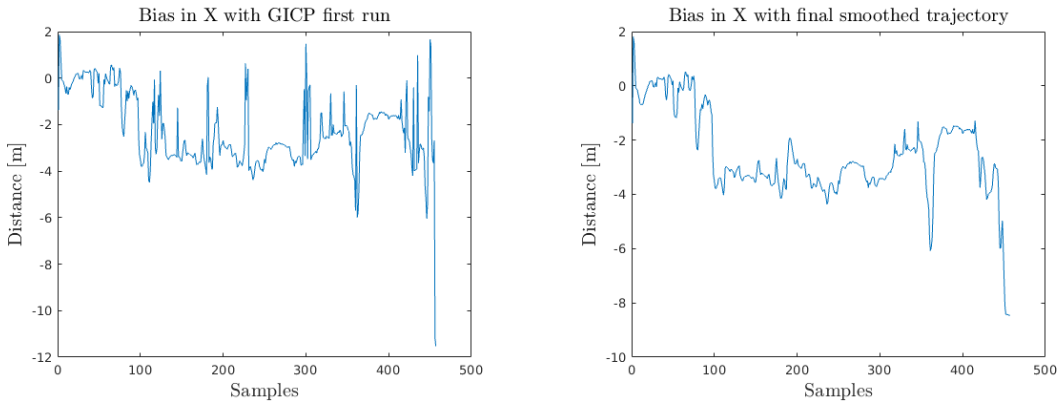
Since the GICP algorithm is very sensitive to its starting position, it fails to align itself correctly to the reference cloud when its prior location estimate is too far off. With a good initial estimate, the GICP is able to localize very well and even ignore ghost objects. It is therefore susceptible to getting stuck in a local minimum, but when close to the global optimal minimum solution, it converges well.

However, the cases when the GICP fails to localize the source cloud correctly are frequent when the initial GPS position is poor. This means that the GICP algorithm is not sufficiently good to be used solely for the localization.

9.1.7 Smoother performance

The smoother contributes at large to the performance of the algorithm, by improving the initial position of the input to the second GICP pass and also in the final smoothing of the combination of the two GICP trajectories. This has been shown previously in Figure 6.1 where the GICP results improved between iterations with new initial positions based on smoothing the measurements from the first GICP pass. An evaluation of the algorithm's localization performance was also performed without the final smoothing and the number of correct localizations decreased when the final smoothing step was omitted.

The performance of the smoother can be further evaluated by the trajectory of the bias of the GPS measurements since the smoother operates on the bias. The bias is expected to change slowly over time and the smoother should be able to smooth out outlier measurements which have been modelled with high covariances. The bias gets first evaluated after the first GICP pass and is smoothed to obtain the updated locations for the second GICP pass. After the second GICP pass, the bias is calculated again and smoothed for the last time. A comparison of the raw bias from the first GICP pass and the final output bias for the night run is shown in Figures 9.1 - 9.3 below.

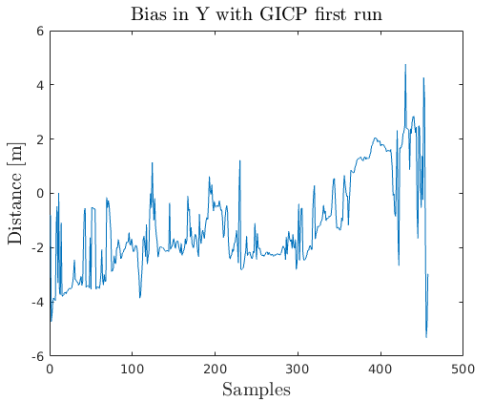


(a) Bias in the longitude after the first GICP pass.

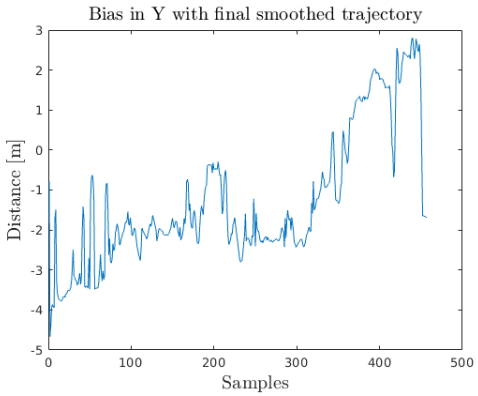
(b) Bias in the longitude in the final output of the algorithm.

Figure 9.1: The bias of the GPS measurements in longitude estimated after the first GICP pass (a) and after the second smoothing (b).

After performing the first GICP pass, the bias in the longitude (or X) direction is clearly affected by outliers, as can be seen from Figure 9.1 (a). Comparing it to Figure 9.1 (b), one notices that the final estimated bias is more consistent. Over the first 100 samples, it seems to be centered around 0 meters, but for samples 100 to 400 it seems to be centered around -3.5 meters. However, there still remain points that look like outliers and the estimated bias is clearly varying between measurements.



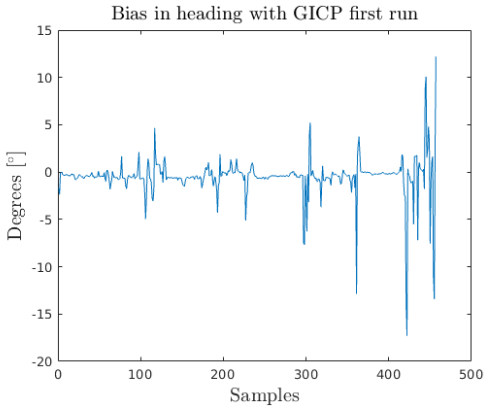
(a) Bias in the latitude after the first GICP pass.



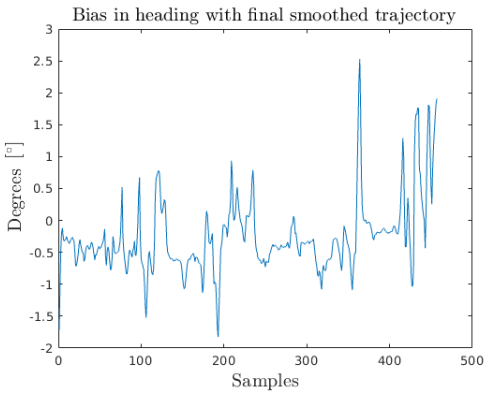
(b) Bias in the latitude in the final output of the algorithm.

Figure 9.2: The bias of the GPS measurements in latitude estimated after the first GICP pass (a) and after the second smoothing (b).

The bias in the latitude (or Y direction) fluctuates a lot in the output of the first GICP pass, but after the final smoothing it looks like it grows more linearly over the route. Shown in Figure 9.2 the bias starts off at -4 meters but steadily grows to above 2 meters. However, the same as with the bias in longitude, the final smoothed trajectory still suffers from outliers.



(a) Bias in the longitude after the first GICP pass.



(b) Bias in the longitude in the final output of the algorithm.

Figure 9.3: The bias of the GPS measurements in the heading estimated after the first GICP pass (a) and after the second smoothing (b).

The bias of the heading seems to be small, ranging between 0 and -1 °. It is the same however for the longitude and latitude, that the measurement sequence from the initial GICP run suffers from outliers. Yet in the case of the heading, the final smoothed bias is varying significantly less than the initial GICP output and outliers can clearly be recognized.
From this discussion it can be noted that the smoothing of the bias seems to help quite a lot. After the GICP rerun and the final smoothing, the bias seems to behave

more constant, as one would expect. However, there are still quite a few outliers in the final estimates of the bias. These outliers could be the instances where the localization performs badly. This was however not verified, but adding an outlier rejection to the final smoothed trajectories might help to improve the localization.

9.1.8 Evaluation of robustness to changes in conditions

Robustness of localization in regard to ghost objects and varying outdoor conditions was one of the main research questions of the thesis. Due to the dynamic environment of an urban traffic scene, this is a crucial part of accurate localization. The evaluation of the robustness localization to changes in the physical, structural environment as well as varying weather conditions is presented in this section.

Structural changes

Structural changes in the perceived 3D environment can cause misalignment of the GICP algorithm itself, due to planes being shifted to different locations compared to the reference cloud. An example of this can be seen at Nya Allén, where maintenance is being performed on several buildings. The scaffolding which is built around the facade of the buildings creates a new plane which is detected by LiDAR scanners. This creates discrepancies between the reference point cloud and the local point clouds, as some scaffolding is present in the reference cloud which is no longer present in the local clouds, and vice versa. The GICP naturally struggles in such areas, as these new planar structures cause an easy misalignment in its point matching. The smoother performs well at this stage, as it is not influenced by the structural shapes of the point clouds themselves, and rectifies the bias found in the GPS trajectory.
In general the algorithm performs really robustly to ghost objects and even such structural changes in the environment. This can again be verified in a more difficult environment, by driving through construction areas, such as can be seen in Figure 9.4. This is a localization example at Nordstan where the construction site altered the nearby structures as well as the vehicle path.
The algorithm still manages to localize the test vehicle with exceptional accuracy, even in this difficult environment. As long as the GICP receives an accurate initial position, it manages to converge correctly to the global minimum, instead of getting stuck in a local minimum. When the initial alignment is poor, ghost objects indeed affect the GICP alignment a lot. The smoothing instances, however, perform exceptional in ignoring ghost objects and predicting new starting positions for the second round of the GICP.

Robustness to varying outdoor conditions

The number of possible outdoor conditions to be evaluated is vast. Due to the availability of the test vehicle and seasonal weather conditions, only a limited number of varying conditions were evaluated. For both the day and night run, a difference in the season caused the foliage on trees to be more prominent than when the dense reference point cloud was created. The alignment did not seem to be affected by this however. The weather for the day run was sunny and the images from the day run

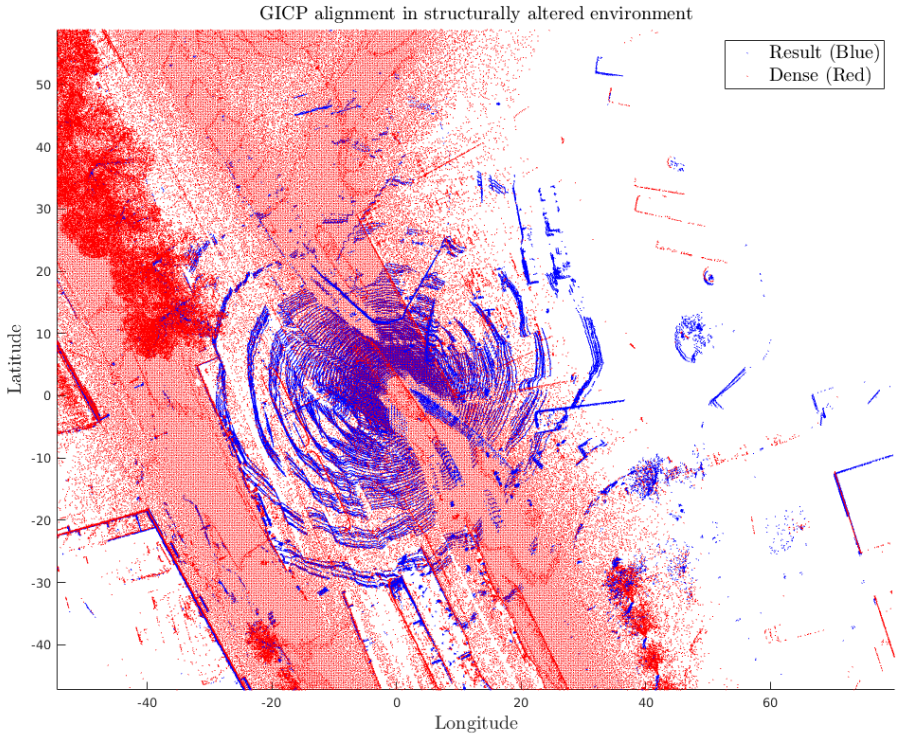


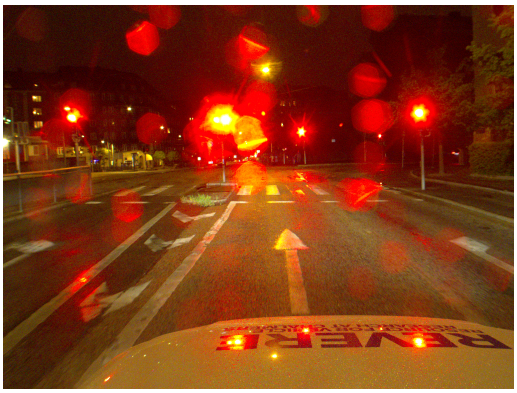
Figure 9.4: An example of an altered structural environment at Nordstan, Gothenburg. The blue cloud represents the local environment of a construction site, which is drastically altered to the red reference point cloud. Nevertheless, due to the accurate starting position and clear surface features of the buildings, the GICP algorithm is able to converge.

look bright and are at times at risk of overexposure which would cause information loss. This can be controlled by setting the maximum exposure limit for the camera and so this did not influence the localization. The second data set was recorded was in night time conditions, with lower visibility due to the dark and light rain. The darkness affects the camera images mainly, capturing data at a different exposure level than during daytime. Visually, the images were clear and one could distinguish objects well. Some difficulties encountered in the night run are discussed below in Section 9.1.9. The algorithm would need to be run on data sets captured under more extreme weather conditions (such as snow) in order to fully conclude how robust the algorithm performs under different weather conditions.

9.1.9 Evaluation of poor performance at night

As mentioned in the results section, the location estimates for the night run have far more (and larger) outliers than for the day run. Combined with their locations on the trajectory map in Figure 8.8 it can be seen that these classifications lie mainly on straight road sections. The localization for these instances lags behind the true location values by a couple of meters at times. It seems that the bias in the driving

direction (X direction) fails to be estimated by the smoother and neither is the GICP algorithm able to correct for this misalignment. By visual inspection of these localization instances and plotting their point cloud alignments, it can be seen that strong planar surfaces are measured in the driving directions, but a lack of planar surfaces in the lateral to the vehicles driving direction. This could be a factor that causes the bias to fail to be estimated fully, since potentially outputting a low fitness score (and hence low covariances assignment to the measurements).
A further problem for the night run is that some images have been poorly segmented to detect objects. In many cases the semantic segmentations perform fairly, being able to detect vehicles, road signs and other structures in the surroundings correctly. The detection seems to struggle in the presence of rain however, as light refraction in water drops on the camera lens disturb the image greatly. An example of this can be seen in Figure 9.5, where the semantic segmentation classifies pixels as a car in the sky.



(a) Image example of a night run scenario.

(b) Semantic segmentation of the night run scene.

Figure 9.5: Shown above is an example of a night run image influenced by rain, and its respective semantically segmented image. As can be seen, the semantic segmentation fails in this case and predicts a car to be in the sky.

9.2 Conclusion

Based on the results presented in this thesis, above 97% of the time the algorithm is able to localize the test vehicle with an accuracy of 50cm at 3° for the day run. The night run with light rain yielded a lower accuracy estimation, mainly in position, only localizing 93.24 % of instances within 50cm at 3°. The accuracy below 10cm and 1° drops by about 7%, which is due to positional misalignment since the orientation was estimated more accurately than for the day run. The improvement in the heading estimation is due to the fact that a more denser trajectory was considered, in which the smoother is able to better update consecutive measurements. Considering the overall performance, by inspection of the cloud alignments themselves, the vast majority of instances seem to provide sufficient accuracy so that the nearest neighbour in the local cloud is the corresponding object in the dense point cloud.

The thesis has answered the research questions proposed at the beginning of the project, yet only providing a limited evaluation of different outdoor conditions. The individual sections of the algorithm, such as EGO-motion rectification, ghost object removal and the merging of several local clouds, have been implemented and validated. Overall, the algorithm is able to improve the localization accuracy compared to the previous work by [1], as was set out to be achieved.

9.3 Future work

A definite improvement could be done by creating denser validation and ground truth trajectories. These trajectories were strongly down-sampled due to the computational efforts needed, as well as human time required to set up the ground truth. Improving these trajectories should however yield improved results, as the smoother will have more and denser estimates to compute the location and pose of individual point clouds. This will allow the smoother to obtain a more consistent estimate of the bias and, as discussed in Section 9.1.7, should smooth out even more of the bad measurements. This should boost the accuracy and consistency of results.
A further aspect to be addressed include fixing the LiDAR sample timestamps. These are currently not ordered in the correct sequence, although their time estimate is expected to be more accurate than the sent timestamp which was used instead. This is, however, an issue which the test vehicle's development team needs to address since the data obtained for the thesis contains this issue already. This would yield overall higher accuracy in aligning local point clouds, removing the impact of integration drift and aligning a point cloud to an image to remove ghost objects.
The ghost object removal could potentially be improved by using a 3D object detection algorithm. This would allow dynamic objects behind the camera FOV to be removed as well, but at this time was not an option for this thesis due to the lack of time which would be required to train such a network oneself.
An interesting thing to look at would be attempting to remove ghost objects from the data after the localization has been performed. As the localization is often pretty accurate, meaning that most of the points from the two clouds align have a corresponding neighbour in the other cloud, this could perhaps be used to detect the ghost objects. A point in the local cloud that has no neighbour in the dense cloud within a certain distance is very likely to originate from a ghost object. Clusters of points in the local clouds that have no near neighbours in the dense cloud could thus possibly be identified as ghost objects and removed based on that. A similar methodology could be applied for detecting and removing points in the dense cloud but that would require a bit more sophisticated approach, as the local clouds don't necessarily fully present the surroundings. In this way, if the localization is performed correctly, it is possible to annotate points from the LiDAR point clouds with the information stored in the corresponding point in the dense clouds and disregard the points that correspond to ghost objects.
Other recommendations to improve alignment include detecting and matching like objects in the local and dense point clouds, for instance matching poles to poles.

Bibliography

[1] Gunnar Bolmwall and Måns Östman. Precision localization in dense point cloud maps. Master's thesis, Chamers University of Technology, Sweden, 2018.

[2] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[3] Richard Szeliski. *Computer vision algorithms and applications*. Springer, London; New York, 2011.

[4] Fredrik Kahl. Lecture notes, 2018. Lecture Notes in Computer Vision EEN020 at Chalmers University of Technology.

[5] Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, and Hang-Fei Cheng. Complete solution classification for the perspective-three-point problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25:930– 943, 09 2003.

[6] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[7] N. D. McKay P. J. Besl. A method for registration of 3-d shapes. *IEEE Trans. on pattern analysis and machine intelligence*, 1992.

[8] Aleksandr V. Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp.

[9] Simo Srkk. *Bayesian Filtering and Smoothing*. Cambridge University Press, New York, NY, USA, 2013.

[10] Pierre Merriaux, Yohan Dupuis, Rémi Boutteau, Pascal Vasseur, and Xavier Savatier. Lidar point clouds correction acquired from a moving car based on can-bus data, 2017.

[11] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[12] Mans Larsson, Erik Stenborg, Lars Hammarstrand, Marc Pollefeys, Torsten Sattler, and Fredrik Kahl. A cross-season correspondence dataset for robust semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[13] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *CoRR*, abs/1612.01105, 2016.

[14] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[15] G. Neuhold, T. Ollmann, S. Bulo, and P. Kontschieder. The mapillary vistas dataset for semantic understanding of street scenes. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5000–5009, Los Alamitos, CA, USA, oct 2017. IEEE Computer Society.