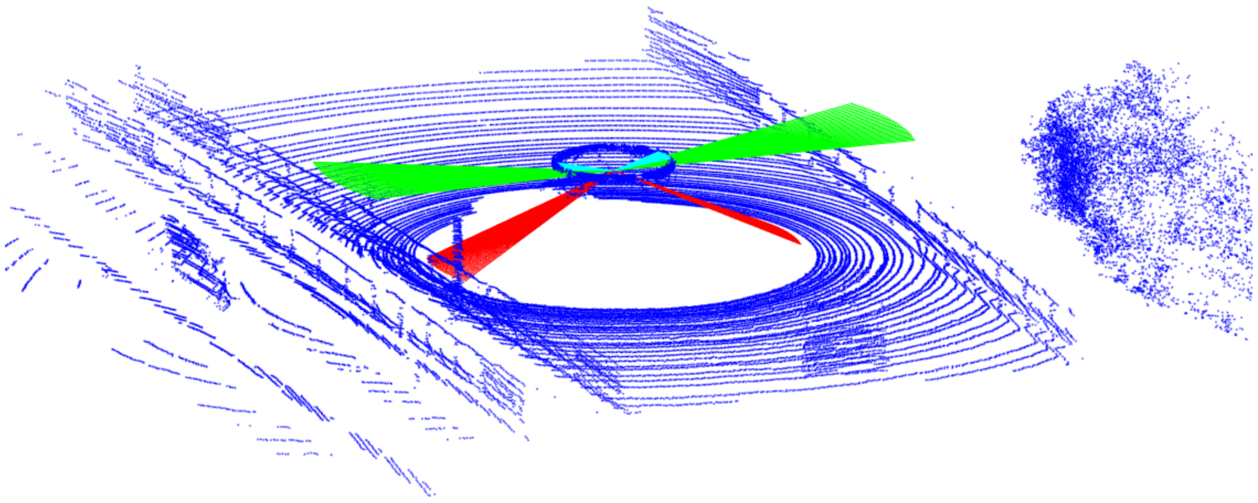
CHALMERS
UNIVERSITY OF TECHNOLOGY



# Ego vehicle localisation using LIDAR and compressed digital maps

Master's thesis in Systems, Control and Mechatronics

Albin Aronsson & Adam Eriksson

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

Master's thesis 2018: EX036/2018

# Ego vehicle localisation using LIDAR and compressed digital maps

Albin Aronsson & Adam Eriksson

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2018

Ego vehicle localisation using LIDAR and compressed digital maps

Albin Aronsson & Adam Eriksson

© Albin Aronsson, Adam Eriksson, 2018.

Supervisor: Tony Gustafsson, Andreas Schindler, Mina Alibeigi Nabi, Rafael Valencia Carreno, Junsheng Fu & Markus Hammarsten.
Company: Zenuity
Examiner: Lars Hammarstrand, Electrical Engineering

Master's Thesis 2018: EX036/2018
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A representation of a point cloud from part of a LIDAR scan of the surroundings of the vehicle.

Typeset in LaTeX
Gothenburg, Sweden 2018

Ego vehicle localisation using LIDAR and compressed digital maps

Albin Aronsson & Adam Eriksson
Department of Electrical Engineering
Chalmers University of Technology

## Abstract

Many people are wounded and die every year due to traffic related accidents. There has also been a huge increase in interest and development in autonomous and advanced driver-assisted systems(ADAS) in the last decade. These techniques have the definite potential of reducing the suffering from accidents by preventing them in the first place.

An important aspect for ADAS and autonomous driving is the ability to determine a vehicles position as precisely as possible, while also doing it in real-time. This project attempts to tackle ego-vehicle localisation by utilising a new type of highly compressed digital map layer called RoadDNA, which is designed for speed and performance. An importent component is also a roof-mounted rotating Velodyne LIDAR.

To accomplish the project objective, a localisation algorithm is created which centres around a particle filter. A coordinated turn model is used for motion prediction, the measurement update uses a "nearest neighbour" approach where LIDAR point clouds are correlated with the digital map. The position is estimated from the filter with an MMSE method.

The algorithm is evaluated on pre-recorded data of vehicles driving around the DriveMe test track in Gothenburg, Sweden. With a standard deviation on the lateral error at $\sim 0.37$m and a standard deviation of longitudinal error at $\sim 1.3$m, from the longest continuous test (4.7km). The project is concluded to be a success. For future works this localization algorithm can be combined with other localisation method based on different set of sensors.

Keywords: LIDAR, GPS, localisation, autonomous drive, AD, ADAS, RoadDNA, map, map layer, Velodyne, filtering, laser, sensors, DriveMe.

# Acknowledgements

This project has been long and sometimes wearisome and it has not always been easy to clearly see the road ahead of us. Yet we have not strayed this path completely alone and our gratitude go out to the people who have helped us on the way. Our examiner at Chalmers, Lars Hammarstrand, who took on our project even though he was very busy and had already taken on several other groups. Our supervisors at Zenuity, Tony Gustafsson and Andreas Schindler, for guiding us through this. Also the rest of the Road Estimation team at Zenuity for all their help and many interesting and insightful ideas. And also our friends and families for helping us keep sane and persevere until the finishing line.

Thank you all for the help and advice we got, it would have been very difficult without!

Albin Aronsson & Adam Eriksson, Gothenburg, June 2018

# Contents

# 1
# Introduction

## 1.1 Background

Today more then 1.25 million people die every year and around 20-50 million people get injured due to traffic related accidents [1]. It is estimated to become the seventh biggest killer in the world by 2030 [1]. Traffic accident costs are behind approximately 3% of a country's GDP[1] and is therefore even a financial problem, aside from the humanitarian tragedy, that needs to be solved.

Therefore research on autonomous vehicles is a topic of significant interest worldwide. A first goal would be to let the driver choose when to drive actively, and when to let the car drive itself. This will hopefully lead to less traffic accidents and a more comfortable driving experience. There is also huge business interest, the autonomous car market is approximated to a part of a 7 trillion dollar market by 2050 [2]. Important steps towards fully automated vehicles consist of a large number of active safety systems designed to warn or support the driver.

Both autonomous vehicles and active safety systems require accurate and reliable information about the surroundings in order to make the right decisions both during normal driving and in critical situations. Localization of the vehicle is therefore an essential part in all autonomous driving projects. The description of the surroundings consists both of the static environment and other road users. In this case the static environment is modelled using digital map layers which provide data about the road, lanes, landmarks, etc. In addition to the static environment the car has on-board sensors (LIDAR, GPS, IMU, etc.). These can be used in conjunction with the digital maps for performing localization.

## 1.2 Problem formulation

The problem that is to be tackled by this report is to: produce an algorithm in the form of a statistical filter, its primary function is to determine the position of a vehicle with high accuracy and, if possible, in real-time. The algorithm is to make use of the digital map layer called RoadDNA [3] provided by TomTom and the available on-board sensors of the test vehicle.

The project is to be seen as successful if an algorithm is implemented with which the position of a test vehicle can be tracked sufficiently well. The ultimate target is a position error of a few decimetres or less.

## 1.3 Limitations

In terms of the scope of the project, it is decided to mainly focus on pure signal processing aspects. What that means is that we intend to focus on the core filtering algorithm and the inputs and output values. Lesser focus will be put on such questions as how the algorithm can smoothly switch between different map files. The vehicle in question is also assumed to be located on a road (as opposed to off-road), and always have a RoadDNA map layer associated with it.

## 1.4 Report outline

**Chapter 2** gives the theoretical foundations on Bayesian filter theory and necessary concepts, sensors and methods that are used in the implementation of the final algorithm. The chapters structure starts with describing the fundamental information needed in multiple sections with related subjects, while the final section is a collection of miscellaneous subjects needed for work with the point clouds.

**Chapter 3** details how the final algorithm is implemented. The first section gives an overall view of the algorithm along with a flow chart. The following sections explain different components of the implementation.

**Chapter 4** concerns the evaluation of the implemented algorithm. Included is a description of the data set used, how the evaluation was performed and statistical results from the evaluation. Some observations on the general behaviour of the algorithm are also included.

**Chapter 5** consists of the final conclusions of the report along with suggestions on future work.

# 2
# Theory

This chapter provides and explains the theory behind the various subjects encountered in this report.

## 2.1 Vehicle sensors

This section describes the types of sensors available on the vehicle platform used in the project which are also relevant to the implementation.

### 2.1.1 LIDAR

A LIDAR is a sensor which uses laser beams to measure the distances where the beams hit and reflect [4]. See figure 2.1 for a clear visualisation of how the LIDAR works. The beams are composed of either visible light, or more commonly by infrared light. The angles of the beams are recorded and together with the distances a 3D point cloud of reflection points can be built up around the LIDAR sensor.

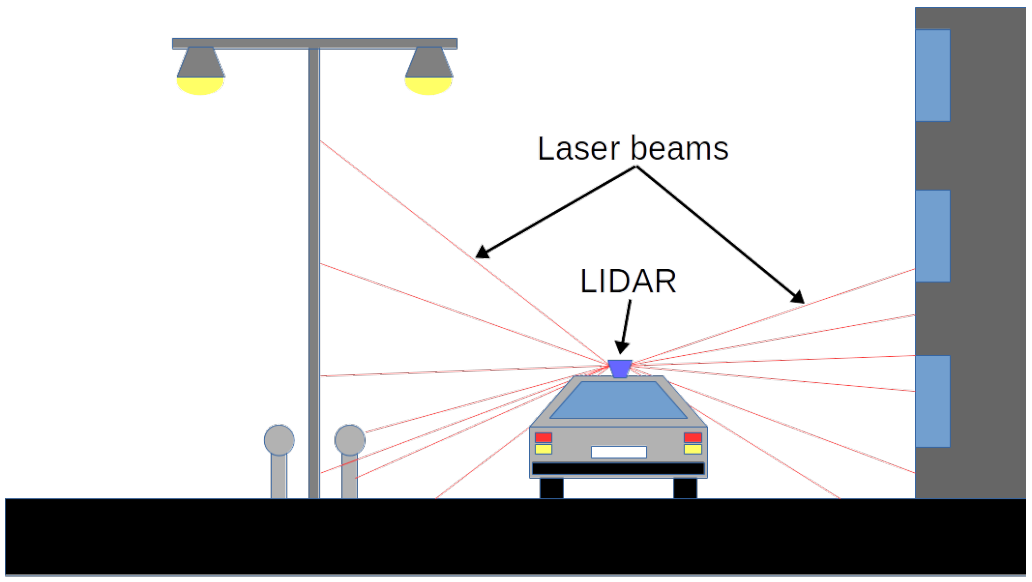


**Figure 2.1:** Visualization of how a roof-mounted LIDAR can work while driving on a road. The laser beams reflect of objects and structures in the environment. The angles and distances of the reflections are recorded by the LIDAR.

LIDAR is an abbreviation of "LIght Detection And Ranging" and is very similar to the more commonly known radar [4]. Both LIDAR and radar work in similar ways and give similar outputs. One of the biggest differences is that while LIDAR use laser light, radar use (relatively) low frequency electromagnetic waves such as microwaves. This has some consequences in terms of the technical implementation and also for example the resolution of measurements.

Another quantity the LIDAR can measure is the reflectivity value of the materials on which the laser beams reflect. This is a measure of how much light was reflected and returned to the sensor when the beam hit the object. Different materials will absorb (and reflect) different amounts of light, which might be useful to determine which types of object the LIDAR detects.

The LIDAR used for this project is a Velodyne HDL-64E [5]. This is a roof-mounted, continuously rotating 64 beam LIDAR. The sensor has a 360° horizontal field-of-view with a 24° vertical field-of-view. It rotates around the vertical axis with a rate of about 10Hz and provides roughly one million 3D coordinate points of reflections per second. This amount of data points is without any type of filtering of the measurements.

### 2.1.2 GNSS

GNSS means "Global Navigation Satellite System" and refers to any satellite based system which is used for navigation and localization. The most commonly known such system is the GPS system. The Global Positioning System (GPS) is originally an American military positioning system [6]. Other similar systems include: the Russian Glonass, the European Galileo and the Chinese Compass [7]. Since GPS has been so common the name has almost become synonymous with GNSS and in many cases "GPS" might refer to one of the systems. For simplicity's sake throughout the report the name GPS will be used for any satellite based positioning system/GNSS.

A GPS sensor is a device which receives signals from orbiting satellites and uses these to trilaterate the sensors global position [6]. Simply explained the satellites send out short messages detailing their current positions and when the message was sent to determine the sensors position.

An important question when working with positioning of any vehicle or device is whether GPS is to be used alone. However, consumer grade GPS systems alone are generally not considered to be highly accurate [8] and it is often difficult to obtain sub-metre levels of accuracy. The update rate of the position estimate (such as from a GPS sensor) is also important for time critical systems such as steering an autonomous car. Update rate depends on the receivers design and can vary a lot between different GPS devices.

The GPS sensor used in this project comes from an OxTS RT3000 sensor package [9]. It is a non-consumer grade device which among other things provide the geographical position of the device. It can use Real-time kinematic (RTK) [10] GPS for positioning and thereby has a very high accuracy. RTK uses integrated high precision IMU and can use post-processed correction signals which enables even higher accuracy.

### 2.1.3 Odometry

Odometry is a means of estimating the distance travelled by recording speed and direction and integrating that with the elapsed time [11]. For cars and other vehicles the speed is often measured with a sensor that measures the rotational speed of the wheels. If the radius of the wheel is known (or approximated) the speed of the vehicle can then be easily calculated.

A common deficiency of odometry is that, due to integration, any errors in the speed measurements are accumulated over time [12]. This means that the accuracy of the position estimate will generally degrade over time. Wheels can also slip on the road surface and since this is not necessarily noticed or recorded it also contributes to errors.

In this project odometric sensing is available from a standard set of various sensor combinations built into the test vehicle. These are simply called *car signals* and they provide a multitude of different sensor data. The specific ones related to odometry used in this project are *linear velocity* (m/s) and *turn rate* (rad/s).

## 2.2 RoadDNA map layer

RoadDNA is a relatively new type of digital map layer provided by the Dutch company TomTom [3]. These maps layer can be considered as a heavily compressed form of LIDAR data of the road stretches covered. Raw LIDAR data, which is often represented by 3D point clouds, can take up very large amounts of digital storage space. This is purportedly what RoadDNA solves by compressing the data in a novel way.

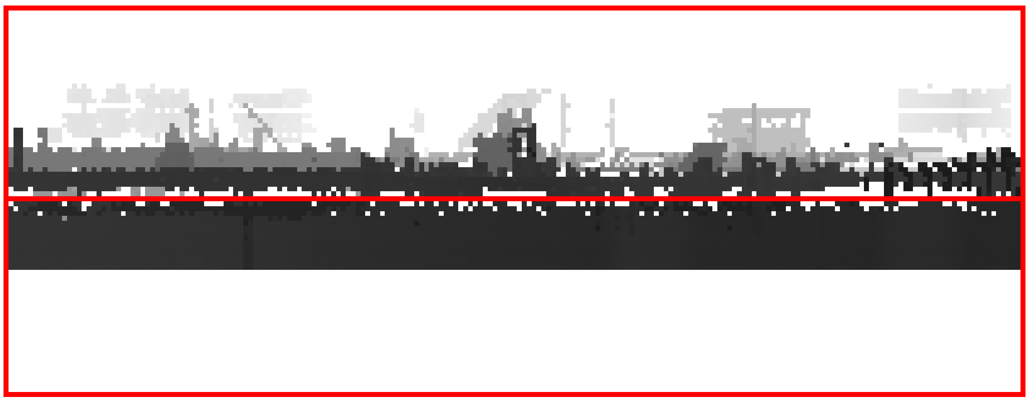


**Figure 2.2:** An example of roughly how a RoadDNA map layer segment looks like. The red lines are added and the middle line shows the separation of the right-hand and left-hand sides. The upper half is left and lower is right. (NOTE! This is NOT an official map from TomTom! It is our own creation.)

In RoadDNA only the depth information along the road is kept [13]. The depth information is essentially the distances to structures or objects seen perpendicular from the road. This information is taken from LIDAR scans along the roads. The roads in the map layers are defined by reference lines. Each line is a sequence of global coordinate points (latitude, longitude and altitude).

Maps are stored in 2D grayscale image files of PNG format. See figure 2.2 for an example of how this might look like (note that there is a wall on the right side which means less unique features compared to the left side). The image height is 80 pixels and the width depends on the length of the road stretch. The upper half of the image is the left side view of the road and the lower half is the right view. This means that the maps are directional and that there are 40 vertical pixels for each side.

One pixel on the horizontal axis of the image equals 0.5m along the reference line. The pixels on the vertical axis are non-linearly distributed, meaning that pixels at different heights are not of equal size. The first pixel row corresponds to just above the road surface and the last pixel row is 16m above the road. The exact equation for the distribution is confidential and not publicly available.

The grayscale color of each pixel is given as an integer number in the range [0,255], where 0 appears as black and 255 as white. Each pixel represents one distance perpendicular to the road and can range from 0m up to a maximum of 61m. A pixel value of 255 will therefore mean that the distance is 61m or more. This means that any distance which is considered as infinity (such as the sky) has a pixel value of 255. The distribution of the pixel values are, just like the vertical pixels, non-linearly distributed. The exact equation for this distribution is also confidential and not publicly available.

The combination of these divisions (horizontal, vertical, depth) means that the world is essentially divided into a non-linear 3D grid of voxels (voxel = the 3D equivalent of a pixel, like a box). The depth value for each pixel is given by detecting in which voxel the closest LIDAR detection was given. If no detection exists for that pixel it is given the maximum pixel value of 255.

## 2.3 Particle filter

This section covers the subject of Bayesian filtering by using the method of particle filtering.

### 2.3.1 What is Bayesian Filtering

Bayesian filtering is the means by which unknown quantities are estimated by using past and present observations (as opposed to smoothing which also makes use of future observations) [14]. Furthermore, it implies that a Bayesian statistical framework is used for the underlying operations. An important part of this is the statistical formula known as *Bayes theorem*, see (2.7). This formula provides the ability to estimate a quantity from other related quantities (often measurements). This project specifically attempts to use it to relate vehicle measurements and digital maps to the position and orientation of the vehicle.

### 2.3.2 Overview

A particle filter (PF) is a Bayesian filter based in Monte Carlo methods [14]. This means that it randomly draws samples from some distribution and uses them for estimation of said distribution. These samples are called particles and is the reason for the naming of the filter. Each particle represents a possible value of the state vector (see section 2.3.3) of the system, essentially being a guess of the "true" state of the system. Each particle also has an associated weight which can be seen as how "good" or likely of a guess it is.

The particles are propagated using a motion model, which can be called state prediction (see section 2.3.4). Each particle then has its weight updated by using some measurements, this can be called measurement updating (see section 2.3.5).

After the weights have been updated the particles can be combined with the weights to give an estimate of the true state of the system (see section 2.3.7). Finally the particles are resampled (see section 2.3.7) and the entire process is repeated for the next step in time and with a new set of measurements.

The primary reason for electing to use a PF in this project was that it is relatively easy and fast to implement while still providing good opportunity for tuning and optimising. In some early testing we performed it did also seems to be quite robust for systems with possibly non-Gaussian behaviour. It is by no means the only way of solving this task, which is for example touched on in the Discussion chapter where a PMS (Point Mass Filter) is suggested for future work.

### 2.3.3 State vector

A state vector is a specified set of quantities which describes the state of a system. Exactly what these quantities are can greatly vary from one implementation to another. But for tracking the position of a system such as a car, it is reasonable to at least include some combination of position, orientation, velocity and acceleration.

The state vector can be said to describes the current state of a system at any point in time, compared to a motion model which describes how the system behaves from any point in time to another.

$$X_k = \begin{bmatrix} x_k \\ y_k \\ \dot{x}_k \\ \dot{y}_k \end{bmatrix} \tag{2.1}$$

In (2.1) is an example of how the state vector can be composed. This example has two position values ($x_k$ and $y_k$) and two velocity values ($\dot{x}_k$ and $\dot{y}_k$). A dot over the variable is a common way to indicate time derivative (and hence velocity).

The index $k$ refers to the discrete-time index, as the state vector can be used in either discrete or continuous format. Both form have different implications and uses. But in this project a discrete form is used because our observations from measurement are in discrete time.

$$X_k = \begin{bmatrix} x_k \\ y_k \\ \phi_k \end{bmatrix} \tag{2.2}$$

The specific state vector used in this project follows directly from the motion model used (which is a reduced *Coordinated Turn* model). This includes an $x_k$ and $y_k$ coordinate and also a $\phi_k$ yaw angle. More details on this can be found in section 3.4.1, which more discusses the implementation.

### 2.3.4 State prediction

To predict the state of a system for a future point in time a motion model is used. The motion model mathematically describes how the states of a system behaves given some inputs and time passed. Since it is often practically impossible to create an exact model of a system an approximation is often used. A simplified approximation is usually also simpler and faster to use in calculations.

A common practice when modelling systems is to select one of several standard motion models. Some of the most common models are: Random Walk (RW), Constant Velocity (CV), Constant Acceleration (CA) and Coordinated Turn (CT). These can be represented in both continues time and discrete time. Since the discrete form is more useful in practice when using computers, that is how they are represented here.

RW models a system as only having velocity of a stochastic variable (noise) $\xi_k$ that can be approximated as a normal distribution with zero mean and variance $\sigma^2$, see (2.3) [14].

$$\text{RW: } \begin{bmatrix} x_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \end{bmatrix} + \xi_k \qquad \xi_k \sim N(0, \sigma^2) \tag{2.3}$$

This is a very simple model which only uses position as states. It can be used to simulate a wide variety of system but will likely suffer in accuracy since it is so simple.

CV adds velocity as states and instead models the accelerations as a stochastic variable (noise) $\xi_k$ that can be approximated as a normal distribution with zero mean and variance $\sigma^2$, see (2.4) [14].

$$\text{CV: } \begin{bmatrix} x_{k+1} \\ \dot{x}_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \Delta T \cdot \dot{x}_k \\ \dot{x}_k \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \xi_k \qquad \xi_k \sim N(0, \sigma^2) \tag{2.4}$$

This makes it slightly more complex and suitable for modelling systems which rarely changes its velocity. The term $\Delta T$ here refers to the sampling time, the time between measurements.

CA adds acceleration to the states of the model and in turn models the rate of change of acceleration (referred to as jerk) as a stochastic variable (noise) $\xi_k$ that can be approximated as a normal distribution with zero mean and variance $\sigma^2$, see (2.5).

$$\text{CA: } \begin{bmatrix} x_{k+1} \\ \dot{x}_{k+1} \\ \ddot{x}_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \Delta T \cdot \dot{x}_k + \frac{\Delta T^2}{2} \cdot \ddot{x}_k \\ \dot{x}_k + \Delta T \cdot \ddot{x}_k \\ \ddot{x}_k \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \xi_k \qquad \xi_k \sim N(0, \sigma^2) \tag{2.5}$$

These three models are very similar in structure and can be seen as the same model but with different levels of complexity as higher order derivatives are added to the states [14].

The last model CT is slightly different since it includes the angle (or orientation) of the system. See (2.6) for a two-dimensional example.

$$\begin{gathered} \text{CT: } \begin{bmatrix} x_{k+1} \\ y_{k+1} \\ v_{k+1} \\ \phi_{k+1} \\ \omega_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \Delta T \cdot v_k \cdot \cos\phi_k \\ y_k + \Delta T \cdot v_k \cdot \sin\phi_k \\ v_k \\ \phi_k + \Delta T \cdot \omega_k \\ \omega_k \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \xi_k \qquad \xi_k \sim N(0, Q) \\ Q = \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{bmatrix} \end{gathered} \tag{2.6}$$

As seen in the equation this model also includes the linear velocity $v_k$ and angular velocity $\omega_k$ in the states. These two states are also modelled as constant with added noise $\xi_k$ that can be approximated as a normal distribution with zero mean and covariance $Q$. This model is sometimes considered as a good description of how vehicles such as cars move because of the turning motion [15, 16].

All four models presented have their separate advantages and disadvantages and it is also possible to create hybrid models with different combinations of states. It is of course also possible to add other states such as roll, pitch and certain biases. A rule of thumb is to keep the motion model as simple as possible to reduce complexity and computation time.

The specific model used in the project implementation is a reduced version of the CT model. The difference is that the linear velocity $v_k$ and angular velocity $\omega_k$ are not used as states and are instead simply used as inputs. More details on that model can be found in section 3.4.2 in the implementation chapter.

### 2.3.5 Measurement update

To perform a measurement update a measurement model is required. A measurement model essentially mathematically describes how measurements relate to the states of the system.

How a measurement model is constructed highly depends on the system it is meant to describe. But in general what one wants to have is the so-called posterior. The posterior $p(x|y)$ is the probability of $x$ given $y$, where $x$ could be seen as the position of a car and $y$ as measurements.

According to Bayes theorem the posterior can be calculated as in (2.7).

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{2.7}$$

Where $p(y|x)$ is the probability of measurements $y$ given state $x$, $p(y)$ is the probability of measurement $y$ and $p(x)$ is the probability of state $x$, also called the prior. If $p(y)$ is considered to be fixed, then the equation can be simplified into the form in (2.8). We consider it fixed since $p(y)$ does not change during any one point in time and so we vary over states not measurements.

$$p(x|y) \propto p(x) \cdot p(y|x) \tag{2.8}$$

You then have to find or select a usable equation to approximate the posterior with. This can be a difficult process so making assumptions and simplifications might be relevant, if not necessary. The specific equation to approximate the prior used in this project is introduced and discussed in section 3.4.3.

### 2.3.6 Extracting state estimates from PF

To get an estimate of the states from the particle filter the particles and weights, which together essentially make up a likelihood distribution, have to be combined to give an estimate of the "true" state of the system. This can be done in different ways, and which way that is best depends on many factors, for example what type of system and how it behaves.

A simple way of estimating the states is to compute a weighted mean of the states from the particles and with their corresponding weights as seen in (2.9). This can be interpreted as calculating the *Minimum Mean Square Error* (MMSE) value [17]. It is important to note that the sum of the weights are equal to one so the sum does not need to be divided by the number of particles $N$.

$$x_k^{\text{MMSE}} = \sum_{i=1}^{N} w_k^{(i)} x_k^{(i)} \tag{2.9}$$

When the weighted mean estimation is calculated the covariance of the estimation can be calculated according to (2.10).

$$cov(x_k^{\text{MMSE}}) = \sum_{i=1}^{N} w_k^{(i)} (x_k^{(i)} - x_k^{\text{MMSE}})(x_k^{(i)} - x_k^{\text{MMSE}}) \tag{2.10}$$

But using a weighted mean is not the only way, another example is to use a "Maximum A Posteriori" (MAP) estimator [18]. The formula for that can be seen in (2.11). What it does is to essentially take the value of $x$ (the state) which gives the highest peak in the approximated posterior distribution. In practice this could mean taking the particle which has the highest weight.

$$x_k^{MAP} = \arg\max_{x_k} p(y_k|x_k) \tag{2.11}$$

However the MAP estimation can be more difficult to implement well compared to MMSE. For example if the results are somewhat noisy the peak of the estimated posterior might move around while the "mean" (MMSE) might mostly remain still (less noisy). The MMSE method might therefore be better to implement for more consistent end result then the MAP. But as always this heavily depends on the situation and it is a design choice you need to do depending on how you evaluate and what you believe is most important and what kind of models you have.

### 2.3.7 Resampling

An important mechanic of the PF is resampling. If the filter is allowed to run for an extended period of time often only a few or no particles have any significant weight left. This stops the filter from giving reasonable results. To combat this resampling is often used. This means that when pre-decided conditions are met, the particles are resampled.

Resampling is usually done by drawing new particles from the old particles, where a higher weight gives a higher chance of that particle being resampled. But sometimes other methods are also used, such as drawing some particles from a GPS measurement to make sure the filter can't stray too far from the "true" position.

The conditions to perform resampling can for example be that a fixed amount of time has passed since the last resampling or possibly when a certain amount of particles with non-zero weight remain.

## 2.4 Coordinate systems

This section explains the four different coordinate systems relevant in this report.

### 2.4.1 Global geographical system

The geographical coordinate system is very common in use, both now as well as historically. It indicates a position with three (sometimes four) quantities [19]. These are latitude, longitude and altitude (and possibly heading).

Latitude gives the position along a north-south axis between the geographical poles. See figure 2.3 for a visualisation, where the blue lines indicate a constant latitude. Latitude position is divided into 180 degrees with 0° at the equator and ±90° at either poles.

Longitude is the position along the equator which runs around the globe. Also see figure 2.3 for this, where red lines indicate constant longitude. Longitude is divided into 360 degrees with $0^o$ (also called the Prime Meridian) running through Greenwich observatory in London. Positions west of the prime meridian are given in the range [0°,-180°] and to the east of it in [0°,+180°].

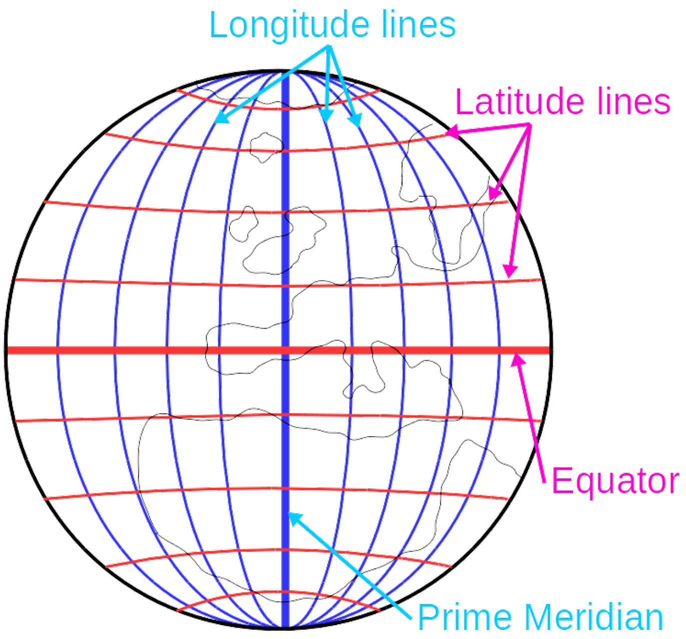


**Figure 2.3:** The global geographical coordinate system. Longitude lines are shown in blue and latitude lines are shown in red. (Coastlines are not entirely accurate.)

Latitude and longitude can be given either in exact decimal degrees such as [57.688194° N, 11.978722°E] or divided into [*degrees*° *minutes*' *seconds*"] such as [57°41'17.5"N, 11°58'43.4"E]. The second format uses arc-minutes and arc-seconds and is often used by humans when navigating. However the first format is arguably more practical when using computer algorithms since less unit conversions are necessary. Therefore that format is used in this project.

The altitude is given in meters and is often given in relation to an ellipsoid which represents the average surface of the planet. This is done since the planet is not an exact sphere, but rather is more ellipsoidal in shape. Which type of ellipsoid and its exact measurements are slightly different depending on the implementation or definition used.

When a heading is included it is given as a compass heading. This means that the heading is the angle (in degrees) from the northern direction and measured clockwise to the direction a person/object is facing. This is opposed to the most common way of indicating angles in mathematics, which is the angle from positive x-axis (or eastern direction) and measured counter-clockwise (a right-handed system).

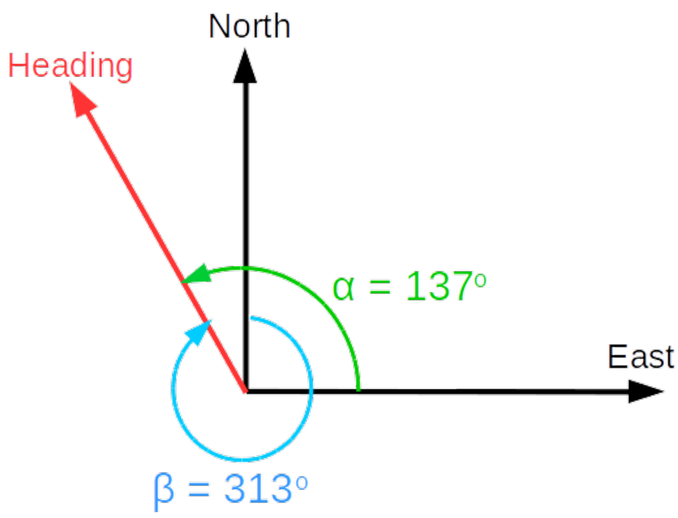


**Figure 2.4:** An example of a right-handed angle $\alpha$ in green compared to the corresponding geographical compass heading $\beta$ in blue.

An example of this is with a "mathematical" angle $\alpha$ of 137° (counter-clockwise from east) which would as a geographical compass heading $\beta$ be 313° (clockwise from north). See figure 2.4 for a visualisation of this example, and also (2.12) for how the calculation is done.

$$\beta = 90° - \alpha = 90° - 137° = -47° = -47° + 360° = 313° \tag{2.12}$$

One important point with the geographical system is that generally one degree in latitude does not equal one degree in longitude in terms of arc distance (in meters and along the surface of the planet). Close to the poles the longitude lines are much closer then they are at the equator. This also means that the longitudes lines are not truly parallel lines.

### 2.4.2 Global Cartesian system

The "global Cartesian coordinate system" in this report is a Cartesian approximation of the global geographical system. This means that instead of using latitude and longitude with degrees as unit, a position is instead given with x- and y-coordinates and with meters as the unit. This approximation basically assumes that the planet is completely flat or elliptical cylinder around a certain point of origin (the origin of the Cartesian system). But since longitude lines are (generally) not parallel and since the planet is curved this system has limitations.

It can not be used for distances that are far away from the origin since the errors then become significant enough to be of a problem. It is however very useful for more local (within some km's) areas where the error contributed from converting to this coordinate system is very minor. Since the surface is assumed to be flat it is then very easy and fast to calculate for example the Euclidian distance between two points.

See figure 2.5 for an example of how the coordinate system looks when applied on a city. Lines on maps are often displayed with slight curves since that is how the latitude and longitude lines behave, but on this map the lines are completely straight. This illustrates well how this coordinate system approximates the planet as flat.

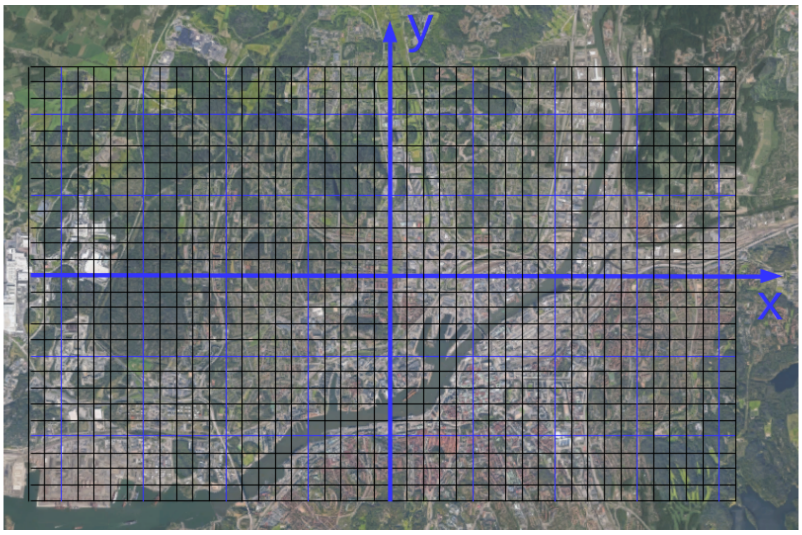


**Figure 2.5:** Example of the global Cartesian coordinate system, using a satellite image of Gothenburg. Positions far from the origin will suffer from increasing error as distance to origin increases.

There exist multiple methods which create a Cartesian coordinate system from a geographical one and depending on the applications they have varying advantages and disadvantages. One of the more common types of map projection is the *Transverse Mercator*.

There in turn exist multiple different methods for the *Transverse Mercator* projection and they all have similar characteristic and formulas. One is the Gauss-Krüger (Gauss conformal projection) coordinate system that gives accurate results given that the origin is fairly close to the position of interest. This one is used in the report.

The equations for performing this map projection from the global geographical coordinate system to the global Cartesian coordinate system can be viewed in appendix A. The code for these calculations was not created during this project. Already existing implementations were used for that.

### 2.4.3 Local vehicle system

To handle positions of sensors, objects or measurements in relation to the vehicle a definition for a local coordinate system is required. This local coordinate system is a right-handed Cartesian system. The origin is located in the centre of the rear axle of the vehicle, with positive $x$-direction towards the front of the car, positive $y$-direction to the left and positive $z$-direction straight upwards. See figure 2.6 for visualisation. The angles yaw, pitch and roll are defined around the three axes as given in the figure.

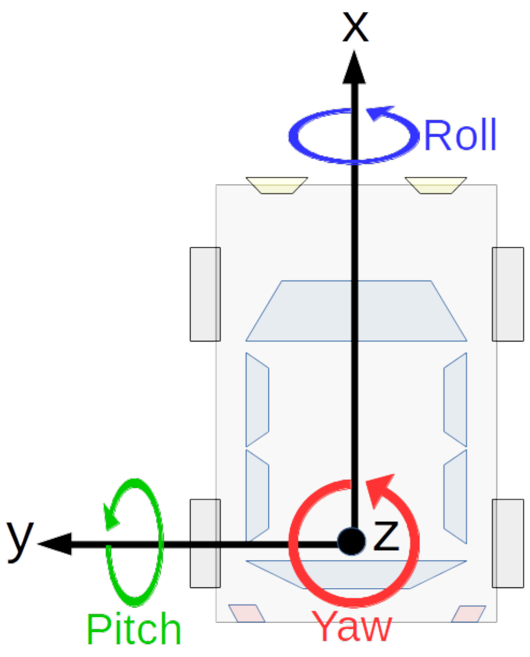


**Figure 2.6:** Local vehicle coordinate system according to ISO 8055 [20]. The origin is placed in the middle of the back-axle of the vehicle.

This defined system is standardised in accordance with ISO 8855:2011 [20], which is a standard specifically meant for road vehicles such as cars and busses. The units of measurement are meters for distances and radians for angles.

If any sensor has its own local coordinate system, with for example origin in the sensor, it needs to be transformed to the standardised local coordinate system. This can be done by performing a 3D affine transformation which transforms from one coordinate system to another. Both spatial translations and rotations can be included in this.

### 2.4.4 Road based coordinate system

The road based coordinate system is a system which gives the position of a vehicle in relation to the road. The position is given as a *lateral* and *longitudinal* value. See figure 2.7 for an illustrative example of this system.

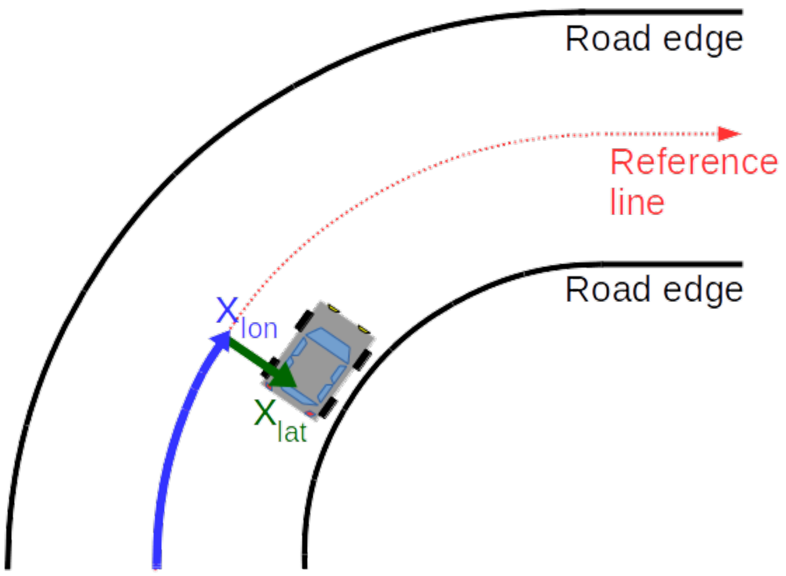


**Figure 2.7:** Road based coordinate system. $X_{lon}$ is the longitudinal position along the reference line and $X_{lat}$ is the lateral position orthogonal to the reference line.

The lateral position is the position of the vehicle on an axis which is always orthogonal to the direction of the road. This means that it is very easy to tell how close the vehicle is to the centre or sides of the road. It is also easy to tell which lane the vehicle is in.

The longitudinal position indicates the position of the vehicle along the road being driven upon. This follows the curvature of the road and is a very handy way of telling for example how far the vehicle has travelled. An absolute position can be problematic to give since a zero (or reference) point must be defined. Therefore this system is more easily used for relative distances. When an absolute position is required, the starting point of the current road can for example be used.

The "reference line" in figure 2.7 is only intended as an example of how the position can be given. In practise the longitudinal and lateral positions could be given in relation to any line or point on the road. In the evaluation of the implemented algorithm the position errors are actually given in relation to the "ground truth" positions of the vehicle, which is very accurate position data.

## 2.5 Miscellaneous

This section provides theory on various topics too small for separate sections.

### 2.5.1 RANSAC

RANSAC stands for Random Sample Consensus and was introduced by Fishler and Bolles [21]. The RANSAC algorithm can be used to identify different geometric shapes in a set of measurement. Which can be useful for example outlier detection or identifying planes in the data set.

In this project RANSAC is specifically used to identify the road surface which the vehicle is driving on. By approximating the road as a flat plane the algorithm can search for that type of geometric shape. If there is some difference between the vehicle and road surface, such as in rotation or distance, the algorithm should help measure that so it can be corrected later.

The algorithm works by iteratively looking for measurements in a data set that fits with the specified geometry. This is done by first looking at a small section in the data set and then iteratively expanding and shifting the section with measurements that are consistent with the specified geometric shape. Then it uses a smoothing technique to compute and improve the estimation that describes the geometry [21].

*Matlab* have a function *pcfitplane* [22] that uses a version of RANSAC called MSAC (M-Estimator Sample And Consensus) that can identify geometric shapes, the M-estimator in MSAC is a redescending M-estimator. RANSAC penalise the outliers that do not fit the specified geometric shapes, while MSAC penalises the outliers and scores inliers. Removing the chance that all the solutions which fit a specific geometric shape could be scored equally. Thereby giving one solution instead of multiple [23].

### 2.5.2 Median-Estimator

A median-estimator is a version of an M-estimator which is a practical method that can be used to deal with outliers and occlusions [24]. For example, when comparing a map and a LIDAR point cloud generated from on-board sensor, there might be objects in the LIDAR data that is unwanted. Passing cars could be an example of this.

The LIDAR point cloud can have multiple outliers that do not fit the map and also occlusion such as cars. The median-estimator can then be applied to lower the effect of these outliers and occlusion by dampening the effect of the measurement that do not fit the model. This is done by calculating the median of the error between measurement and model and then weigh down the measurements that are above the median error. Thereby decreasing the outliers and occlusions impact [24].

$$\begin{gathered} b = median(e_a) \\ c_k = \begin{cases} 1 & if \quad e_k \leq b \\ 0 & if \quad e_k > b \end{cases} \end{gathered} \tag{2.13}$$

In (2.13) describes the equation used for a median-estimator. Variable $e_a$ are all the error values, the differences between model and measurements. First the median is calculated for all the error values, then each independent error value($e_k$) are weighted. If they are above the median they are weighted to zero and if they are below they are weighted to one. This will filter out some, but not all, occlusions and outliers.

### 2.5.3 Affine coordinate transformation

To transform between two different Cartesian coordinate systems one can use an affine coordinate transformation. Since in this project the Cartesian systems only differ in the location and orientation of the origin it is a fairly simple operation. A more complex operation would for example be a coordinate system with non-orthogonal axes.

If a coordinate is represented as $X = [x, y, z]^T$, the transformation can be performed as seen in (2.14) [25]. The extra 1's in the equation are only added to allow for the translation $T$. In practise it is only added at the last moment before making the transformation and the "1" in the result is thrown away.

$$\begin{bmatrix} X_{new} \\ 1 \end{bmatrix} = [R_x \cdot R_y \cdot R_z \cdot T] \cdot \begin{bmatrix} X \\ 1 \end{bmatrix} \tag{2.14}$$

The objects $R_x$, $R_y$, $R_z$ and $T$ represent three rotation matrices (around the x-, y- and z-axis) and one translation matrix. A rotation matrix rotates the defined coordinate system around an axis while the translation matrix shifts the coordinate system in some direction.

The different rotation matrix can be seen in (2.15). $R_x$, $R_y$ and $R_z$ are counter clockwise rotation around the x-, y- and z-axis, respectively. The arguments A, B and C describes how much rotation (in radians) to be performed in each rotation, this rotations follows ISO 8855 [20].

$$
\begin{aligned}
R_x &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos A & -\sin A & 0 \\ 0 & \sin A & \cos A & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_y = \begin{bmatrix} \cos B & 0 & \sin B & 0 \\ 0 & 1 & 0 & 0 \\ -\sin B & 0 & \cos A & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
R_z &= \begin{bmatrix} \cos C & -\sin C & 0 & 0 \\ \sin C & \cos C & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad T = \begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
\tag{2.15}
$$

The translation matrix can be expressed as also seen in (2.15). Here $\Delta x$, $\Delta y$ and $\Delta z$ are translation in the x-, y- and z-axis.

By multiplying the rotation and translation matrices a transformation matrix $M$ is acquired, see (2.16). The transformation matrix can consist of multiple rotations and translations.

$$
M = R_x R_y R_z T \tag{2.16}
$$

# 3 Implementation

This chapter describes the implementation of the localisation algorithm. First a system overview is presented, followed by a description of the sensors and map layers used. Then the pre-processing of the LIDAR sensor data is described, followed by the main localisation algorithms and the position estimation. The final section describes implemented utility functions which connect coordinates systems for different parts of the algorithm.

## 3.1 System overview

Figure 3.1 shows a simplified view of how the filter algorithm flows and how the different packages are combined. A more complex diagram can be viewed in figure 3.5.

Firstly the sensor data and map layers are pre-processed. The map layer only needs to be pre-processed when a new RoadDNA map layer is loaded. But the sensors, especially the LIDAR sensor, needs to be pre-processed every time new data is received. And this essentially happens continuously.

The particle filter "module" contains three sub-modules: state prediction, measurement update and resampling. Depending on which sensor data is available at time $t_k$ a prediction or a update step is made, and resampling is performed at fixed intervals. The particle filter calls on different utility functions depending on which of the sub-modules in the particle filter is running. They perform such tasks as projecting points onto the reference line, or transforming between coordinate systems. The last part of the filter (called position estimation) takes the results from the particle filter and uses them to give an usable estimate of the vehicles position.

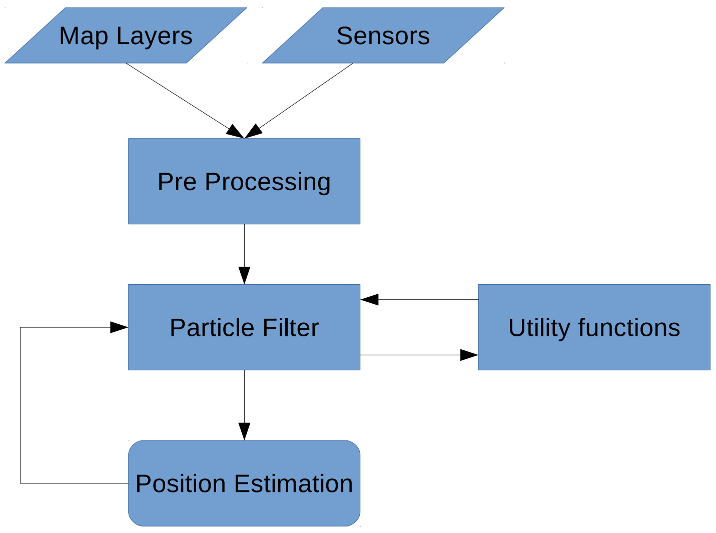


**Figure 3.1:** A simplified flowchart of the localisation algorithm.

## 3.2 Sensor and map data

The different sensors used in this implementation are as described in Chapter 2: RT3000, Velodyne HDL64E and odomentry (*car signals*). For optimal utilisation of the different sensors and the corresponding data it is important to get a general understanding of them. The different sensors, their data structure, units and the sheer amount of data. The following paragraphs will describe what data from the sensors that is vital and also the overall data structures.

The RT3000 is a highly accurate sensor that provides a multitude of different sensor data but the most important is the geographical positions (latitude, longitude and altitude) and the heading (yaw angle) in degrees as described in section 2.4.1, these sensor values are ground truth values. The sensor data from RT3000 is concatenated with the corresponding on-board Velodyne 64-E sensor data that provides a local point cloud described in section 2.1.1. The Velodyne and RT3000 sensor data is also concatenated with the *car signal* sensors described in section 2.1.3. Every sensor measurement has a timestamp that is added to the data structure.

The amount of data depends heavily on how fast the different sensors provides information. The frequency at which the LIDAR sensor provides information is approximately 10Hz, while the car signals and the RT3000 has a significant higher frequency at approximately 40Hz. The LIDAR sensor provides the largest amount of data even though it has a lower frequency. Every work file (log file) contains around 600 complete LIDAR scans (frames) to limit the size of the file. The work file corresponds to approximately one minute of drive time.

The map layers provided from TomTom has a different data structure, the map layer is known commercially as RoadDNA. RoadDNA consist of two major components

RoadDNA image and reference line described in section 2.2. Every RoadDNA image corresponds to a specific reference line and when they are linked the reference line provides the global position while the images provides the local surroundings. The RoadDNA image can be compared to a local point cloud given some prior knowledge of your global position.

## 3.3 Pre-processing

Before the filter can be implemented some vital pre-processing is needed for the on-board sensors. The sensor that requires pre-processing is the LIDAR sensor. Every LIDAR frame contains a complete 360 degrees rotation scan. There is multiple step to the pre-processing of the point cloud as described in the following list:

- Untwisting
- Remove outliers
- Identify road plane
- Transform point cloud (align point cloud to road plane)

Each LIDAR scan in every frame is first untwisted, which means you factor in the velocity of the LIDAR sensor during a scan. When the LIDAR sensor data is processed a assumption is made that it is static (not moving). Therefore you end up with a local Cartesian coordinate system where the origin is the position of the Velodyne sensor at a certain time, and every measurement is in respect to the Velodyne sensor position at that time. But in reality the Velodyne senor is usually moving while scanning therefore the points in the point cloud needs to be adjusted accordingly.

The LIDAR scans contains a multitude of outliers in a circle around the Velodyne sensor and detection's of the hood of the ego-vehicle. These outliers need to be removed, since they contain no useful information and the circle that appears around the Velodyne sensor is probably due to how the sensor is mounted on top off the car. The removal of theses outliers is quite simple. Detect and remove all the points that is within a certain area of the car. A good area is the dimensions of the ego-vehicle or something slightly bigger.

After the standard untwisting and calibration is made there is a need to locate the road plane, which corresponds to the road surface. It is a plane with a normal vector close to $[0 \quad 0 \quad 1]$ (in the vehicles local coordinate system) with some angular-tolerance. This tolerance is basically the difference in angle between the road surface and the LIDAR sensor on the vehicle, such as when the vehicle is leaning in some direction. "Leaning" could for example be caused by the vehicle accelerating, braking or turning.

The road plane is approximated with MSAC. MSAC is a version of RANSAC explained in section 2.5.1 but the implementation is made with the help of the function *pcfitplane* [22] in *Matlab*.

When the normal vector to the plane is identified the Velodyne point cloud is rotated in roll and pitch angle with the corresponding angles calculated from the identified plane. The origin is also moved down to the road plane's z-value according to (3.1). The movement of the origin to the road plane simplifies the correlation between RoadDNA and the on-board LIDAR sensor's point cloud.

$$
\begin{aligned}
&plane = [e_x \quad e_y \quad e_z \quad d] \\
&\phi = \arcsin(-e_x), \quad \theta = \arcsin(e_y) \\
&M = \begin{bmatrix} \cos(\phi) & 0 & \sin\phi & d\sin\phi \\ \sin\theta\sin\phi & \cos\theta & -\sin\theta\cos\phi & -d\sin\theta\cos\phi \\ -\cos\theta\sin\phi & \sin\theta & \cos\theta\cos\phi & d\cos\theta\cos\phi \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
\tag{3.1}
$$

In (3.1) *plane* is the normal vector for the plane ($e_x, e_y$, $e_z$ and $d$) that describes the road surface, in this case something that is close to $[0 \quad 0 \quad 1]$. The variable d is the distance from origin to the plane, $\theta$ is the roll angle and $\phi$ is the pitch angle for the road plane. M is the final transformation matrix, which consist of a rotation in roll and pitch and a translation down to the road plane. By rotating the point cloud the the road plane becomes straight with minimal difference in z-elevation. The end result is a point cloud with the road plane at elevation $z = 0$ and all the measurements in the point cloud is aligned with the road plane.

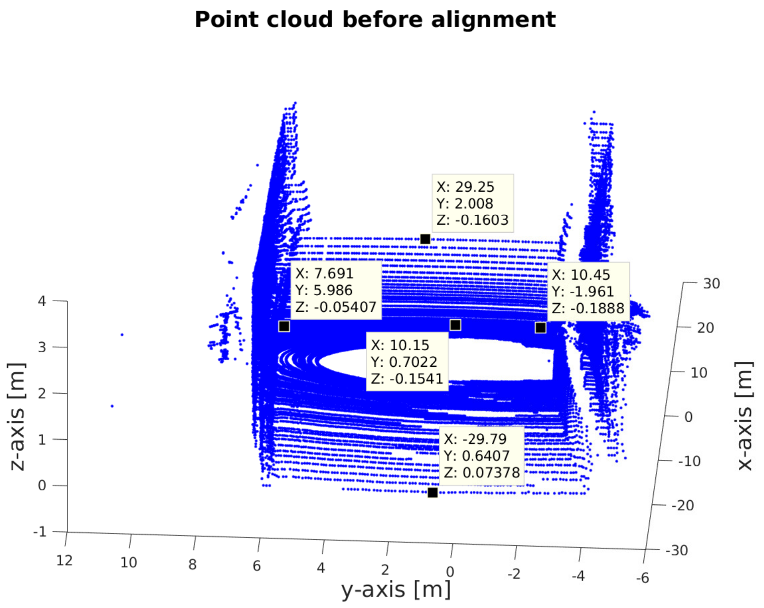


**Figure 3.2:** Shows the point cloud before alignment. As seen the difference in z-value are higher at the edges in the road then when not performing alignment compared to when performing alignment figure 3.3.

An illustration of the difference between not aligning the point cloud and when aligning the point cloud with the road can be seen in figure 3.2 and 3.3. In figure 3.2 the point cloud has in general a bigger difference in z-value which can be contributed to the chassis position when driving the car. Example if the car has a high acceleration the point cloud will be biased in pitch angles even though the road might be straight. While in figure 3.3 the point cloud is levelled out and now more aligned then the original point cloud.

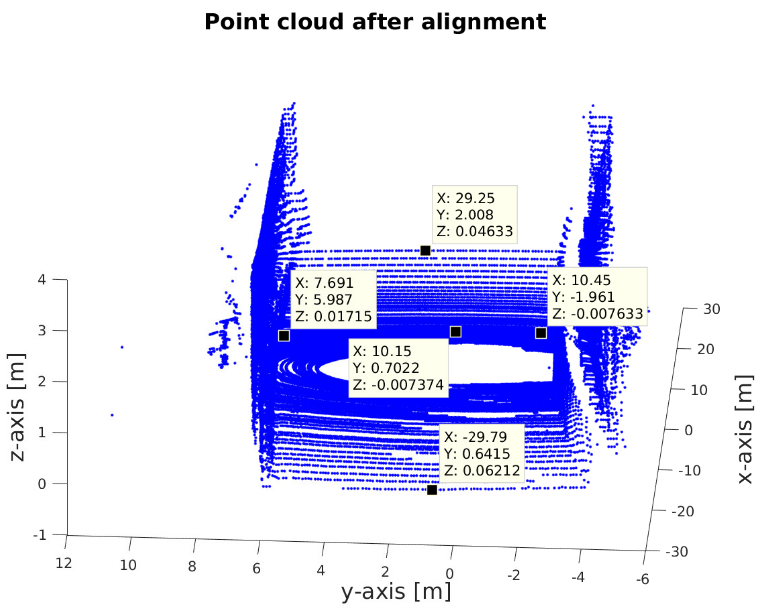
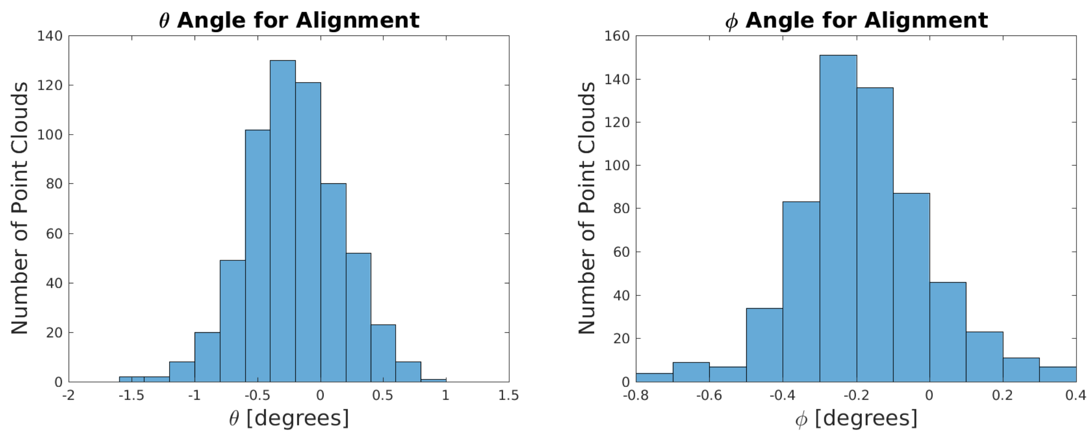


**Figure 3.3:** Shows the point cloud after alignment MSAC and transformation matrix is applied to point cloud. Now the z-values are in general closer to zero and the road is aligned, lower pitch and roll effect on the point cloud compared to figure 3.2.

When calculating $\phi$ and $\theta$ for all the point clouds in one log file a bias can bee seen displayed in figure 3.4b and 3.4a. This means that the point clouds are constantly shifted to one side. This can be due to road condition but also with the calibration of the point cloud and the movement of the car.

Before the filter starts all the different global geographical position's need to be converted to the global Cartesian coordinate system. The global Cartesian coordinate system has an origin close to the trajectory to minimise offset errors that can occur during transformation from different coordinate systems. The data that is converted is the reference line from RoadDNA, and the geographical positions from RT3000. The conversion is done with the utility function described in section 3.5.2. If the RoadDNA sequence is known the whole trajectory can be converted and because RT3000 is only used for evaluation and initialisation it does not misrepresent the filter's calculation time if the whole true trajectory is converted before the filter starts.



**(a)** The $\theta$ angle when using MSAC for alignment.

**(b)** The $\phi$ angle when using MSAC for alignment.

**Figure 3.4:** The alignment angles when using MSAC to identify the road plane.

The RT3000 also gives a heading angle in the format geographical compass heading described in section 3.5.2. This heading needs to be converted to a right-handed angle so that rotations can be made when evaluating the local error. This is done accordingly to as described in section 3.5.2.

After conversion to the Cartesian coordinate system the reference line from RoadDNA is interpolated. The interpolation is needed so that every pixel in the RoadDNA image correspond to a specific position in the Cartesian coordinate system. Otherwise it becomes very hard to associate the reference line with the RoadDNA image. Now every coordinate in the reference line has a index that corresponds to a pixel in the image. This makes it easier to search and connect the image with the reference line.

## 3.4 Particle Filter

The implemented particle filter estimates the ego-vehicle position given available sensors at a certain timestamp. The majority of the work is how to associate RoadDNA with the LIDAR sensor for the update step.

Figure 3.5 shows the overall design of the implemented particle filter. The filter is initiated with ground truth GPS position and heading sensor value from the RT3000 at a given timestamp. After initialisation either a prediction or a update step is made depending on the sensors. The decision between update or prediction step is made by comparing timestamps from the LIDAR and *car signals*. The filter doesn't know which sensors will provide the newest set of information. Therefore depending on which sensor that provides the newest set of information decides if a prediction step or update step is made. If the *car signals* has the next chronological timestamp the prediction step is made and if the LIDAR has the next chronological timestep the update step is made. Because *car signals* has a higher frequency than the LIDAR sensor this means that there are multiple small prediction step between every update step. This approximation is equal to making one longer prediction step followed by a update step. The update step in the particle filter gives the output result and is followed by a resampling step as illustrated in figure 3.5.

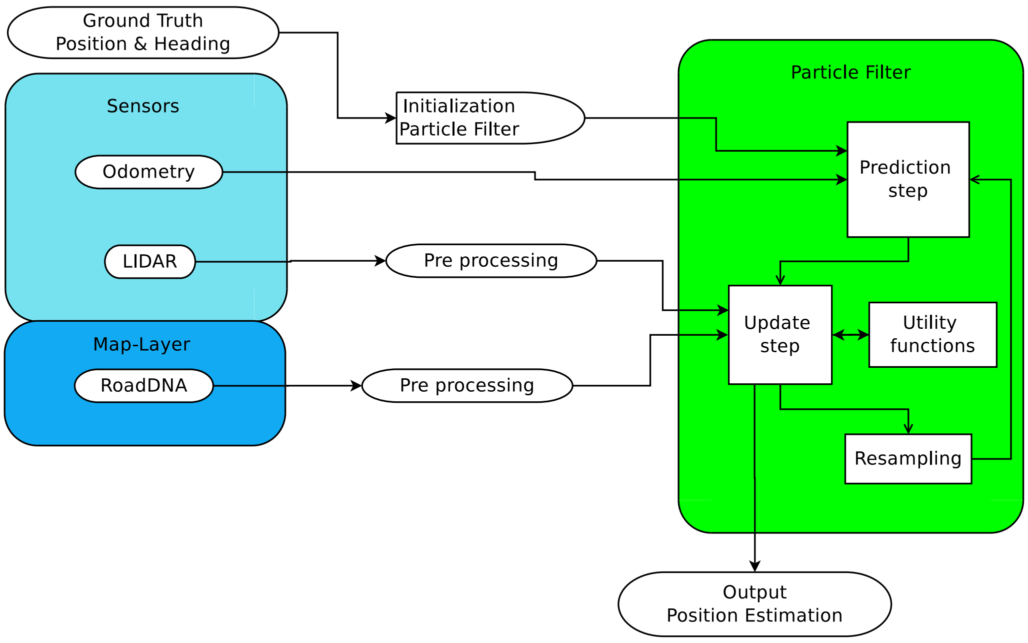


**Figure 3.5:** The flowchart starts with a initialization with exact information for the states. Then the different timestamps from the sensors decide the action between update and prediction step in the particle filter, the update step uses LIDAR and RoadDNA while the prediction uses odometry. After every update step a the position estimation values are taken and then a resampling step is made.

### 3.4.1 State Space

The state space is decided depending on what sensors are available and what measurement- and motion-model that is used. Another factor that is contributing to decide the size of the state space is the computational speed of the filter. Less states mean fewer calculations in every prediction and update step which leads to a faster filter, but a drop in accuracy can occur.

The implemented state space can be seen in (3.2) where $X_k$ and $Y_k$ is the Cartesian coordinates in the global coordinate system given in meters and $\phi_k$ is the heading/yaw angle (rotation around Z-axis) given in radians. Because the road plane can be identified with *pcfitplane* [22] algorithm the z-state is not needed. The motivation for this is that usually the car is driving on the Road plane and is not in the air.

The alternative is to have z as a state. But this results in a state that is very hard to model with the sensors available. The alternative is then a random walk motion model for the z-axis and to keep the existing update step. This will lead to the need of more particles and the uncertainties and errors will increase. The usage of *pcfitplane* [22] is a more robust approach and generates a lower error with the same number of particles. This behaviour can be seen in appendix B.

$$\begin{bmatrix} X_k \\ Y_k \\ \phi_k \end{bmatrix} \tag{3.2}$$

### 3.4.2 Prediction

With the given state vector in (3.2) the following motion model described in (3.3) for the prediction step seen in the figure 3.5 is implemented. In this motion model (inspired by coordinated turn section 2.3.4) the car signals yaw rate, velocity and the previous yaw angles from the update step are used, also a process noise is added corresponding to a percentage of the car signals.

The reason for using coordinated turn is that it can predict the kind of behaviour that is expected of a car and the model corresponds well with the sensors that are available. The process noise helps the particle to explore new states so it can adapt to changes in the trajectory fairly quickly. The process noise can be tweaked but they depends heavily on which sensors are used and how many particles that are needed. The number of particles used in this implementation is primarily 200 (but 500 particles were also tested). The reason for the number of particles has to do with computation time, more particles means a much higher computational time. There seams also to be a minimal effect on the lateral error, but there is a better performance in the longitudinal error as seen in section 4.3.2.

The process noise is represented as a normal distribution with mean at zero ($\mu_k$) and a covariance matrix $Q_k$ as seen in (3.4) but can be optimised. The optimisation that can be made is the tweaking of the process noise so that it is optimal for the sensor that is used in the car. The implementation uses a process noise that works and seems to perform decent but it is based on an approximation that the *car signals* has an error which is a percentage of the total value that the sensor provides.

$$\begin{bmatrix} X_{k+1} \\ Y_{k+1} \\ \phi_{k+1} \end{bmatrix} = \begin{bmatrix} X_k \\ Y_k \\ \phi_k \end{bmatrix} + \begin{bmatrix} \Delta T \cdot v_k \cdot \cos(\phi_k) \\ \Delta T \cdot v_k \cdot \sin(\phi_k) \\ \Delta T \cdot \omega_k \end{bmatrix} + N(\mu_k, Q_k) \begin{bmatrix} \cos(\phi_k) & 0 & 0 \\ 0 & \sin(\phi_k) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$Q_k = \begin{bmatrix} 0.1 \cdot v_k \cdot \Delta T & 0 & 0 \\ 0 & 0.1 \cdot v_k \cdot \Delta T & 0 \\ 0 & 0 & 0.1 \cdot \omega_k \cdot \Delta T \end{bmatrix} \quad \mu_k = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \tag{3.4}$$

In (3.3) the states are the same as described in the state vector and $\Delta T$ is the time in seconds from one time stamp to the next time stamp. Variable k denotes the current discrete time, and k+1 denotes the next discrete time in this case the next predicted state vector. The variable $\omega_k$ stands for the angular velocity and $v_k$ stands for the longitudinal velocity.

There exist a multiple of reasons for why we decided to have independent covariance for $X_k$ and $Y_k$ in (3.3) and (3.4) (two independent noise models for the velocity). The biggest reason is that it is a design choice for convenience when testing the update function which is the main focus. With this design we can test large longitudinal or lateral errors independently to see how the update function behaves and in the process try and optimise it, also debug code and fix special cases that can. The independent changes of the noise model is made by changing the constant in this case 0.1 in $Q_k$, during the project this constant 0.1 was changed many times for testing special cases but when later presenting the result in this report the constant 0.1 was used. This helps us understand and further develop the update function and helps us explore and understand the RoadDNA map layer.

If you do not have to evaluate and learn about how the RoadDNA map layer behaves and at the same time test your update function for instabilities. Then use this knowledge to improve the update function or hammer out code bugs depending on special cases that can accrue there is a argument to be have. That instead of using two independent noise models for the velocity it is enough to have one dependent process noise model for the velocity.

How dependent the process noise should be on the sensor values is hard to define and depends on the sensor specifications. If the sensor values should be proportional to the standard deviation or the covariance is a good question. In our case this will mostly just affect the scaling factor 0.1 in (3.4), which would be changed to a appropriate value. What is best in the end would require allot of testing and consume allot of time therefore this design was used. Because we found a decent scaling factor that gave decent end results (estimation of the position) with a reasonable time spent on designing and testing the prediction model. Thereby in the end significantly decreasing the amount of time needed for testing and debugging.

The reason for this process noise is that we want the filter to handle changes by exploring multiple states especially in this grid world the particle filter will operate in. Big lateral displacements is not expected during a short amount of time but it can be tested with this process model.

### 3.4.3 Update

In section 2.3.5 in the theory chapter the problem of performing a measurement update was defined as approximating the posterior $p(x|y, m)$ as in (3.5). A difference from the equation in section 2.3.5 is that we have changed the equation slightly by adding the letter $m$, which in essence represents the digital map. This means that the results are conditioned on the measurements $y$ and the map $m$. But the map that we decide to look at depends on what position we have therefore we get $m(x)$.

$$p(x|y, m(x)) \propto p(x) \cdot p(y|m(x), x) \tag{3.5}$$

To perform a measurement update a measurement model must be created. One suggested approach is to approximate the posterior $p(x|y, m(x))$. The function $p(x|y, m(x))$ basically means the probability of having position $x$ given measurements $y$ and map $m$ at position $x$. The likelihood function is defined as (3.6) where $\lambda(y, m(x))$ is used for simplicity in further equations.

$$\lambda(y, m(x)) = e^{-\frac{(|m(x)-y|)^T R^{-1}(|m(x)-y|)}{2}} \tag{3.6}$$

In this case $m(x)$ specifically means the expected measurements which are taken from the map at position $x$, and $y$ means the received measurements from the LIDAR. These two sets of values indicate how well the current measurements correspond to the current map. $R$ is the covariance matrix for the measurements. It is considered as a tuning term since it it very complex to determine in any exact way.

The expected measurements $m(x)$ are taken from the RoadDNA map layer and are centred differently for each particle. The actual expected measurements are the 3D voxels that are occupied in the map. White pixels (which correspond to voxels) in the map are considered as unoccupied since we do not know if there really is nothing there or if the map is for example missing data there. The occupied voxels can then be compared with the measured LIDAR data $y$.

The likelihood function in (3.6) can be rewritten into a form more suitable for computation, which is given by (3.7). To perform this change two assumptions are made. The first one is that there are $N$ number of measurement. The second assumption is more serious, which is that the covariance matrix $R$ is diagonal. This implies that the measurements are considered completely independent to each other. That might not be entirely true in the "real world", but it is likely a good approximation. A "true" $R$ might be next to impossible to find since it likely depends on both time and space.

$$\lambda(y, m(x)) = e^{-\frac{\sum_{j=1}^{N} r_j(|m_j(x)-y_j|)^2}{2}} \tag{3.7}$$

The values computed by using (3.7) sometimes result in very small values. In some cases we received values of around $10^{-16}$. But we can be unlucky and get much worse lets say that all our measurement are 1m off and we have 2000 measurement after the median-estimator, this can lead to $e^{-1000}$. Which can result in problems when using computers since they use floating point variables to represent non-integer numbers.

To ensure that the computed numbers are kept at more reasonable values, the equation can be changed slightly into the form in (3.8). By dividing the sum in the exponent by $N$ this can be interpreted as using the mean of the measurements rather than the sum of them. It can also be seen as the $N$th root of the value $\lambda$.

$$\lambda^{\frac{1}{N}}(y, m(x)) = e^{-\frac{\sum_{j=1}^{N} r_j(|m_j(x)-y_j|)^2}{2N}} \tag{3.8}$$

This modified value is then used for updating the weights of the particles. This will change the scaling of the distribution. The highest peaks for the likelihood function should still be the highest after taking the root. You can also see the dividing by N in terms off interpreting all the measurement from one LIDAR scan and the corresponding map value as one single measurement and map. This will have effects on MMSE and/or MAP values when evaluating the particles which can be seen as a drawback depending on how much you can trust the map, the map is a black box which makes it very hard to determine how much you can trust it. But in this case it seams to work fine, because of the low errors we get in our result when estimating the position. There probably exist better methods to deal with this computational problem we have come across but that is a subject for future work, and is briefly discussed in section 5.4. The advantages of this method is that it works (relatively robust) and is easy to implement in our case.

To determine the region of interest in RoadDNA every particles is projected in global coordinates to the RoadDNA's reference line according to section 3.5.1. This returns the centre index of the RoadDNA image, the distance between the reference line and the particle and finally the yaw (heading) of the reference line. RoadDNA is then cut around the centre index, usually around 50m (25m on each side of centre index). The size of the slice is also a tuning parameter and depends on the point cloud and how fast the algorithm is supposed to be.

Before the differences between the RoadDNA map layer and the on-board LIDAR measurements can be calculated according to the update (3.8) the point clouds need to be transformed. This transformation consist of a rotation around the z-axis that is the difference in particle heading and reference line heading and a transformation that depends on the distance calculated from the projection and then the closest discrete pixel as seen in (3.9).

$$\begin{aligned} &\phi_\Delta = \phi_p - \phi_r \\ &T = \begin{bmatrix} \cos\phi_\Delta & -\sin\phi_\Delta & 0 & x_\Delta\cos\phi_\Delta - y_\Delta\sin\phi_\Delta \\ \sin\phi_\Delta & \cos\phi_\Delta & 0 & x_\Delta\sin\phi_\Delta + y_\Delta\cos\phi_\Delta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{3.9}$$

Where $\phi_p$ is the yaw angle state in the state space for the particle and $\phi_r$ is the reference lines yaw at the projected index. T is the transformation matrix with a counter clockwise rotation around z-axis and a translation. The $y_\Delta$-value is the distance between the particle states y-value and the reference line, this is calculated in the projection. Because the particles are not discrete values as the pixels in a image there is a need for a small adjustment in x which is $x_\Delta$ in (3.9). This adjustment correspond to the movement to the closest discrete pixel value in the RoadDNA image. The projection that calculated the $x_\Delta$ and $y_\Delta$ is visualised in figure 3.6.

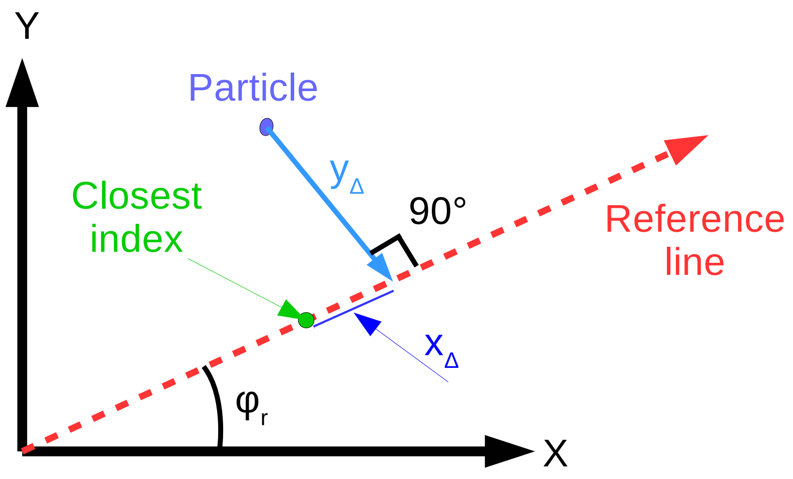


**Figure 3.6:** Visualisation of how $x_\Delta$ and $y_\Delta$ is acquired in (3.9) with the help off projection.

Now the local point cloud is correctly adjusted and finally we want our point cloud to have the same grid as the RoadDNA image so that we later can calculate the differences between the RoadDNA image pixel values and our point cloud values. This is done with a discretization of the point cloud so that it has the same grid as the RoadDNA image and then a projection of the discretezised point cloud to the RoadDNA reference line to calculate the distances. First the reference line from RoadDNA is moved to the local coordinate system that the point cloud is represented in. Then $y_{i,q,n}$ is calculated for every point in the point cloud as seen in figure 3.7. In $y_{i,q,n}$, i correspond to index on the x-axis, q to index in the z-axis and n represent all the measurements y in one x-, z-voxel. The index and $y_{i,q,n}$ are calculated with the help of the projection equation's described in section 3.5.1.

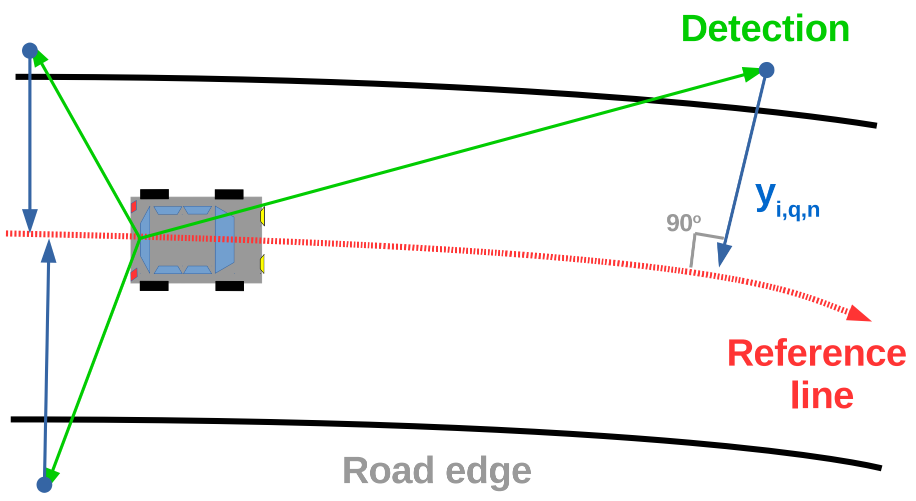


**Figure 3.7:** This picture describes how the LIDAR detect objects and how the measurement value $y_{i,q,n}$ is acquired

Now the point cloud and RoadDNA images are aligned and every pixel in RoadDNA (x, z) can be compared to the corresponding x-, z-voxel in the point cloud. The difference is then calculated by comparing the RoadDNA's pixel value and the corresponding closest point in the point cloud, if no value is found in the point cloud the difference is calculated to NaN, the way this is calculated can is represented with pseudocode in algorithm 1 and it gives the lowest possible error for the whole difference matrix ($diff_{i,q}$). This is done for every pixel which leads to a difference matrix ($diff_{i,q}$) with size (width, height) and it is also done for every particle in the particle filter. Every particle has shifted the point cloud slightly different because of their state vector and therefore the algorithm try's to fit the most suitable point cloud to the RoadDNA map layer. After the calculation off the difference matrix ($diff_{i,q}$) the difference values are used to calculate the weights $w_k^{(i)}$.

**Algorithm 1** Pseudocode

```
[a,b]=size(M)   % M are all map values in region of interest
diff =ones(a,b)·$NaN$   % difference matrix
for each $y_{i,q,n}$   % For all measurement do
    if isempty($m_{i,q}$)   % if there is no values in maps x-, z-voxel then
        continue
    else if $abs(y_{i,q,n} - m_{i,q}) < diff_{i,q}$   //   $diff_{i,q} == NaN$ then
        $diff_{i,q}$=abs($y_{i,q,n} - m_{i,q}$)   % find the closest measurement $y_{i,q,n}$
end
$ve = diff_{i,q}(:)$ % matrix to vector
$ve(isnan(ve)) = []$ % remove NaN
$ve = median(ve)$ % call median estimator function
$N = length(ve)$
$w = exp(sum(-ve.\hat{}2)/(2N))$ % calculate weight $w_k^{(i)}$
% the (.) means that the operation is made on each value in the vector $ve$
% After all the weights are calculated normalization and resampling is performed
```

To calculating $m_j - y_j$ in (3.8) the difference matrix ($diff_{i,q}$) in algorithm 1 is made into one long vector and then all the NaN values are removed. Because of the problem of calculating the covariance $r_j$ in the update step a simpler solution is adapted that is based on the median-estimator section 2.5.2. Where the median off all the difference values are calculated and the weights of the independent error values are put to zero if it is above the median and 1 if below the median, because the sheer quantity of measurements this method seems to work fine. The idea with the median estimator is to lower the effect of occlusions and outliers from the LIDAR scans when calculating the weights. Then the weights $w_k^{(i)}$ is calculated as seen in algorithm 1 which is based on (3.8).

After the median estimator that removes the outliers and occlusions that exist in the LIDAR point cloud the weight's need to be normalized according to (3.10). Where $w_k^{(i)}$ is the weights with index i at time k and N is all the total number of weights. After each update step resampling is performed according to section 3.4.4.

$$w_k^{(i)} = \frac{w_k^{(i)}}{\sum_i^N w_k^{(i)}} \tag{3.10}$$

### 3.4.4 Resampling

The states in the particle filter have a tendency to diverge leading to a lower amount of efficient particles, therefore a resampling algorithm is introduced as described in section 2.3.7. No GPS resampling is introduced but could be added, this is because it is easier to evaluate the particle filter and especially the update function if no resampling with GPS is made. The resampling algorithm is based on [14] where the following steps are made:

- See the weights ($w_k^{(i)}$) as the probability of obtaining a particle i in the set i={1,2...N} where N is the total number of particles
- Draw N samples from the discrete distribution (weights) and replace the old particles with the new drawn particle set.
- Finally set all the weight to the same constant value $w_k = \frac{1}{N}$

### 3.4.5 Position Estimation

There exist multiple ways to calculate the position estimation. This implementation uses the weighted mean (MMSE) equation that is described in section 2.3.6 (2.9). This way of estimating the position gives a more robust result with less outliers when evaluated.

## 3.5 Utility functions

The particle filter calls on different utility functions depending on which sensors that is used. The main functions of these utility function is to connect the different sensors coordinate system. Linking global and local coordinate system and converting the global geographical coordinates lateral and longitudinal to global Cartesian coordinates.

### 3.5.1 Projection of points onto reference line

Since the digital map is represented by a reference line it is sometimes necessary to detect where a point $p_x$ is located in relation to the line. The reference line is represented by smaller line segments which are in turn represented by distinct reference coordinate points $p_i$. See figure 3.8 for a visual example, with $p_x$ as a red dot and the reference points $p_i$ in blue.

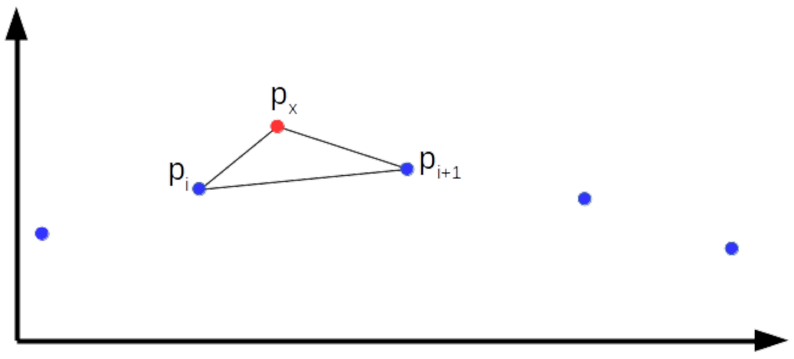


**Figure 3.8:** An example of a reference line represented by blue dots and a point of interest $p_x$ as a red dot. The closest reference points $[p_i, p_{i+1}]$ are marked and form a triangle with $p_x$.

By having a point $p_x$ and taking the two nearest reference points $p_i$ and $p_{i+1}$ a triangle is formed. See figure 3.9 for a more detailed diagram of this. The reason for using the indexes $(i)$ and $(i + 1)$ is since the reference line can be considered directional and it can be useful to know which side of the line a point is on.

$$\left.\begin{aligned}\bar{a} &= p_x - p_i \\ \bar{b} &= p_{i+1} - p_i\end{aligned}\right\} \Rightarrow \bar{c} = \frac{\bar{a}\cdot\bar{b}}{|\bar{b}|^2}\,\bar{b} \Rightarrow \bar{d} = \bar{a} - \bar{c} \tag{3.11}$$

The vector formed between $p_x$ and $p_i$ is called $\bar{a}$ and the one between $p_i$ and $p_{i+1}$ is called $\bar{b}$. The projection vector of $\bar{a}$ onto $\bar{b}$ is called $\bar{c}$ and the rejection vector from $\bar{b}$ to $\bar{a}$ is called $\bar{d}$. The magnitude of $\bar{d}$ gives the orthogonal distance $L_\perp$ between the point and the line. The magnitude of $\bar{c}$ gives the distance along the line from $p_i$ to the projection of $p_x$. By adding the distance $L_i$ from the start of the reference line $p_0$ until $p_i$ the total distance $L_x$ along the reference line to the projection of $p_x$ onto the reference line is calculated.

$$L_i = \sum_{j=0}^{i-1} |p_j - p_{j+1}| \tag{3.12}$$

$L_i$ is defined in (3.12) as the cumulative sum of the distances from the first reference point until the $i$-th one.

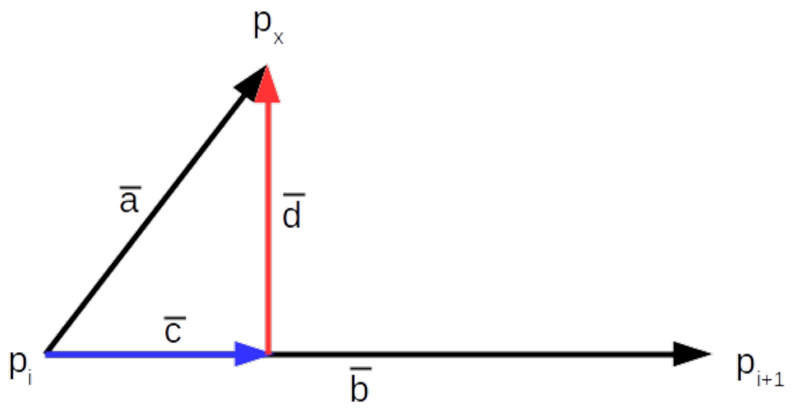


**Figure 3.9:** A more detailed diagram of figure 3.8. The vectors of interest formed in the triangle are marked and named. The vector projection (of $\bar{a}$ onto $\bar{b}$) is blue and the vector rejection (from $\bar{b}$ to $\bar{a}$) is red.

Since $L_x$ is the position of a point $p_x$ along a reference line and the reference line represents a road; $L_x$ can be considered as the longitudinal position along the road. The orthogonal distance $L_\perp$ between the line and point can conversely be considered as the lateral position on to the road. However $L_\perp$ is a vector magnitude so it will always be non-negative. This means that it can not indicate on which side (left or right) of the line the point is located. As mentioned the reference line is considered to be directional with increasing indexes so this notion of left and right of the line makes sense.

$$\bar{c} \times \bar{d} = \begin{vmatrix} \bar{x} & \bar{y} & \bar{z} \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{vmatrix} = \begin{vmatrix} c_y & c_z \\ d_y & d_z \end{vmatrix} \bar{x} - \begin{vmatrix} c_x & c_z \\ d_x & d_z \end{vmatrix} \bar{y} + \begin{vmatrix} c_x & c_y \\ d_x & d_y \end{vmatrix} \bar{z} \tag{3.13}$$

To detect which side the point $p_x$ is on one can use the assumption that the point is on a flat 2D plane which consists of an $x$- and $y$-axis. This is a reasonable assumption since we are considering a car driving on a road. The $\bar{c}$ and $\bar{d}$ vectors will then be laying flat on the plane. The $z$-component of the cross product as defined in (3.13) will indicate if $\bar{c}$ and $\bar{d}$ form a right-handed or a left-handed system by being positive or negative. If the point is on the left side it is a right-handed system and the result is positive and opposite if the point is on the right side.

This leads to (3.14) which takes the sign of the $z$-component. This can then be multiplied with the actual orthogonal distance to the point $L_\perp$ to give a lateral position, as in (3.15).

$$\delta = sgn\left([\bar{c} \times \bar{d}]_{\bar{z}-comp}\right) = sgn\left(\begin{vmatrix} c_x & c_y \\ d_x & d_y \end{vmatrix}\right) = sgn\left(c_x d_y - c_y d_x\right) \tag{3.14}$$

Finally the lateral and longitudinal position of the point $p_x$ can with relation to a reference line be expressed as in (3.15) by gathering the previous equations. The point is in this new transformed coordinate system called $P_x$.

$$P_x = \begin{bmatrix} \text{lateral pos} \\ \text{longitudinal pos} \end{bmatrix} = \begin{bmatrix} \delta \cdot L_\perp \\ L_x \end{bmatrix} = \begin{bmatrix} \delta \cdot |\bar{d}| \\ L_i + |\bar{c}| \end{bmatrix} \tag{3.15}$$

The projection utility function is applied on every particle in the particle filter to find the pixel the on the interpolated reference line which corresponds to a specific pixel in the RoadDNA image.

### 3.5.2 Converting global geographical coordinates to Cartesian coordinates

There exist multiple ways to convert the geographical coordinate system to a Cartesian coordinate frame as explained in section. The Cartesian Coordinate system used in this thesis is built with the Gauss Conformal Projection that uses Gauss-Krüger's equations to project a Cartesian coordinate system that can convert longitude and latitude position in degrees to x- and y- in a Cartesian coordinate system.

All the equations and constants are shown in appendix A, table A and (A.1). The origin for the global Cartesian coordinate system used in this project is located in Slottskogen, Gothenburg.

# 4
# Evaluation & Results

This chapter presents the data set and tests used to evaluate the implemented localisation algorithm. The results from the evaluation are also presented.

## 4.1 Data set

The data set used for the evaluation is taken from a test vehicle with sensors as described in previous chapters. The test vehicle has been driven through the city of Gothenburg while collecting measurement data. It has followed a specified track which is called the "DriveMe track". The DriveMe track is designed to have a certain set of various traffic situations. But also to constrain test drives of autonomous and assisted driving vehicles to a specific track, for reasons of safety.

The raw data set with measurements from all sensors has not been used. But rather a collection of the most relevant sensors measurement data was used. Some pre-processing has been done such as untwisting the LIDAR point data. More details on exactly how this data set is formatted can be found in chapter 3.

## 4.2 Evaluation procedure

The evaluation was performed by running the algorithm in Matlab on a laptop computer. The algorithm was feed measurement data from the pre-recorded data set in order to simulate a test run. Testing the algorithm on an actual test vehicle during a test run was not possible due to time and was also not included in the scope of the project.

Since the project was intended to focus on the signal processing aspects of localization, the switching between digital maps was performed offline. This meant that the algorithm always had the "correct" map provided for it.

As the algorithm ran through the data and provided position estimates, all the results were recorded and saved. The primary results of interest were the lateral, longitudinal position of the vehicle along the road and the heading. These positions show very well how the algorithm behaves since they relate well to the immediate surroundings of the vehicle. The global positions are not as informative but were also recorded since they are useful for some applications.

The results were gathered and statistical analyses were performed. The position results were primarily compared against the ground truth-data, which gives the best possible and most truthful positions of the vehicle. This gives the best possible indication of how accurate the results from the implemented algorithm are to reality. The ground truth data and the Velodyne sensor are not completely synced therefore the closest ground truth-data in time is used when evaluating the filter. This time sync error is usually of size $\sim$ 0.006 seconds, this is a very low error ($\sim$ 10cm longitudinal) that will not effect the overall longitudinal error much. The longitudinal error this provides is less than one pixel and therefore should not affect performance noticeable, since it is still within the same pixel in the image.

The evaluation then becomes the error from our estimated position and the ground truth position. The global-, longitudinal- and lateral-error is then calculated. The global error is calculated as seen in (4.1).

$$\begin{aligned} X_{error|k} &= X_{gt|k} - X_{pf|k} \\ Y_{error|k} &= Y_{gt|k} - Y_{pf|k} \end{aligned} \tag{4.1}$$

Where $X_{error|k}$ and $Y_{error|k}$ are the global error in x- and y-coordinates at time k. $X_{gt|k}$ and $Y_{gt|k}$ are the global ground truth positions at time k from the RT3000. While $X_{pf|k}$ and $Y_{pf|k}$ are the global estimated positions (weighted mean) from the particle filter at time k.

The global errors can then be used to calculate the local longitudinal- and lateral-error with the help of the ground truth heading acquired from the RT3000. This is done by rotating the coordinate system so that the x-axis is in the direction of the heading that the car has at that moment according to (4.2).

$$\begin{aligned} R_{heading} &= \begin{bmatrix} \cos h_{gt|k} & \sin h_{gt|k} \\ -\sin h_{gt|k} & \cos h_{gt|k} \end{bmatrix} \\ \begin{bmatrix} x_{error|k} \\ y_{error|k} \end{bmatrix} &= R_{heading} \begin{bmatrix} X_{error|k} & Y_{error|k} \end{bmatrix}^T \end{aligned} \tag{4.2}$$

Rotation matrix $R_{heading}$ converts the global error to local error with the help of $h_{gt|k}$ which is the ground truth heading at time k. The rotation matrix is then multiplied with global error which will give a longitudinal- and lateral-error represented as $x_{error|k}$ and $y_{error|k}$.

The filter always started with ground truth positions as initialisation for the state vector. This is because what is interesting is to evaluate is how the filter behaves when it has reached its steady state. Otherwise the result will heavily depend on how the initialisation was performed.

The filter is also evaluated in computational time and heading error. The heading error is calculated by comparing the ground truth heading with the estimated state heading as seen in (4.3). Where $h_{pf|k}$ denotes the heading that the particle has a t time k.

$$h_{error} = h_{gt|k} - h_{pf|k} \tag{4.3}$$

The final evaluation is the computational time. This evaluation is made by clocking the time it takes for the filter to run through a complete log file. Then the total time is divided by the number of update steps made and in the end the computational time acquired is the time it takes for the filter to do a prediction and update step.

## 4.3 Results

The first section 4.3.1 shows the global trajectory used when evaluating the filter (one long and one short trajectory). Section 4.3.2 will then present the results of the algorithm given in the road based coordinate system, in other worlds lateral and longitudinal displacement when comparing filter output and ground truth data. Then section 4.3.3 will provide the performance of the filter in heading and the final section 4.3.4 will show the computational speed for the filter.

### 4.3.1 Global position

The following trajectories were chosen when performing evaluation seen in figure 4.1 and 4.2. The two figures also shows the localisation results in the global Cartesian coordinate system. In this perspective the estimated and the "true" positions are very close and it is almost impossible to distinguish them. This shows that while there might be some local errors, the global position is in overall tracked fairly well.

Figure 4.1 shows the global trajectory (ground truth) and the estimated position for one log file which is a shorter distance of approximately 1330m. The lateral and longitudinal error from this localisation are shown in figure 4.3.

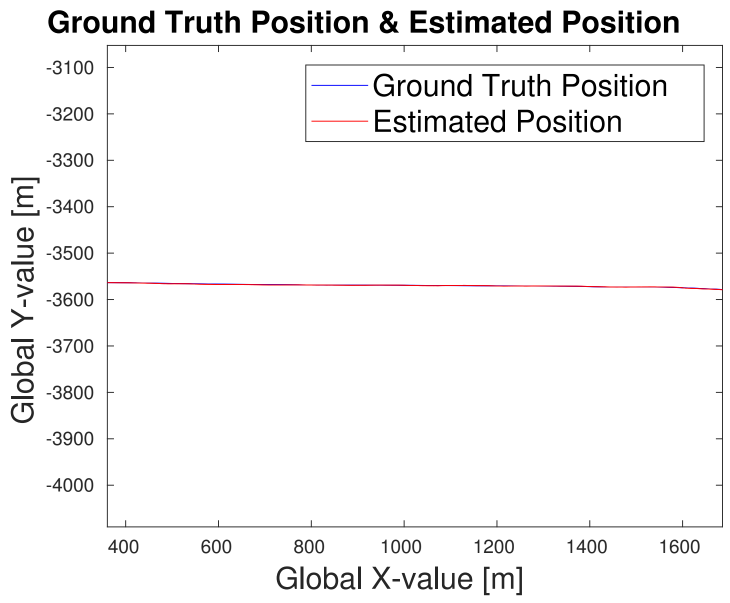


**Figure 4.1:** Time plot of position from both the algorithm and ground truth (in global Cartesian coordinate system). Corresponds to the same data as in figure 4.3.

The longer global trajectory with multiple data segments (multiple log files) and estimated position is shown in figure 4.2. The trajectory where localisation was performed is approximately equal to 4690m. Here is it visible why this is a more comprehensive test compared to the shorter single-segment one, since it for example has more turns and is significantly longer.

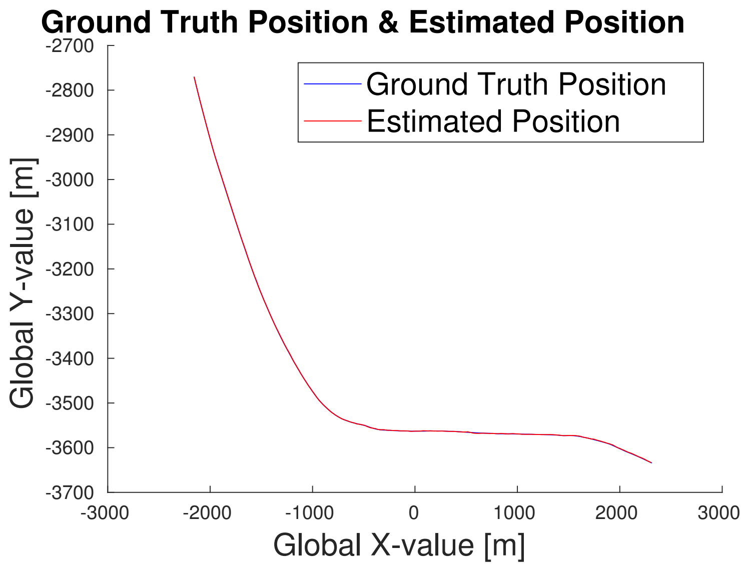


**Figure 4.2:** Time plot of position from both the algorithm and ground truth (in global Cartesian coordinate system). Corresponds to the same data as in figure 4.6.

### 4.3.2 Lateral & longitudinal position

The longitudinal and lateral errors are calculated according to (4.2). The result for the shorter trajectory when using 200 particles are shown in figure 4.3. The time-to-time differences are fairly small, no large and sudden "jumps" happen in the position estimates.

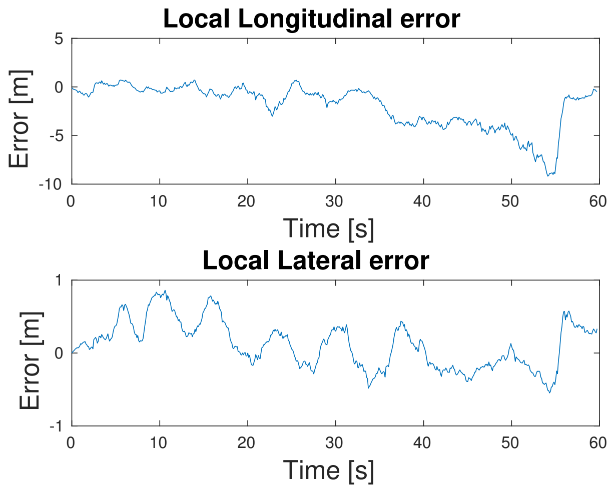
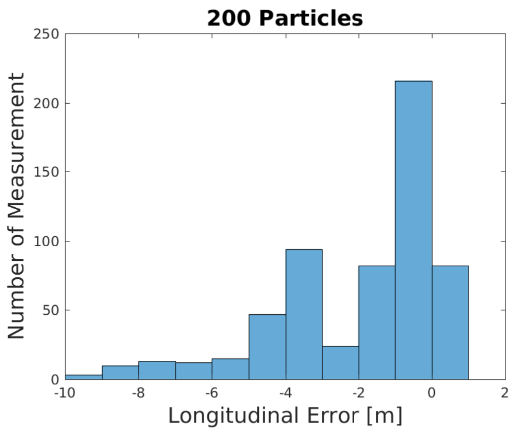
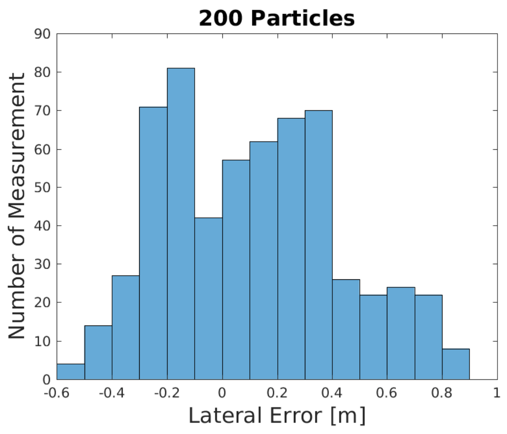


**Figure 4.3:** Time plot of error in lateral and longitudinal position between the algorithm and ground truth. For a single 60s data segment.

Some amount of "bias" is fairly noticeable in both of the plots. This is even more obvious when looking at figures 4.4a and 4.4b, which are histograms of the same data as figure 4.3. The lateral error has a mean of about $\sim -0.2$m with a standard deviation of $\sim 0.37$m and the longitudinal error has a mean of about $\sim -1.9$m and a standard deviation of $\sim 2.2$m. Both histograms are roughly in the shape of Gaussian distributions, with some amount of skew in the lateral distribution.



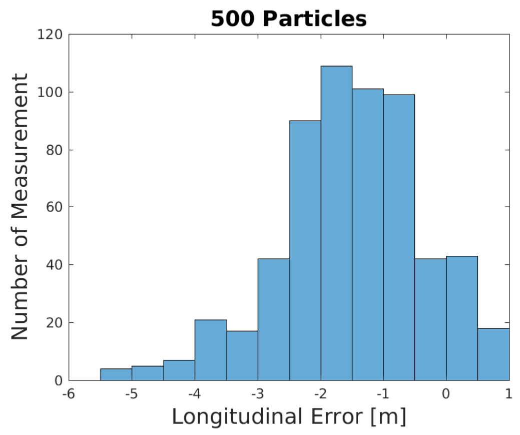
**(a)** Longitudinal error histogram.



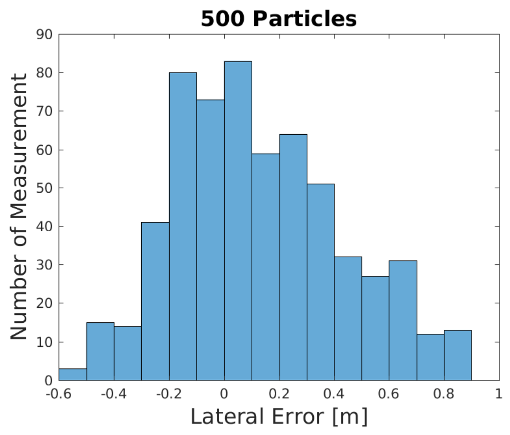
**(b)** Lateral error histogram.

**Figure 4.4:** Histograms of error in lateral and longitudinal position between the algorithm and ground truth. Corresponds to the same data as in figure 4.3.

When performing the same localisation on the same trajectory but with more particles (500) the filter performed slightly better in lateral displacement figure 4.5b (bias:0.12, standard deviation:0.3) and significant better in longitudinal displacement figure 4.5a (bias:-1.5 standard deviation 1.1). But the time to do these calculation increased significantly.



**(a)** Longitudinal error histogram.



**(b)** Lateral error histogram.

**Figure 4.5:** Histograms of error in lateral and longitudinal position between the algorithm and ground truth. Has the same trajectory as in 4.1.

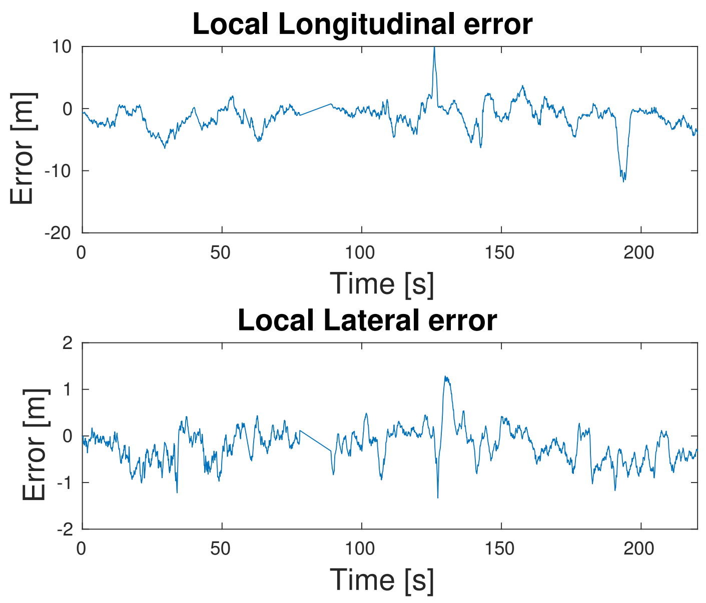
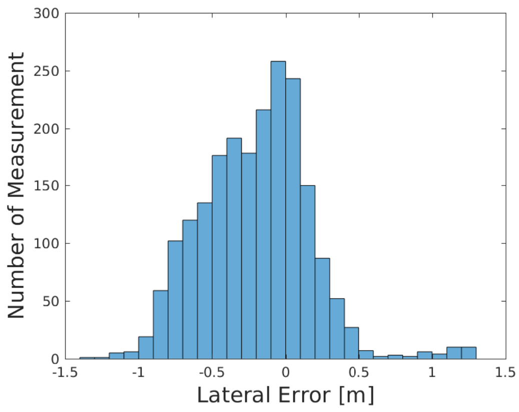


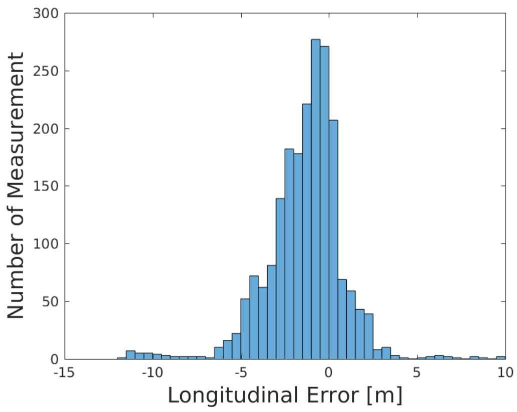
**Figure 4.6:** Time plot of error in lateral and longitudinal position between the algorithm and ground truth. For multiple consecutive data segments of ∼ 220s in total.

In figure 4.6 are the results from running multiple consecutive data segments through the algorithm. These are about ∼ 220s of time in total and covers a distance of 4.7km and should provide a more "realistic" view of the algorithm since trends can more easily be identified. And judging by the figure there are no apparent trends in the results, but there exist some outliers.

The histogram for the results in figure 4.6 can be seen in figure 4.7a and 4.7b. The mean for lateral errors is at about $\sim -0.2$m with a standard deviation of $\sim 0.37$m. For longitudinal the mean error is at about $\sim -1.3$m with a standard deviation of $\sim 2.3$m. They are not quite shaped like pure Gaussian distributions.



**(a)** Lateral error histogram.



**(b)** Longitudinal error histogram.

**Figure 4.7:** Histograms of error in lateral and longitudinal position between the algorithm and ground truth. Corresponds to the same data as in figure 4.6.

These histograms are somewhat more noisy compared to the previous ones, especially they have more bigger outliers. These big outliers are close to when the algorithm switch between the individual data segments see figure 4.6 at around time 120s. The switch is made with dead reckoning and that can be the reason for the big outliers. That should not happen but it is a possible error source. This is also a much longer run with more turns, which should reduce the accuracy of the algorithm slightly.

### 4.3.3 Heading

The main objective of the implemented algorithm is to estimate the position of a vehicle, whether that is given in a global Cartesian or road based coordinate system. But the algorithm also estimates the heading of the vehicle. Therefore the heading estimates can be interesting to examine.

Figure 4.8 shows the ground truth heading and the estimated heading from the algorithm when using 200 particles. This is taken from the same run as in figure 4.3. It is noticeable that the heading does not change much during this run which is due to the vehicle not rounding any sharp corners. The estimation of the heading is markedly less smooth than the "true" heading, but it does in overall follow it fairly well. This becomes more obvious when running the filter on the longer trajectory see figure 4.11.

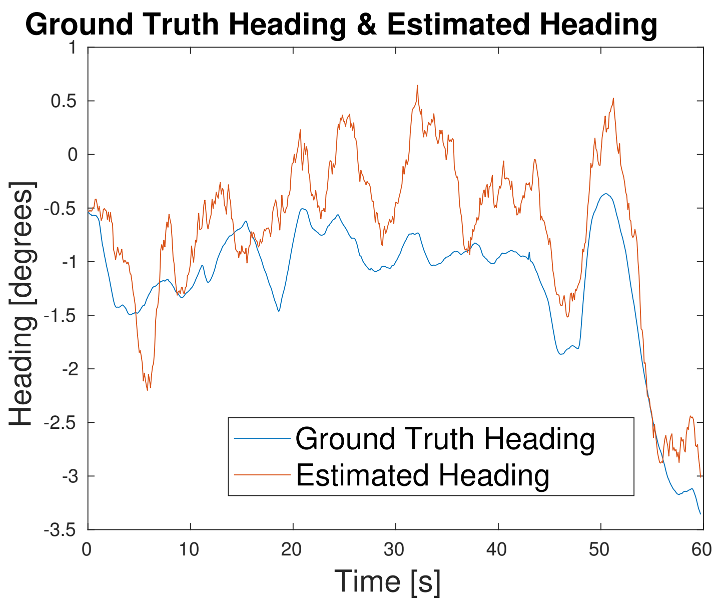


**Figure 4.8:** Time plot of estimated heading and ground truth heading. For a single $\sim$ 60s data segment and 200 particles.

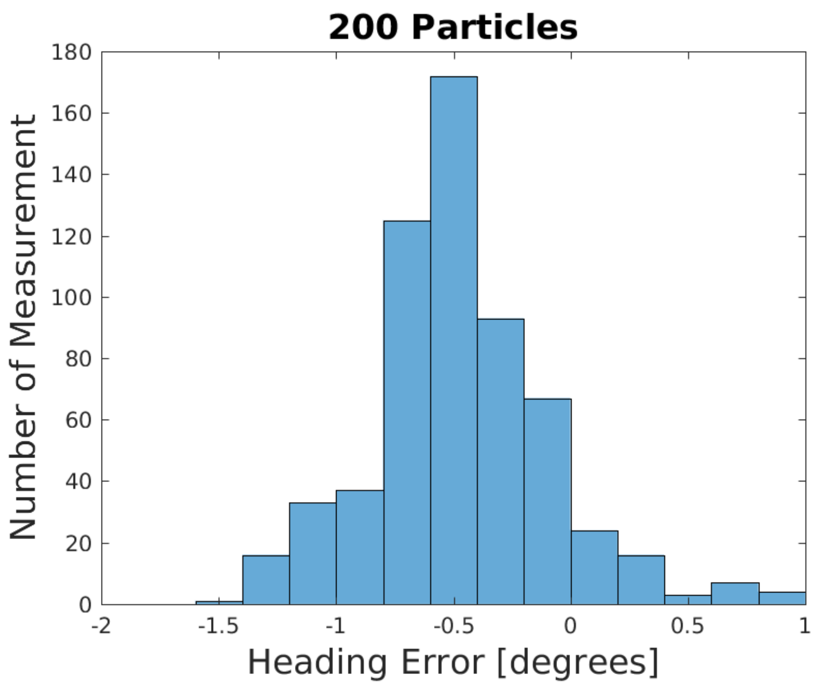


**Figure 4.9:** Histogram of error in heading with 200 particles.

The heading error is also calculated and represented in figure 4.9 as a histogram. The mean error is around $\sim -0.47°$ and with a standard deviation off $\sim 0.38°$. The histogram makes it easier to distinguish the distribution which is very close to a Gaussian distribution.

When using the same trajectory but with 500 particles the following results are acquired see figure 4.10. This figure correspond to the same run as 4.5a and 4.5b. The mean error for the heading is ∼-0.48 and the standard deviation is $\sim 0.39°$. There seems to be no significant change between 200 particles and 500 particles in this case but that can be due to the car almost always having the same heading.

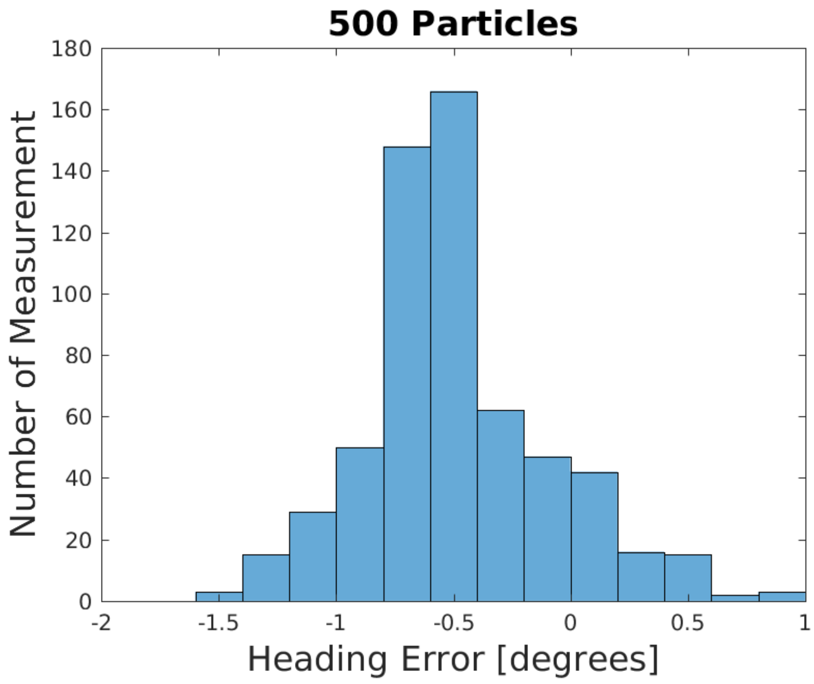


**Figure 4.10:** Time plot of estimated heading and ground truth heading. For a single $\sim$ 60s data segment and 500 particles.

The heading error from the longer test run is available for viewing as a histogram in figure 4.12. These results show largely the same behaviour as for the shorter test run, except with slightly bigger standard deviation and bias. The mean error is around $\sim -0.3389°$ and the standard deviation is $\sim 0.66°$ That could definitely come from the fact that the algorithm switches between multiple log files with dead reckoning which can provide significant outliers.

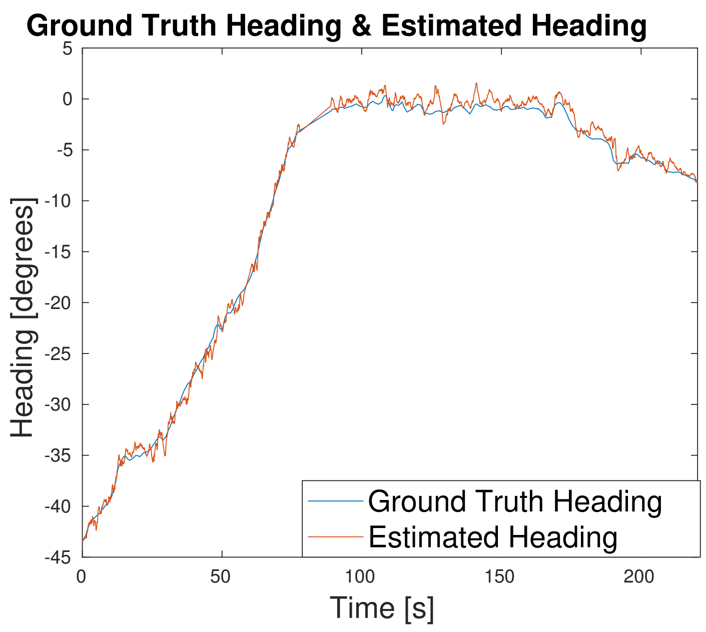


**Figure 4.11:** Time plot of estimated heading and ground truth heading. For multiple log files, ∼ 220s and 200 particles.

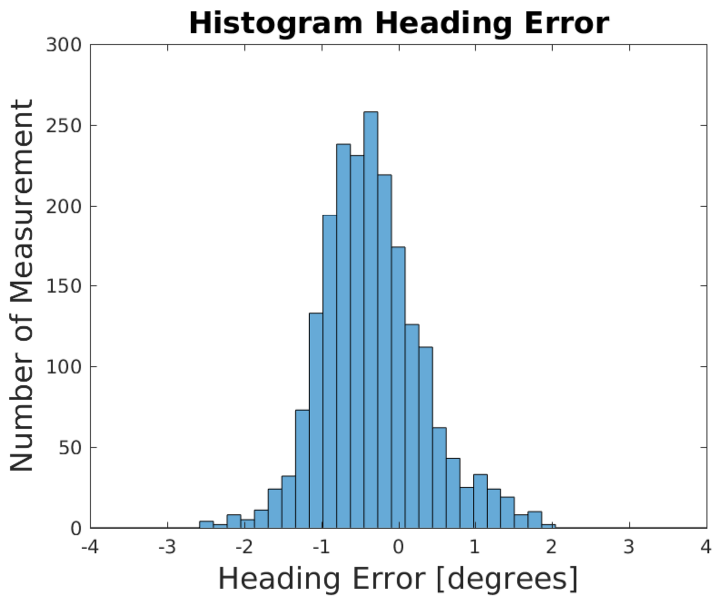


**Figure 4.12:** Time plot of estimated heading and ground truth heading. For multiple consecutive data segments of ∼ 220s.

The evaluation of heading estimation is not exhaustive, but it can still be said that the estimation is fairly good. An error which stays within a span of less than 2° is likely not going to negatively affect any other processes significantly.

### 4.3.4 Computational speed

The last evaluation metric is the computational speed, how fast can the filter do prediction followed by a update step for the different filters.

When dealing with 200 particles the computational speed for one continues prediction and update is approximately $\sim$ 4.5s which gives a frequency of around $\sim$ 0.2Hz. This leads to one single log file taking around 45 minutes to run through.

When the same localisation is done with 500 particles the computational speed is approximately $\sim$ 12s which generates a frequency of around $\sim$ 0.08Hz. This means that one single log file takes around two hours to run through. This makes evaluation of this amount of particles difficult.

The modules that contribute most to the slow computational speed is the update step and the utility functions that the update step needs see figure 3.5. The most taxing operations are all the different transformations of the point cloud that is performed see (3.9), every particle has their unique state vector that leads to a transformation of the point clouds. Another very taxing operation is the calculation of $y_{i,q,k}$ for every point in the particle filter see figure 3.7. If then a z-state is also incorporated more calculations are need due to the fact that z is then also implemented in to the transformation matrix. Which leads to an even slower computational speed and $y_{i,q,k}$ needs to be calculated for more points.

# 5 Discussion

What follows here are our discussions on a few topics concerning the project.

## 5.1 Position results

The results from the algorithm could be considered quite decent considering how new we are to the map. Also the algorithm is not really combined with other types of filters or maps, and very limited time has been available for actually tuning and optimising a final version of the algorithm. Some significant improvements might have

One very important segment to discuss when analysing the localisation results is the bias in longitudinal and lateral position. The bias that appears in the localisation can be due to multiple different factors. The alignment of the point cloud to the road might not be perfect, with the angles and the variable $d$'s value acquired for (3.1).

Many of the point clouds also have biases in the alignment angles. These biases can appear depending on the cars chassi in relation to the road, but there exist another bigger problem. This is the fact that a majority of the point clouds seems to have no or minimal calibration. This means that the LIDAR sensor was not calibrated before it started to collect data and therefore a multitude of biases in different angles and elevation can occur in the point cloud. The alignment tries to compensate for this but it might not be able to completely deal with it.

If this calibration is solved there exist a possibility to more easier relate RoadDNA and point clouds by implementing the changes in z-axis without having to deal with a z-state in the filter. The reason for this is due to the fact that the reference-line from RoadDNA has a x-,y- and z-position. If the point cloud is perfect the pitch angle in the point cloud should correspond with the change in elevation in the RoadDNA's reference line.

## 5.2 Behaviour of algorithm

One fairly important aspect of the algorithm is the speed of computation. Especially since this relates to whether it can be used for on-line localisation of vehicles.

The speed of the algorithm is at about $\sim 1/40^{\text{th}}$ of what is required for it to be considered "real time". This is however without any significant optimisation efforts and can likely be improved further.

In earlier iterations of the algorithm it tended to "get stuck" in certain environmental situations. These could for example be a group of trees which for untold reasons caused the algorithm to give very inaccurate results and give a constant position (while the vehicle actually kept moving). The latest implementation of the algorithm has less of this behaviour. At least not any that has been observed with tests. Instead the algorithm tends to jump forward until it reaches a point where enough structure is detected and there it waits until the car reaches that position.

Other reasons for the biases can be due to the problem that exist when localising against trees or other living and/or slightly moving objects. This in the end leads to the question of how long can you use a static map before it needs updating further, discussed in section 5.5.

One final contribution that might give a reason for our errors in longitudinal direction is the problem that occur when dealing with an environment that is homogeneous. Where the map or the point cloud looks the same in a certain region. This can be solved by using a bigger part of the point cloud or concatenating multiple point clouds. This is further elaborated in section 5.4,

The speed of the algorithm was at the time of evaluation not quite fast enough to use directly for on-line/real time application. However it was written completely in Matlab code and run in the Matlab environment. Optimisation was not implemented in terms of specialised computation hardware which could probably improve performance. The algorithm could also be implemented in a more efficient programming languages such as C++ to further improve performance. But generally speaking the current speed of the algorithm is not far from being realisable as a real time application. So we would consider it a good candidate for that, assuming that some steps are taken to improve performance slightly. With the rise in autonomous driving it might even occur that special purpose GPUs or even ASICs can improve the computational speeds significantly.

There exist an increase in performance when dealing with more particles in the particle filter. The problem is then the speed of the filter. The slow speed of the filter leads to it becoming difficult to evaluate the filter. Then the filter either needs to be optimised or another filter approach is required with the same or similar update step.

## 5.3 RoadDNA

As mentioned the algorithm mainly focuses on using the RoadDNA map layers, partially in order to evaluate how good results it can yield. However for a "real" application it might be prudent to combine it with other maps and filter types. This would mean increased redundancy, where there is less chance of the vehicle loosing its position. Accuracy might also be improved as filter results are fused.

The RoadDNA map layer is marketed as an efficient means of localizing a vehicle, both in terms of computation speed and also storage space. That could suggest that an algorithm using those maps can be used as a fairly light-weight localisation option. When more accuracy is needed more sensors, and possibly maps, could be connected/enabled and more processing power dedicated for it.

An interesting situation which might result in difficulties if only relying on RoadDNA is when there are no real structures on the sides of the road. This might for example happen in flat country terrain such as large grassy fields or deserts. Without trees, buildings or lamp posts there is no available "information" for the maps, meaning the maps would simply look like white images. Dead-reckoning and occasional GPS-measurements could be used, but will not provide incredible accuracy. It would be advisable that other methods are used to complement RoadDNA.

## 5.4 Future Work

The particle filter might not be the most efficient type of filter for this task (with this specific type of maps). The idea to use a Point Mass Filter was considered but unfortunately there was not enough time to implement and test it. A PMF works very much like a particle filter but the "particles" are distributed in an organised way throughout the discrete grid of the map. It could lead to less "particles" being necessary and the world (according to the map) is already divided into grids. This can help with performance and the reliability of the algorithm. A point mass filter would be a good option to look into more for further work related to this project.
Another idea for future work is to use classifications so that the object in the LIDAR cloud can be classified. Then depending on what kind of object it is they have different usefulness and "reliability". Trees and other vegetation might not be the best objects to perform localisation with since they can look very different depending on wind, time of year and so on. Therefore by classifying such object they can either be filtered out entirely or have less reliance given to them in the localisation algorithm since they are less reliable.

The median-estimator used in this project exist to deal with occlusion from the LIDAR sensor and vegetation in the map. This method can probably be improved upon by completely filtering out everything that can be classified as occlusions. Much work can likely be done in with either this method or others that can help in removing (more or less) useless objects from the LIDAR data such as other vehicles.

If all occlusions can be filtered out from the point cloud the update step should be able to be changes so that it does not only take in to account the voxels that has map measurement in them but also the voxels that do not have map measurement in them. Then you will also compare the absence of information in the update step which should give a improvement in result. Also the fact that many of our point cloud have minimal calibration and sometimes strange biases is fixed this should help with performance.

There are also many different parameters that can be changed in our implementation. If the LIDAR sensor has a high update frequency the possibility exist to only look directly perpendicular to the cars position and only use this part for localisation. Then concatenate multiple of these smaller point clouds and use this to improve localisation, then you are certain that all the information in the point cloud is directly perpendicular to the Road. The problem that needs to be solved for this approach is the concatenation of multiple of these smaller point clouds. It can probably be made with the help of the odometry but that approach feels like it might not be the best from a performance aspect.

Another future work is to change the update function so that it can handle the computational problems better while not affecting how much we trust the map. Instead of using $\lambda^{\frac{1}{N}}(y, m(x))$ in (3.8) where we make the assumption that there is only one measurement and thereby lowering the trust in the map there might exist a better option that can handle the computational problems and not affect the particles and the evaluation as much. One such future work is that instead of dividing by N you divide with a lower factor. This will change the interpretation so instead of saying that you have one measurement and map you have multiple. Which will make you trust the map more and still make it computationally possible, but this then becomes a tuning factor that needs to be taken in to account. Also some problems can occur depending on how low the factor is and how many measurement you have at that time and how the map looks like at that position and how far away you are from the reference line thereby making it time consuming to optimise, because of all the special cases that will occur.

## 5.5 General

From some reflection and experience, it is in the belief of the project group that any self driving car should always strive to make use of a combination of sensors and methods for detection, localisation and control tasks. Cameras, LIDARs and radars can all be covered in snow, mud, water or even damaged or vandalised. What is required is redundancy in most of the systems of the vehicle.

One very interesting aspect to this project is that with a high probability the on-board LIDAR sensor and the sensor that creates RoadDNA is probably not the same this can be a contributing factor to the larger error our algorithm sometimes get. There is also the important aspect of when RoadDNA was created and when our cars were out driving. If the dates is to far away the static environment might have changed and therefore the localisation will perform worse. So one very important aspects become how long time should pass until the static environment has changed so much that the map needs to be updated.

# 6
# Conclusion

A conclusion to draw from the results of the evaluation would be that the implemented algorithm does indeed seem to function as intended. Positions are estimated within a margin of error which is in the single meter range. They also do not appear to stray far from the true values as time passes. Although centimetre or decimetre level accuracy was hoped for it was not expected, we and the project group consider the result satisfactory.

We would thus consider the project a success even though there is much potential future work which might improve results. We would also recommend that the algorithm created in this project, or variations thereof, be investigated further as we believe it shows promise. Alterations to the algorithm as presented in the discussion chapter is a good option for further investigations, and a recommendation from us would defiantly be to investigate how this algorithm can be combined with other methods and maps to create an overall more robust and reliable localisation system.

# Bibliography

[1] World Health Organization. *Road traffic injuries*, January 2017. http://www.who.int/mediacentre/factsheets/fs358/en/.

[2] David Z. Morris. Driverless cars will be part of a $7 trillion market by 2050. http://fortune.com/2017/06/03/autonomous-vehicles-market/, 2017. Online; accessed 11 June 2018.

[3] Engadget. Roaddna new article. https://www.engadget.com/2015/09/14/tomtom-roaddna-self-driving/, acquired June 2018.

[4] Paul McManamon and SPIE eBooks (e-book collection). *Field Guide to Lidar*. SPIE, Bellingham, 2015.

[5] Velodyne. Hdl-64e data sheet. http://www.velodynelidar.com/lidar/products/manual/HDL-64E%20Manual.pdf, acquired May 2018.

[6] Lantmäteriet. Gps system. https://www.lantmateriet.se/sv/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/GPS-och-satellitpositionering/GPS-och-andra-GNSS/GPS/, acquired May 2018.

[7] Lantmäteriet. Other gnss systems. https://www.lantmateriet.se/sv/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/GPS-och-satellitpositionering/GPS-och-andra-GNSS/Ovriga-GNSS-och-satellitbaserade-stodsystem/, acquired May 2018.

[8] US Gov Agricultural Applications. Gps accuracy. https://www.gps.gov/systems/gps/performance/accuracy/, acquired May 2018.

[9] Oxford Technical Solutons. Rt3000 data sheet. https://www.oxts.com/app/uploads/2017/07/RT3000-brochure-170606.pdf, acquired May 2018.

[10] Richard B Langley. Rtk gps. *GPS World*, 9(9):70–76, 1998.

[11] OXTS. Rt3000 v2. http://176.58.104.161/app/uploads/2017/07/RT3000-brochure-170606.pdf, acquired January 2017. Product description.

[12] Mordechai Ben-Ari, Francesco Mondada, SpringerOpen (e-book collection), Directory of Open Access Books (DOAB) Free (e-book collection), SpringerLink (e-book collection), and SpringerLink (Online service). *Elements of Robotics*. Springer International Publishing, Cham, 2018.

[13] TomTom. Roaddna information sheet. https://automotive.tomtom.com/wordpress/wp-content/uploads/2017/01/RoadDNA-Product-Info-Sheet-1.pdf, acquired June 2018.

[14] Simo Särkkä. *Bayesian filtering and smoothing*. Cambridge university press, Cambridge, 2013.

[15] F. Gustafsson and A. J. Isaksson. Best choice of coordinate system for tracking coordinated turns. volume 3, pages 3145–3150 vol.3, 1996.

[16] Malin Lundgren, Erik Stenborg, Lennart Svensson, and Lars Hammarstrand. Vehicle self-localization using off-the-shelf sensors and a detailed map. 2014.

[17] Particle filter based map state estimation: A comparison. *Proceedings of 12th International Conference on Information Fusion 2009*, page 278, 2009.

[18] Robert Bassett and Julio Deride. Maximum a posteriori estimators as a limit of bayes estimators. *Mathematical Programming*, pages 1–16, 2018;2016;.

[19] Ordnance Survey. A guide to coordinate systems in great britain. `https://www.ordnancesurvey.co.uk/docs/support/guide-coordinate-systems-great-britain.pdf`, acquired June 2018.

[20] Sam Souliman. Oxts reference frames and iso8855 reference frames. `https://support.oxts.com/hc/en-us/articles/115002859149-OxTS-Reference-Frames-and-ISO8855-Reference-Frames#Angles`, 2018. Online; accessed 7 June 2018.

[21] Martin Fischler and Robert Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, 1981.

[22] MATLAB. pcfitplane. *Describes the package pcfitplane in MATLAB.* `https://se.mathworks.com/help/vision/ref/pcfitplane.html`.

[23] P. H. S. Torr and A. Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78(1):138–156, 2000.

[24] Tommi Tykkälä. *Real-time image-based RGB-D camera motion tracking and environment mapping.* Theses, Université Nice Sophia Antipolis, September 2013.

[25] Mathworks. Matrix representation of geometric transformation. `https://se.mathworks.com/help/images/matrix-representation-of-geometric-transformations.html`, 2018. Online; accessed 7 June 2018.

[26] European Petroleum Survey Group. Coordinate conversions and transformations including formulas. `http://www.ihsenergy.com/epsg/guid7.pdf`, 2004. Online; accessed 7 June 2018.

[27] Lantmäteriet. Gauss conformal projection (transverse mercator). `http://www.lantmateriet.se/globalassets/kartor-och-geografisk-information/gps-och-matning/geodesi/formelsamling/gauss_conformal_projection.pdf`, 2008. Online; accessed 7 June 2018.

# A

# Appendix 1 - Equations

The Gauss-Krũger equations have two constants that have different values depending on where in the world the coordinate system is used. Therefore there exist a reference system known as GRS80 that gives these values [26].

Table A shows the Gauss-Krũger equations and constants that is solved when creating a Cartesian coordinate system from latitude and longitude. The constants a and f is taken from GRS80 and then used to solve the rest of the equations. FN and FE is the distance from the natural origin to the positions of interest. All the equations are further explained and based on [27, 26].

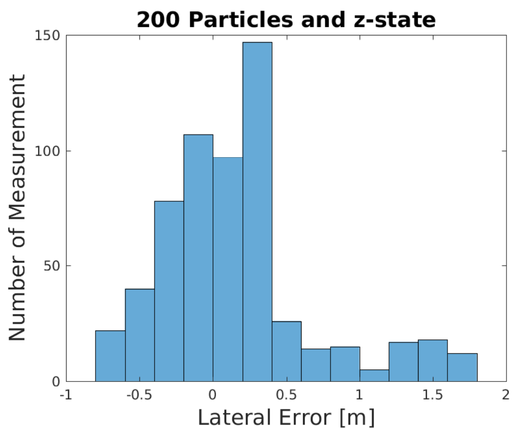
| Constants | Equations or value |
|---|---|
| $a$ | 6378137 |
| $f$ | $\frac{1}{298.257222101}$ |
| FN | 0 |
| FE | 150000 |
| $e^2$ | $f(2-f)$ |
| $n$ | $\frac{f}{2-f}$ |
| $\hat{a}$ | $\frac{a}{1+n}\left(1+\frac{1}{4}n^2+\frac{1}{64}n^2\right)$ |
| $\psi^*$ | $\psi - \sin\psi\cos\psi(A + B\sin^2\psi + C\sin^4\psi + D\sin^6\psi$ |
| $A$ | $e^2$ |
| $B$ | $\frac{1}{6}(5e^4 - e^6)$ |
| $C$ | $\frac{1}{120}(104e^6 - 45e^8+)$ |
| $D$ | $\frac{1}{1260}(1237e^8)$ |
| $\delta\lambda$ | $\lambda - \lambda_0$ |
| $\xi'$ | $\tan^{-1}(\frac{\tan\psi^*}{\cos\delta\lambda})$ |
| $\eta'$ | $\tanh^{-1}(\cos\psi^*\sin\delta\lambda)$ |
| $\beta_1$ | $\frac{1}{2}n - \frac{2}{3}n^2 + \frac{5}{16}n^3 + \frac{41}{180}n^4$ |
| $\beta_2$ | $\frac{13}{48}n^2 - \frac{3}{5}n^3 + \frac{557}{1440}n^4$ |
| $\beta_3$ | $\frac{61}{240}n^3 - \frac{103}{140}n^4$ |
| $\beta_4$ | $\frac{49561}{161280}n^4$ |

Finally when all the Gauss-Krũger equation in table A is solved the final part is left and that is to calculate x and y according to (A.1) This will finally give a Cartesian coordinate system [27].

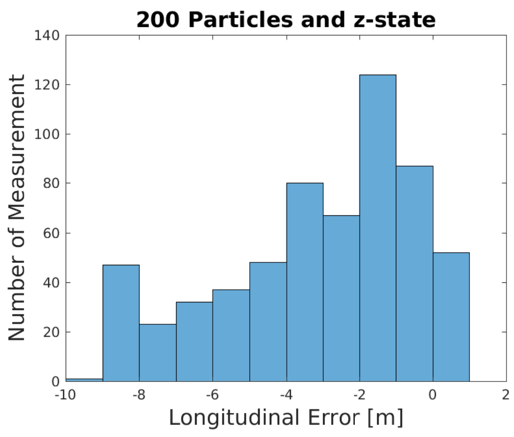
$$\begin{aligned}
x &= k_0\hat{a}(\xi' + \beta_1 \sin 2\xi' \cosh 2\eta' + \beta_2 \sin 4 * \xi' \cosh 4\eta' + \beta_3 \sin 6\xi' \cosh 6\eta' \\
&\quad + \beta_4 \sin 8\xi' \cosh 8\eta') + FN \\
y &= k_0\hat{a}(\eta' + \beta_1 \cos 2\xi' \sinh 2\eta' + \beta_2 \cos 4\xi' \sinh 4\eta' + \beta_3 \cos 6\xi' \sinh 6\eta' \\
&\quad + \beta_4 \cos 8\xi' \sinh 8\eta') + FE
\end{aligned} \tag{A.1}$$

# B
# Appendix 2 - Results

When implementing the z-state in to the state vector for the particle filter the results seen in figure B.1a, B.1a and B.2. What is observed generally when implementing z is higher standard deviation, bigger bias and longer computational time. In these figures above the longitudinal error has a bias of $\sim -3.1$m and standard deviation of $\sim 2.59$m. The lateral error has a bias of $\sim 0.16$m and a standard deviation of $\sim 0.52$m. The heading has a bias of $\sim -0.456°$ and a standard deviation of $\sim 0.46°$. In general all the errors are bigger with more bias when incorporating z as a state and this seams to be the general case if the same amount of particles are used. The computational speed is also lower when using the same amount of particles, around 0.1Hz $\sim$ 10s. More particles might fix the errors but then the calculation time increases rapidly.



(a) Lateral error histogram.



(b) Longitudinal error histogram.

**Figure B.1:** Histograms of error in lateral and longitudinal position between the algorithm and ground truth with z state as random walk(RW). same trajectory as in 4.1

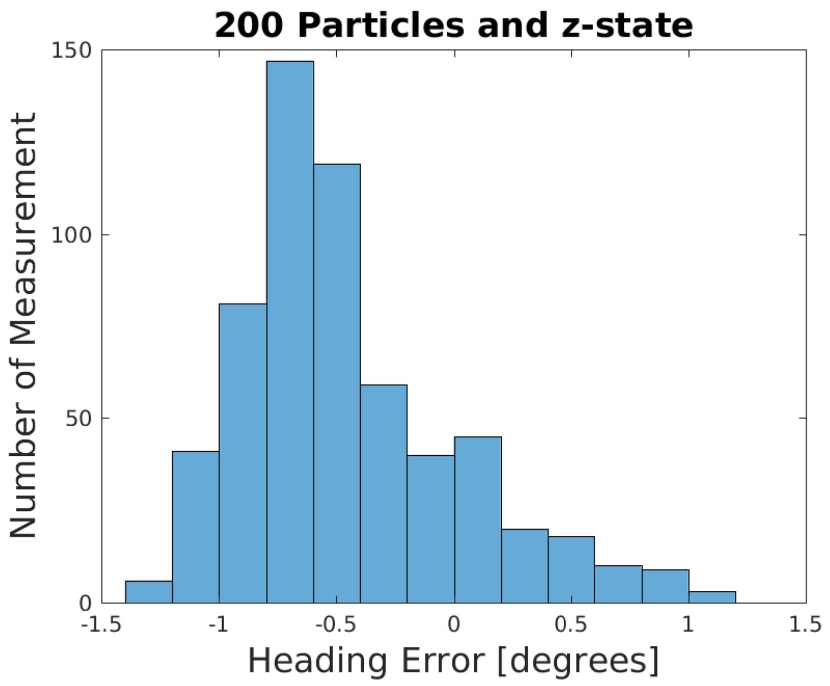


**Figure B.2:** Histogram of error in heading with 200 particles and z-state as random walk.