Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2021

Development of an ICP-based Global Localization System

Rebecka Nylén and Katherine Rajala

Master of Science Thesis in Electrical Engineering

Development of an ICP-based Global Localization System

Rebecka Nylén and Katherine Rajala

LiTH-ISY-EX--21/5434--SE

Supervisor: **Kristin Nielsen**
ISY, Linköpings universitet
Jonas Nygårds
Swedish Defence Research Agency, FOI

Examiner: **Gustaf Hendeby**
ISY, Linköpings universitet

Division of Automatic Control
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

The most common way to track the position of a vehicle is by using the *Global Navigation Satellite System* (GNSS). Unfortunately, there are many scenarios where GNSS is inaccessible or provides low precision, and it can therefore be vulnerable to only rely on GNSS. This master's thesis is done in collaboration with the *Swedish Defence Research Agency* (FOI), who is looking for a solution to this problem. Therefore, this master's thesis develops a system that globally localizes a vehicle in a map, without GNSS. The approach is to combine odometry and the scan registration algorithm *iterative closest point* (ICP), in an *extended Kalman filter* (EKF), to provide global position estimates. The ICP algorithm aligns two different sets of data points, referred to as point clouds. In this thesis, one set consists of *light detection and ranging* (LIDAR) data points collected from a sensor mounted on a vehicle, and the other consists of LIDAR data points collected from an aircraft which forms an elevation map of the area. In the ideal case, the algorithm finds the position on the elevation map where the vehicle collected the data points.

For the EKF to function, the uncertainty of ICP must be estimated. Different methods are investigated, which are; unscented transform based covariance, covariance with Hessian, and covariance with correspondences. The result shows that all the methods are too optimistic when estimating the uncertainty. The reason is that none of the methods take all sources of error into account, and it is therefore difficult to correctly capture the uncertainty of ICP. The unscented transform based covariance is the least optimistic, and covariance with correspondences is the most.

A second problem investigated in this thesis is how odometry and ICP with an elevation map as reference can be combined to provide a global position estimate. As mentioned, the chosen approach is to implement an EKF which weights the different data sources based on their covariance, to one single estimate. The developed global localization system is evaluated in a real time experiment, where the data is recorded using equipment from FOI. The goal of the experiment is to localize a vehicle while it is driving in different environments, including urban, field and forest environments. The result shows that the performance of the system is viable, and it manages to provide localization within a few meters from ground truth. However, since the ICP covariance estimates are not fully accurate, the performance of the EKF is decreased as it cannot weight the different estimates properly.

The ICP algorithm used in the system has a lot of flaws. The worst is that it easily converges to incorrect solutions, in other words that it estimates the wrong position of the vehicle. How this risk can be decreased is also investigated in this thesis. A method that decreases this risk drastically, and makes the viable performance of the system possible, is developed. The approach of the method is to exclude incorrect positions by removing a large amount of points from the point clouds, and keeping the most informative. By only utilizing the most informative data points in the point cloud, global positions with high accuracy are achieved.

Acknowledgments

We would like to thank our supervisors Jonas Nygårds at FOI, and Kristin Nielsen at Linköpings University for great support during this thesis work. We also want to thank FOI for providing the task for this interesting project, helping us with the experiment setup and a good work environment. Furthermore, we would like to thank our examiner Gustaf Hendeby for making this thesis possible.

Linköping 2021,
Rebecka Nylén and Katherine Rajala

Contents

1 Introduction

This chapter presents the background of this thesis and puts the problem in context, along with the problem formulation and an overview of the system. It also presents how the project is divided between the two authors and the outline for the rest of the report.

1.1 Background

The most commonly used localization tool today is the *Global Navigation Satellite System* (GNSS), but unfortunately it is not available or very precise at all locations. Relying fully on GNSS can therefore be vulnerable. A variety of other localization methods have been developed during the last decades. The most used method can be found in for example [19], where *simultaneous localization and mapping* (SLAM) is used to locate a vehicle in urban environments with centimeter precision. SLAM systems usually suffer from odometry errors, which is why in [11] WiFi is tested as a method to use in indoors environments. The *Swedish Defence Research Agency* (FOI) has a SLAM system that uses landmarks detected by a LIDAR sensor, together with odometry to estimate a map of the environment, and locates the vehicle in that map. The localization is only local and not global, meaning that it only provides a position relative to the starting position. It can estimate positions in urban and forest environments, but gives poor positioning in open field terrain due to lack of landmarks [14].

This master's thesis is about vehicle localization, and is a collaboration with FOI. The aim is to develop a global localization system that provides the position of a vehicle in global coordinates, regardless of landmarks and GNSS. This would address the shortcomings of FOI's current localization system, and be an alternative to GNSS that can be of use in areas where GNSS is inaccessible or in situations where the position of a vehicle is of great significance.

1.2 Problem Formulation

To find the position of a vehicle in a global map without GNSS, sensor data that provides information about the environment can be used and compared to the information in a map of the area. A registration algorithm called *iterative closest point* (ICP) aligns two data point clouds that contain similar information, and is therefore suitable to use. It does not need landmarks to align the point clouds, therefore it also addresses shortcomings of FOI's SLAM system. From a LIDAR sensor mounted on a vehicle, a point cloud in local coordinate frame of the sensor describing the ground around it can be obtained. This point cloud can be matched against a global point cloud map containing the height curves of the area using the ICP algorithm, which provides a position estimate of the vehicle in that map. However, ICP has a lot of flaws [7], and should therefore not be the only source that provides position estimates in a localization system. To increase robustness, odometry can be used in the system as well. The problem of how these different position estimates can be combined to provide accurate global localization of a vehicle is addressed in this thesis. When combining these estimates, it is beneficial to know the uncertainty of ICP to be able to decide how trustworthy the position estimates are. Therefore, different ways to determine the uncertainty of ICP is looked into as well.

As mentioned, the ICP algorithm has flaws. The main drawback is that it does not take into account if it has found the best matching position on the whole map, or just the best match in the vicinity of the initial position. In other words it does not differentiate between global and local minimums. How convergence to these incorrect solutions can be decreased is investigated in this thesis as well. A proposed approach is to select an informative subset of points to be used in the ICP algorithm, with the purpose of excluding incorrect local minimums.

This problem description sums up to the following questions that are going to be answered in this thesis

- What are some different possibilities to estimate the uncertainty of ICP, and how correct are the results?
- How can odometry and ICP with an elevation point cloud as reference be combined to provide a global position estimate?
- How can the risk of ICP converging to incorrect solutions be decreased?

1.3 System Overview

In this section, an overview of the developed global localization system is presented, along with the purpose of each part. The system is implemented in C++ together with *Robot Operating System* (ROS) [30].

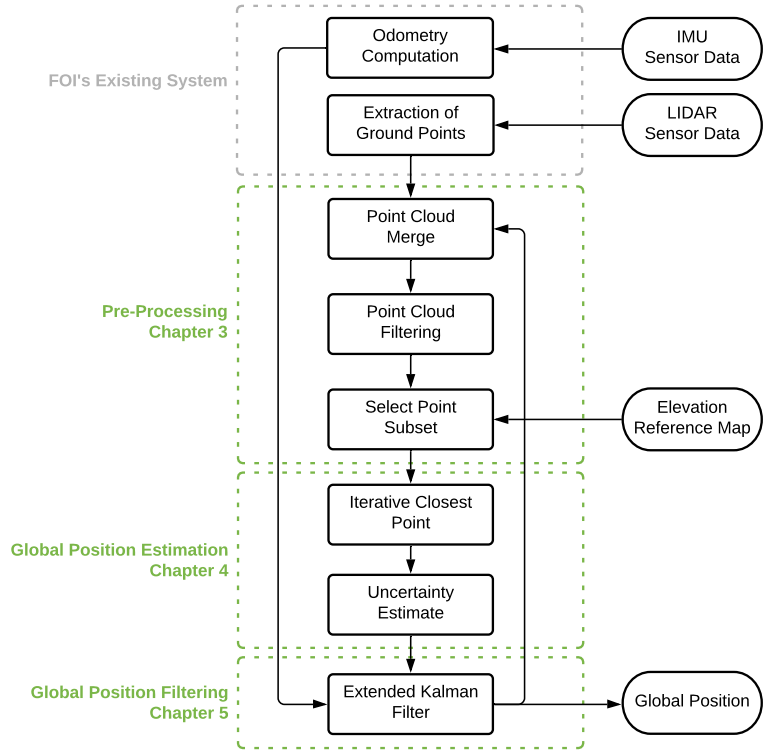


Figure 1.1: *An overview of the system. The green parts are created in this thesis, and the grey are existing parts at FOI that are used straight off.*

Figure 1.1 illustrates the structure of the system that is developed in this thesis. In the figure, it is shown that some parts are taken straight off from the existing system at FOI. The purpose of these parts is to convert the sensor data into data that is needed in the rest of the system. As mentioned, FOI's existing system performs SLAM [12], but the developed global localization system is not integrated into the SLAM part. It only uses the odometry computations (which provide position estimates relative to the starting position, and associated covariance estimates) to be able to evaluate it separately from SLAM. How the odometry in FOI's system is computed, as well as how SLAM functions, is described in a previous master's thesis [14].

Except for the odometry computations, a part that extracts the ground points from the LIDAR sensor data is used as well. These ground points represents the height curves around the vehicle without trees, houses, and other landmarks. This is needed since the elevation map used as reference only contains ground points, and the data sets used in ICP must be as similar as possible.

In addition to the parts that are taken from FOI's existing system, the rest of the developed system can be divided into three steps; *pre-processing*, *global position estimation*, and *global position filtering*. The purpose of the *pre-processing* step is to prepare the point clouds that are going to be used in the ICP algorithm. One single scan from the LIDAR sensor does not contain enough information to be able to match it against the reference map. Therefore, the point cloud merge part is needed to fill the cloud with a suitable amount of information. Previous work done on point cloud filtering, which is a source of inspiration for the filtering part of the *pre-processing* step, is found in [15]. The common factor is that both this system and the one presented in [15] uses large noisy data sets as input, and the methods they use to remove the noise are shown to be successful and very common. To decrease the risk of ICP converging to incorrect solutions, a subset of informative points are extracted as a final part of the *pre-processing* step.

The purpose of the *global position estimation* step is, as the name suggests, to provide a position estimate in the global coordinate frame. As mentioned, the ICP algorithm can, if used with an elevation point cloud map as reference, achieve global position estimates regardless of landmarks. Therefore, it is suitable to use in this system. It also addresses two shortcomings of FOI's existing SLAM system, which are that the SLAM system only provides local localization, and that it is dependent on landmarks. The uncertainty of ICP is determined in this step as well, and why it is needed is explained in the next paragraph.

Both odometry and ICP have flaws, and it is doubtful to individually trust the position estimates that they provide. Therefore, a *global position filtering* step with the purpose to combine these estimates is needed. An EKF is a common algorithm that achieves this. It weights the different estimates, i.e. decides how trustworthy they are, based on their covariance. This is why the uncertainty of the ICP is determined in the previous *global position estimation* step. Previous work regarding this is found in [20], where map-based localization is achieved by merging information from ICP with odometry in an EKF.

1.4 Delimitations

The following delimitations are set in the thesis.

- FOI's existing system computes a path based on odometry which is used in this developed global localization system. How these computations, including the covariance of these measurements, are made is out of the scope of this thesis, but is described in [14].

- The global localization system only estimates a 2D position of the vehicle, i.e. a position in a map seen from above, which are the most relevant coordinates.

1.5 Division of labor

The two authors, Rebecka and Katherine, have divided the work in the following way. Rebecka has created and written about the implemented ICP algorithm which uses functions from PCL (Section 4.1.1), along with methods that help avoid incorrect local minimums, as selecting subset clouds (Section 3.4) and using stochastic initial positions (second part of Section 4.1.4). She has also created and written about the EKF that estimates the filtered global position (Section 5.1). Katherine has created and written about the extended ICP algorithm that is based on unscented transforms (third part of Section 4.1.4), along with methods that estimates the uncertainty of the ICP result, as different covariance variants (Section 4.2) and fitness score (Section 4.1.2). She has also created the algorithm for merging point clouds (Section 3.2). The remaining work has been performed together.

1.6 Outline

After this introductory chapter, Chapter 2 contains theory about point clouds, the ICP algorithm, covariance estimates and EKF, which are needed to understand the components of the global localization system developed in this thesis. Chapter 3, 4 and 5 then presents and evaluates the three different parts of the developed system, which are *pre-processing*, *global position estimation* and *global position filtering*. After evaluating the system parts individually, two system variations are created and tested in an experiment where a vehicle is driving different routes and the goal is to provide global localization during that time. This experiment is presented in Chapter 6. At last, Chapter 7 contains the conclusions and future work of this thesis.

2

Theory

This chapter presents the background theory of this thesis. First, there is information about point clouds and how they can be processed. This theory is used in the *pre-processing* step. Then introducing the ICP algorithm, along with methods to estimate its uncertainties, to give the understanding needed for the *global position estimation* step. The position filtering method EKF, which is used in the *global position filtering* step, is also found in this chapter. Last, there is information about a software library used in the system.

2.1 Point Clouds

A point cloud is a set of data points containing coordinates, which constitutes an object or an environment. Point clouds can be created by hand, through 3D-modelling software programs, or given as output from a sensor as described in Section 2.1.1. Point clouds collected from a sensor in real life will always contain noise and a large amount of data, which can be handled with downsampling and outlier removal algorithms described in Section 2.1.2. Some point cloud matching methods, as for example ICP, requires information about the surface that the point cloud represents. This information can be obtained by estimating surface normals, described in Section 2.1.3.

2.1.1 LIDAR Point Clouds

A LIDAR sensor measures distances to objects in the surrounding by sending out laser light that reflects on targets and measures how long it takes for the light to return. With this, the distance can be calculated and thus giving a point cloud. The LIDAR sensor can be placed on a ground vehicle for local perception of the surroundings or on a aircraft to create a digital elevation map, among others. Fig-

ure 2.1 shows a LIDAR point cloud taken from a vehicle in an urban environment during the experiment presented in Chapter 6.

A LIDAR point cloud contains noise, which comes from multiple sources. Examples of this are drift effects, observed material and incidence and beam angles [6]. The drift effects are caused by temperature, the effect of surface color and the dependency of the distance to an object [44]. These sources can be modelled as sensor bias noise. Unmapped objects in the world such as leaves, trees and houses can also be seen as a type of noise in the point clouds. Laser beams are also a source of noise since there may be beams that do not return, creating lack of information in an area. If the laser beams hit an object but only one beam is returned then this causes outliers to appear which can be seen as noise. The sources of noise that are difficult to find an appropriate model for can be modelled as white noise.

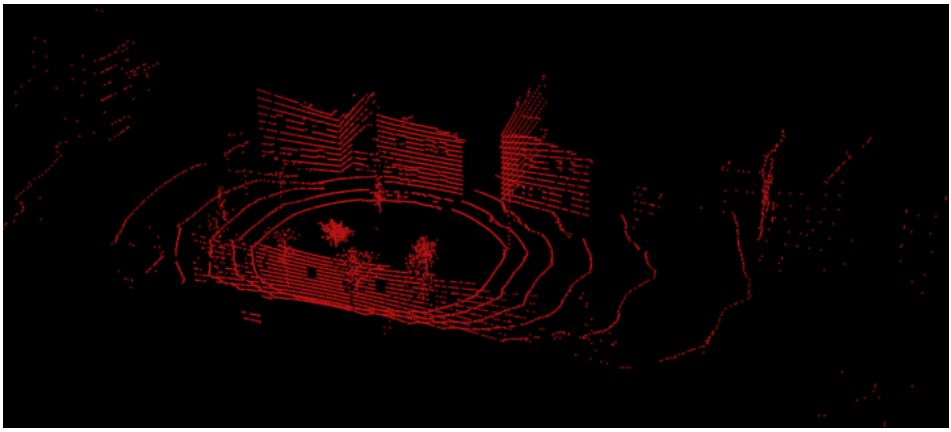


Figure 2.1: *A point cloud collected from a LIDAR sensor mounted on a vehicle, where the vehicle is driving in an urban area during one of the experiments in Chapter 6.*

2.1.2 Point Cloud Downsampling and Outlier Removal

When receiving a LIDAR point cloud, it contains outlier noise and a large amount of data resulting in computationally demanding processing. The number of points can be reduced with downsampling algorithms, which will lower the computational cost. Although, this can cause the point cloud to contain noise since points are being replaced by approximated ones instead. Outliers can be removed with outlier removal algorithms to reduce the noise in the point cloud. This has the opposite effect of downsampling, where it decreases the noise but increases the computational cost.

There are different types of downsampling methods. The one that is used in the system is a method that returns an approximated cloud that represents the orig-

inal cloud but with fewer points and is called *Voxel grid sampling*. This method is described below. The outlier removal method *Statistical outlier removal* used in the system is found below as well. C++ implementations of these methods can be found in the Point Cloud Library, described in Section 2.7.

Voxel grid sampling

A voxel grid filter reduces the number of points in a point cloud by creating a 3D voxel grid, which is equivalent to 3D boxes, that covers the whole cloud [23]. The voxels then contain different amount of points and some will not contain any at all, where in the latter case the voxel is discarded. The point cloud is then downsampled by choosing the centroids, which is the average of all points inside of each voxel. The size of the voxels decides how many points that are removed, where the larger the size, the more approximated the cloud becomes. It is a balance between the importance of the cloud's original structure and the importance of low computational costs, which increases linearly with the number of points and the size of the voxels. This approach represents the surface more accurately than by approximating the points with the center of the voxel, but has a slower computational time in comparison. An implementation of this method is found in [35].

Figure 2.2 shows a 2D example of voxel grid filtering, where the image to the left represents the original point cloud, the image in the middle shows the voxel grid applied and the calculated centroid of each box marked in green. The result after sampling is shown in the image to the right. Another faster version of voxel grid sampling is to approximate the points with the center of each voxel. However, the approach described in this section retains the original layout of the points more accurately.

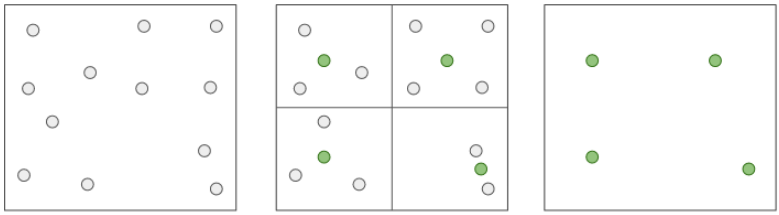


Figure 2.2: *An illustration of a 2D voxel grid filter process. The point cloud displayed to the left is the original. In the middle figure, it is divided into four voxels, where the green point is the computed centroid for all points in each box. The resulting point cloud is seen to the right.*

Statistical outlier removal

Statistical outlier removal is based on the computation of a distribution that is assumed to be Gaussian [36]. For each point, the mean distance to its neighbours

in the cloud is calculated. The number of neighbours that should be taken into account is chosen by the user. The distribution of all the mean distances are assumed to be Gaussian, and all points whose mean distances are outside of the standard deviation of the mean are removed since they are seen as outliers. The user can increase or decrease the number of points that are removed by choosing a standard deviation (STD) multiplier. An implementation of the statistical outlier removal algorithm is found in [33].

A 2D example of this is show in Figure 2.3, where four points have been tested so far. Two nearest neighbours are found for each point, and the mean of those distances are computed. It is clear that the points marked in red would be outside of the standard deviation of the mean, since they have a significantly larger mean distance to its neighbours than the rest. These points are seen as outliers and are removed.

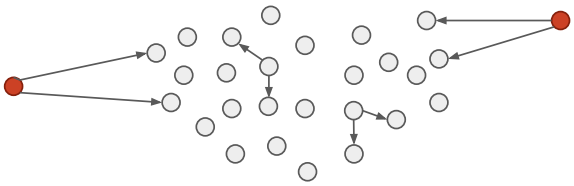


Figure 2.3: *An illustration of a statistical outlier removal process, where four points have been tested so far. After all points have been tested, the two points marked in red are removed as they have a larger mean distance than the rest of the points.*

2.1.3 Surface Normals

A surface normal is a vector perpendicular to an object, a plane or a line, and it is an important property of a geometric surface. The surface normals of each point in a point cloud can be approximated by either using meshing techniques to acquire a surface from the point cloud and computing the normals from the mesh, or by approximating the normals directly from the point cloud as in [36]. The latter method estimates a tangent plane in each point using surrounding neighbour points, and the normals are found perpendicular to the planes. The planes are estimated with least square fitting, which is further described in [36]. Pre-made C++ functions that performs the surface normal estimation of a point cloud is found in [34]. The amount of neighbour points that are considered when estimating the planes are determined by either setting a fixed number of neighbors, k, or to set a fixed radius of a sphere, r, where all points inside are neighbors. The size of r or k determines the accuracy of the normals. A large scale can suppress finer details, while a too small scale can cause lack of information which leads to normals with wrong direction. If there are no nearby points, the normal cannot be calculated as there is no way of determining how the plane should be oriented. Figure 2.4 illustrates a 2D example of how different radii affect the normals.

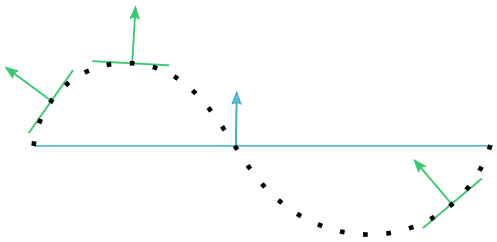


Figure 2.4: *Surface normal estimation in 2D, seen from the side. The normals are only shown for a few points, and they are created with different sizes of the radius, r. The size of the tangent plane, which is drawn perpendicular to the normal, corresponds to the size of r for visual purposes. The blue normal, belonging to the middle point, has a large r that takes all points into account when estimating the plane. This leads to a loss of the characteristics of the curve. The green normals are created with a better suited size of r that only takes the nearest points into account when estimating the tangent planes, this retains the shape of the curve.*

2.2 Iterative Closest Point

The ICP algorithm is used in the *global position estimation* step in the system. It is a widely used scan registration method that aligns a *source* point cloud to a fixed *target* point cloud by finding the optimal translation and rotation that minimizes the distance between them [3]. There are different variations of ICP, for example point-to-plane which has been shown in [32] to have better convergence than for example the most standard variation point-to-point. The specific parts of point-to-plane are described in the next section, the general algorithm is described below.

The source cloud S and target cloud O is denoted as

$$S = \{s_i\}, \quad i \in 1, ..., N_s, \tag{2.1}$$

$$O = \{o_j\}, \quad j \in 1, ..., N_o, \tag{2.2}$$

where s_i and o_j are points in the source and target cloud respectively, and N_s and N_o are the number of points in each cloud.

To achieve the data alignment, the first step is to find a corresponding point in the target cloud for each of the points in the source cloud. These correspondences are found by minimizing a distance metric

$$d(s_i, O) = \min_{o_j \in O} \|o_j - s_i\|, \tag{2.3}$$

where the distance from s_i to each point in O is computed, and the point in O that is closest is chosen and further on denoted o_{i^*}. This is iterated for each s_i,

and all the correspondences are saved.

The next step is to apply a rotation R and translation t to the source cloud that minimizes the distance between it and the target cloud. R and t are found by minimizing a cost function $E(R, t)$, which uses the correspondences found in the first step. This cost function varies depending on which type of ICP is used. After R and t are applied to the source cloud, the algorithm iterates until convergence, where new correspondences are found in each iteration. The rotation R and translation t are stored inside a 4×4 matrix called a transformation matrix

$$T_k = \begin{bmatrix} R_k & t_k \\ 0_{1\times 3} & 1 \end{bmatrix}, \tag{2.4}$$

which is the k^{th} iteration used to transform and move the source point cloud closer to the target point cloud. All transformation matrices are multiplied with its former matrix creating the final transformation matrix, T_{ICP},

$$T_{ICP} = \prod_{k=1}^{n_{ICP}} T_k, \tag{2.5}$$

where n_{ICP} is the total amount of iterations performed. In Figure 2.5 an ideal example of the ICP point-to-plane process is shown.

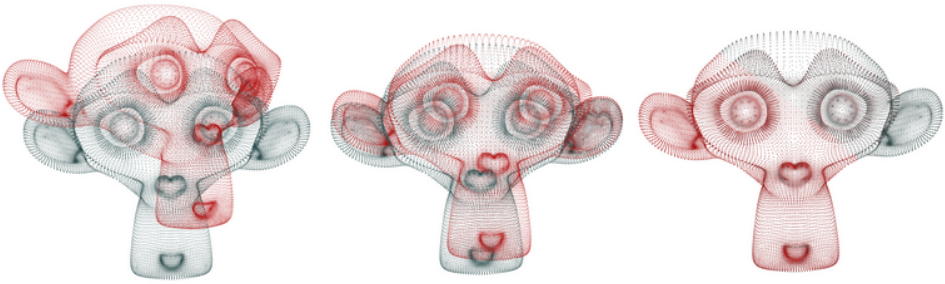


Figure 2.5: *An example of an ICP iteration process on a point cloud. The red point cloud is the source cloud and the grey is the target cloud. The image to the left shows the starting positions of the two clouds, where the source cloud is an exact copy of the target but with a transformation applied. In the middle image, the ICP algorithm has been iterated 7 times, and 23 times in the right image which results in fully aligned clouds. Point cloud model is created by the Blender Foundation [10].*

2.2.1 Point-to-plane

The point-to-plane variant of ICP is often better at finding the optimal transformation than the standard point-to-point type, as shown in [32]. The correspondences are found with (2.3). By estimating surface normals, as described in Sec-

tion 2.1.3, tangent planes are created in each of the points in the target cloud. The surface normals for each point in O are denoted n_j.

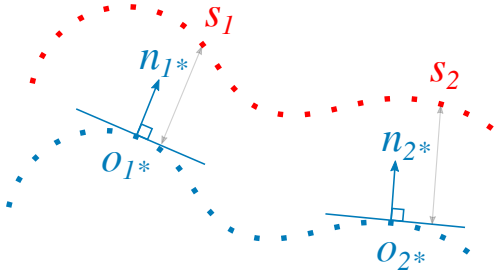


Figure 2.6: *2D example of point-to-plane ICP. The distances marked with grey arrows are minimized in the error metric cost function, only two distances are shown for visual purposes.*

The error metric cost function is defined as the sum of the square distances between each point in the source cloud, and the plane that belongs to the corresponding point in the target cloud

$$E(R,t) = \sum_{i=1}^{N_s} \Big[\Big(o_{i^*} - (Rs_i + t) \Big) n_{i^*} \Big]^2, \tag{2.6}$$

where n_{i^*} is the surface normal of the corresponding point o_{i^*} in the current iteration of ICP. The rotation and translation is applied to the source cloud as

$$s_{i+1} = Rs_i + t. \tag{2.7}$$

2.2.2 Sources of Error

There are a few sources of error that have to be considered when implementing ICP [7]. The first one is **wrong convergence**, where ICP converges to a local minimum instead of the global. This often proves to be the dominant error, since ICP itself does not take this into account when minimizing the cost function. To avoid this error, an initial alignment can be performed to transform the source cloud closer to the global minimum before applying ICP. This can be done either with a qualified guess, or by randomly choosing different initial positions. Another method to avoid local minimums is proposed in [25], where stochastic ICP is introduced. As the article describes, a random Gaussian noise translation is added to the source cloud before each iteration of the ICP algorithm, which allows the algorithm to relocate itself from a local minimum.

The second source is the **under-constrained situations**, for example in environments where there is not enough information to estimate the position and orientation of the object in question. It can be seen as "sliding" occurring in one of

the coordinate axes in the different results, because the ICP cannot pinpoint the exact placement of the object.

The third source of error is the **sensor noise** which can change the outcome of the solution, even though the source cloud is initially positioned in the region of attraction of the true solution before the ICP algorithm is performed. The reason is that the source and target cloud are not as similar as they ideally would be. To decrease this error, outlier removal algorithms described in Section 2.1.2 can be used but only if done so properly, as wrongly defined parameters in the filtering can increase the error. The outliers in the point cloud can be seen as singular points or small groups of points spread out in the cloud. These points skew the alignment due to the ICP algorithm trying to find correspondences for every point in the cloud [26]. To decrease this problem, it is possible to filter the outliers before performing ICP, such that the points are discarded if they are too far away from the points in the target cloud.

2.3 Uncertainty

The uncertainty of the ICP algorithm can be observed by estimating the covariance, which shows the variability between each pair of elements of a given random vector. How accurate the estimate is depends on what sources of error it considers. An estimated covariance matrix can also be used in Kalman filtering. There are multiple methods to estimate it, where there is a trade-off between execution time and accuracy. Monte Carlo simulations is one example of where the accuracy is high but so is the execution time as well.

One estimation method is to use the Hessian to calculate the covariance [4, 7] and another one uses correspondences between two point clouds [27]. What they have in common is that both linearize the cost function used by the ICP around the convergence point. This leads to the covariance only taking *sensor noise* — from the three sources of error mentioned in Section 2.2.2 — into consideration [6].

2.3.1 Cost function linearization

To be able to compute the covariance of ICP with Hessian and correspondences, the cost function mentioned in Section 2.2.1 must be linearized.

The roto-translation vector in 3D can be written as $X = [X_r^T \ X_t^T]$, where X_r is the rotational vector with the elements X_{r1}, X_{r2} and X_{r3} around the x, y and z axis respectively, and X_t is the translational vector with elements X_{t1}, X_{t2} and X_{t3}. The cost function is nonlinear because a received angle, X_{r1}, is not linear in the rotation matrix, but the matrix can be linearized by assuming small incremental rotations and approximating $\cos(X_{r1}) = 1$ and $\sin(X_{r1}) = X_{r1}$ [13]. A

small rotation around the x axis of angle X_{r1} gives

$$\begin{aligned} R_x &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(X_{r1}) & -\sin(X_{r1}) \\ 0 & \sin(X_{r1}) & \cos(X_{r1}) \end{bmatrix} \approx \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -X_{r1} \\ 0 & X_{r1} & 1 \end{bmatrix} \\ &\approx I + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -X_{r1} \\ 0 & X_{r1} & 0 \end{bmatrix} = I + X_{r1}(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix})_\times, \end{aligned} \tag{2.8}$$

where $(a)_\times \in \mathbb{R}^{3\times3}$ denotes a skew symmetric matrix associated with the cross product with $a \in \mathbb{R}^3$ (a is an arbitrary vector, which for (2.8) would be $[1 \;\; 0 \;\; 0]^T$). Small rotations around the y and z axis can be calculated in a similar way as in (2.8) using X_{r2} and X_{r3} and all three rotations can be combined to a full rotation, which is approximated as

$$R = \begin{bmatrix} 1 & -X_{r3} & X_{r2} \\ X_{r3} & 1 & -X_{r1} \\ -X_{r2} & X_{r1} & 1 \end{bmatrix} = I + (X_r)_\times. \tag{2.9}$$

If the two input clouds have been aligned beforehand (which is necessary as ICP can only guarantee convergence locally) then it is possible to take the rotation matrix, $R \in SO(3)$, in (2.6) as close to identity. This justifies the possibility to use the linearizing approximation given in (2.9). By defining the row vector $t = X_t$, $X_t \in \mathbb{R}^3$ from the vector X and employing (2.9) on the cost function in (2.6) for point-to-plane gives the following:

$$E(X) = \sum_{i=1}^{N_s} \Big[\Big(o_{i^*} - (X_t + s_i + (X_r)_\times s_i) \Big) n_{i^*} \Big]^2. \tag{2.10}$$

The multiplication of a skew symmetric matrix with another vector can be written as a cross product between the two vectors, $(X_r)_\times s_i = X_r \times s_i$, and (2.10) can be rewritten as

$$E(X) = \sum_{i=1}^{N_s} \Big[\Big(o_{i^*} - (X_t + s_i + X_r \times s_i) \Big) n_{i^*} \Big]^2. \tag{2.11}$$

Equation (2.11) can be written as

$$E(X) = \sum_{i=1}^{N_s} \| y_i - B_i X \|^2, \tag{2.12}$$

where y_i and B_i are defined as

$$y_i = n_{i^*}^T (s_i - o_{i^*}), \qquad B_i = [-(s_i \times n_{i^*})^T \;\; -n_{i^*}^T]. \tag{2.13}$$

The term $(s_i \times n_{i^*})$ comes from the scalar triple product, $(X_r \times s_i) \cdot n_{i^*} = (s_i \times n_{i^*}) \cdot X_r$.

2.3.2 Covariance with Hessian

As mentioned previously, assuming the point clouds S and O start close to each other, the ICP cost function can take the form of sum-of-squares (2.12). The estimate computed by the ICP is denoted with $\hat{X}$ and the true transformation is X^*.

The ICP is biased if it converges to a local minimum and therefore also have the wrong convergence. Here it can be considered unbiased because of the assumption that the clouds start close to each other (an initial alignment was performed) which means that the ICP converges to a global minimum. Due to this it is possible to assume that the ICP is an unbiased estimator $E(\hat{X}) = X^*$ and the covariance is thus

$$P = E\Big((\hat{X} - E(\hat{X}))(\hat{X} - E(\hat{X}))^T\Big) = E\Big((\hat{X} - X^*)(\hat{X} - X^*)^T\Big). \tag{2.14}$$

The minimizing solution $X = \hat{X}$ is obtained by taking the gradient of the cost function in (2.12) and finding its critical point, which is when the gradient is equal to zero. Thus, $\hat{X}$ is written as

$$\hat{X} = A^{-1}b = \Big[\sum_i B_i^T B_i\Big]^{-1} \sum_i B_i^T y_i. \tag{2.15}$$

$A = \sum_i B_i^T B_i$ represents the (half) Hessian of the cost function and is symmetrical ($A^T = A$) [5].

The residuals are defined as

$$r_i = y_i - B_i X^* \quad \Rightarrow \quad y_i = B_i X^* + r_i, \tag{2.16}$$

and can be used to replace y_i in (2.15) which gives the covariance matrix:

$$\begin{aligned} P &= E\Big(\Big(A^{-1}\sum_i B_i^T r_i\Big)\Big(A^{-1}\sum_i B_i^T r_i\Big)^T\Big) \\ &= A^{-1}\sum_i\sum_k \Big(B_i^T E(r_i r_k^T) B_k\Big) A^{-1}. \end{aligned} \tag{2.17}$$

An error model for the residuals, r_i, has to be assumed in order to assess (2.17). Replacing B_i and y_i with their terms in (2.16) gives the residual $r_i = w_i \cdot n_{i^*} = n_{i^*}^T w_i$. With this it is possible to rewrite (2.17), giving the final covariance

$$P = \Big[\sum_i B_i^T B_i\Big]^{-1} \sum_i\sum_k \Big(B_i^T n_{i^*}^T E(w_i w_k^T) n_{i^*,k} B_k\Big)\Big[\sum_i B_i^T B_i\Big]^{-1}, \tag{2.18}$$

where $w_i \in \mathbb{R}^3$ is the post-alignment error at the i^{th} pair of points due to sensor errors. The post-alignment errors, w_i can be assumed as *Random noise errors*, where it is assumed that the errors are independent and identically isotropically distributed as

$$w_i \sim \mathcal{N}(0, \sigma^2 I_3).$$

$\mathcal{N}(0, \sigma^2 I_3)$ denotes an independent Gaussian distribution with zero mean and σ^2 variance. With these assumptions, the error $E(w_i w_k^T) = E(w_i)E(w_k^T) = 0$, $i \neq k$ reduces the double sum to a single sum. $E(w_i w_k^T) = \sigma^2 I_3$ when $i = k$ simplifies (2.18) to

$$\begin{aligned} P &= \Big[\sum_i B_i^T B_i\Big]^{-1} \sigma^2 \sum_i B_i^T B_i \Big[\sum_i B_i^T B_i\Big]^{-1} = \sigma^2 \Big[\sum_i B_i^T B_i\Big]^{-1} \\ &= \sigma^2 A^{-1}. \end{aligned} \tag{2.19}$$

The matrix A will have a fixed size of 6×6 but the computational cost comes from the increasing amount of points in the point cloud that has to be summarized in A and the computation of its inverse. Random noise errors has been proven to give a too optimistic covariance. This is because the Gaussian noise that was assumed in the errors w_i becomes negligible for increasing number of scanned points [4].

2.3.3 Covariance with Correspondences

The method to estimate covariance with correspondences tries to find correspondences between points in the source cloud to points in the target cloud in the last iteration of ICP. It is done by taking the closest neighbouring point. The corresponding point pairs are then used in the estimation of the covariance matrix, which is described below.

Consider a function F with output $\hat{X}$ and input b, i.e $\hat{X} = F(b)$, that minimizes an objective function $E(\hat{X})$. Using Taylor series expansion of F at a value $b = b_0$ gives

$$\hat{X} = \Big(F(b)|_{b=b_0}\Big) \approx F(b_0) + \frac{\partial F}{\partial b_0}(b - b_0) = F(b_0) + \frac{\partial F}{\partial b_0} b - \frac{\partial F}{\partial b_0} b_0, \tag{2.20}$$

where $\frac{\partial F}{\partial b_0} = \frac{\partial F}{\partial b}|_{b=b_0}$. The covariance of an algebraic expression on the form of $\hat{X} = Db + c$, where c is a constant, is

$$P = D\text{cov}(b)D^T, \tag{2.21}$$

assuming stochastic variables in b with Gaussian distribution. This equation can be used for (2.20), where $F(b_0)$ and $\frac{\partial F}{\partial b_0}(b_0)$ represent the constant c.

$$P \approx \frac{\partial F}{\partial b_0} \text{cov}(b) \frac{\partial F}{\partial b_0}^T. \tag{2.22}$$

The implicit function theorem, which is explained more in detail in the appendix of [7], is written as

$$\frac{\partial F}{\partial b_0} = -\left(\frac{\partial^2 E}{\partial X^2}\right)^{-1} \left(\frac{\partial^2 E}{\partial b \partial X}\right), \tag{2.23}$$

where E is the same error cost function mentioned in previous section, (2.6). Using (2.23) in (2.22) gives the final covariance equation:

$$P \approx \left(\frac{\partial^2 E}{\partial X^2}\right)^{-1}\left(\frac{\partial^2 E}{\partial b \partial X}\right)\text{cov}(b)\left(\frac{\partial^2 E}{\partial b \partial X}\right)^{T}\left(\frac{\partial^2 E}{\partial X^2}\right)^{-1}. \tag{2.24}$$

As the matrix, $\left(\frac{\partial^2 E}{\partial X^2}\right)^{-1}$, is symmetric, taking the transpose of it would not make a difference. It can be seen from (2.24) that three terms, $\left(\frac{\partial^2 E}{\partial X^2}\right)^{-1}$, $\left(\frac{\partial^2 E}{\partial b \partial X}\right)$ and $\text{cov}(b)$ have to be calculated. The three parameters in the terms are X, b and E, where X is the same state vector as previously defined. Note that the state vector is reversed in all the equations defined in [27]. The parameter b is defined as the n_c sets of correspondences $\{S_m, O_m\}$, where S_m and O_m are vectors containing the coordinates for the two point clouds at correspondence m. The computations of the three terms can be found in [27]. This method is based on the pioneering formula by Censi in [7], but it is known to be very optimistic, see [22].

2.4 Unscented Transform based ICP and Covariance

The method of estimating covariance with Hessian and correspondences can be used together with the ICP algorithm described in Section 2.2. A newly discovered way of calculating the covariance of ICP is to use the *unscented transform* (UT) [6], which requires a different approach to the ICP algorithm. This section presents UT, the ICP adaption and how the covariance is computed.

2.4.1 Unscented Transform

The unscented transform is a method for approximating the statistics of a random variable subject to a nonlinear transformation [16, 43]. The procedure is to choose a set of points, called *sigma points*, in such a way that their sample mean and covariance are $\overline{x}$ and P_{xx}, which corresponds to the true mean and covariance of the prior random variable. The nonlinear function is applied to each point and yields a cloud of transformed points with $\overline{y}$ and P_{yy} as the statistics of the posterior. The method has some resemblance to the Monte-Carlo type methods, with the difference that the samples are not randomly drawn, but instead drawn from a specific deterministic algorithm.

The n-dimensional random variable, X, with mean $\mu^{\mathcal{X}}$ and covariance P_{xx} is ap-

proximated by $2n + 1$ sigma points, which are given by

$$\mathcal{X}_0 = \mu^{\mathcal{X}}, \tag{2.25}$$

$$\mathcal{X}_i = \mu^{\mathcal{X}} + (\sqrt{(n+\lambda)P_{xx}})_i \qquad i = 1, 2, \ldots, n \tag{2.26}$$

$$\mathcal{X}_{i+n} = \mu^{\mathcal{X}} - (\sqrt{(n+\lambda)P_{xx}})_i \qquad i = 1, 2, \ldots, n \tag{2.27}$$

$$W_0^m = \frac{\lambda}{n+\lambda}, \tag{2.28}$$

$$W_0^c = \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta), \tag{2.29}$$

$$W_i^m = W_i^c = \frac{1}{2(n+\lambda)} \qquad i = 1, 2, \ldots, 2n, \tag{2.30}$$

where n is the dimension of the state vector X, $(\sqrt{(n+\lambda)P_{xx}})_i$ is the i^{th} column of the matrix square root of $(n+\lambda)P_{xx}$ and W_i is the weight associated with the i^{th} point. W^m is the weight for the mean and W^c is the weight for the covariance. The parameter λ is a scaling parameter, $\lambda \in \mathbb{R}$, defined as

$$\lambda = \alpha^2 \cdot (n + \kappa) - n. \tag{2.31}$$

The parameter α determines the spread of the sigma points, κ is a secondary scaling parameter and β is used for incorporating prior knowledge of the distribution of X.

The transformed sigma points can be obtained by propagating each point through the transformation function as

$$\mathcal{Y}_i = f(\mathcal{X}_i). \tag{2.32}$$

The weighted average of the transformed points gives the mean

$$\mu^{\mathcal{Y}} = \sum_{i=0}^{2n} W_i^m \mathcal{Y}_i, \tag{2.33}$$

and the weighted outer product of the transformed points gives the covariance

$$P_{yy} = \sum_{i=0}^{2n} W_i^c (\mathcal{Y}_i - \mu^{\mathcal{Y}})(\mathcal{Y}_i - \mu^{\mathcal{Y}})^T. \tag{2.34}$$

The sigma points capture the same mean and covariance no matter what matrix square root is used, which means that stable and numerically efficient methods can be used and an example of this kind of method is the Cholesky decomposition.

Performing Cholesky decomposition on the covariance matrix of the prior state estimate, P_{xx}, gives a lower triangular matrix, L. The sigma points from (2.25)-(2.27) can thus be combined into a matrix

$$\mathcal{X} = \mu^{\mathcal{X}} + \begin{bmatrix} 0_{6\times 1} & L^T & -L^T \end{bmatrix}. \tag{2.35}$$

2.4.2 ICP Adaptions

With the theory of the UT it is possible to adapt the ICP algorithm, which is needed for the computation of the covariance estimate.

The i^{th} column vector in $\mathcal{X}$ has to be converted to a transformation matrix in order to transform the point cloud with the sigma points before performing ICP. This is done by calculating $\exp(\mathcal{X}_i)$, where $\exp(\cdot)$ denotes the exponential map of $SE(3)$ and maps the elements of $\mathcal{X}_i$ to the format of a transformation matrix. The equations and procedure of the exponential map are described more in detail in the appendix of [2]. The ICP algorithm is performed for each sigma point, giving new transformation matrices, which have to be propagated in the covariance estimate. This is described more below.

2.4.3 Covariance Estimate

The covariance of the unscented transform based ICP can be divided into two parts. The first part depends on the unscented transform and accounts for *wrong convergence* and *under-constrained*. The second part is a mix between the method based on Hessian in Section 2.3.2 and the method based on correspondences in Section 2.3.3 and accounts for *sensor noise* [6]. The two parts give two covariance matrices which are summed together to obtain the final covariance estimate

$$P = (I_6 - J)P_{xx}(I_6 - J)^T + KP_{sensor}K^T = P_{yy} + P_{sn}, \tag{2.36}$$

where P_{sensor} is the covariance matrix for the sensor noise, w, in the scanner, and I_6 is a 6×6 identity matrix.

Covariance with unscented transform

Following equation (2.32), the sigma points from (2.35) are propagated through the ICP transformation function, giving the new set of sigma points, $\mathcal{Y}_i$ [6]. $\mathcal{Y}_i$ can be inserted to (2.33), giving the components needed to estimate the covariance matrix, P_{yy}, in (2.34).

Covariance considering sensor noise

The second term of the final covariance matrix considers an unknown bias, b. The matrix is computed in the same way as the method based on Hessian in Section 2.3.2, except that the noise is $w = b + v_i$, where v_i is a white noise with variance σ^2. More details about the procedure is found in [6]. The covariance estimate considering sensor noise is defined as

$$P_{sn} = KP_{sensor}K^T = \sigma^2 A^{-1} + A^{-1}C\text{cov}(b)C^T A^{-1}, \tag{2.37}$$

where $A = \sum_i B_i^T \cdot B_i$, as defined before, and $C = \sum_i B_i^T$. The first term in (2.37) is the same as the expression in (2.19) and the second term is the one that is similar to (2.24), where A and C are equivalent to $\frac{\partial^2 E}{\partial X^2}$ and $\frac{\partial^2 E}{\partial z \partial X}$ respectively.

2.5 Covariance Evaluation Method

The methods to compute and estimate the covariance have to be assessed to determine if the estimates are too optimistic or pessimistic. The evaluation method used in this thesis is called *Normalized Estimation Error Squared (NEES)*, and is defined as

$$\epsilon_{X,k} = \tilde{X}^T P^{-1} \tilde{X}, \tag{2.38}$$

where $\tilde{X} = \hat{X} - X$ and P is the estimated covariance matrix. The goal is to get the value of $NEES$ as close to the number of states that the variable has, which in 2D is three and in 3D it is six. The estimation is considered optimistic if the value is above it and pessimistic if it is below [8]. The reason behind using the number of states comes from $\epsilon_{X,k}$ being χ^2 (chi squared) random variables with n degrees of freedom (also known as number of states), where the mean of χ^2 is n and the variance is $2n$.

2.6 Position Estimation

To get an accurate position estimate, an EKF can be used to merge position information provided by different sensors or algorithms. Based on their covariance estimates, the EKF weighs the information and provides a single estimate. Theory about EKF, as well as an odometry motion model that can be used in the EKF, is presented in this section.

2.6.1 Extended Kalman Filter

The Kalman filter, described in [40], is a Gaussian filter that first predicts and then corrects the next states in a linear system. In the prediction step, a model of the system dynamics is used to estimate the states at the next time step. The covariance of the state vector is predicted as well. In the correction step, the model is adjusted with observation updates. The filter decides how trustworthy the observation is through the Kalman gain, which evaluates the error in the prediction along with the error in the observation.

The EKF is a variant of the Kalman filter that allows nonlinear functions to describe the measurements and/or observations. For example when estimating the movement of a car when it is moving in a circular motion, a nonlinear model must be used. The EKF linearizes the nonlinear functions using Taylor Series, and its algorithm is found below [40]. The state and observation models are denoted

$$X_k = f(X_{k-1}, u_k) + w_k, \tag{2.39}$$

$$z_k = h(X_k) + v_k, \tag{2.40}$$

where $f(X_{k-1}, u_k)$ and $h(X_k)$ are nonlinear functions of the previous states X_k and the control vector u_k at time step k. The process and observation noise are denoted w_k and v_k respectively. They are Gaussian distributed stochastic variables with mean zero and variance Q and R, which are covariance matrices of the

process and observation noise.

The prediction step is described by

$$\hat{X}_{k|k-1} = f(\hat{X}_{k-1|k-1}, u_k), \tag{2.41}$$

$$P_{k|k-1} = J_f P_{k-1|k-1} J_f^T + Q_k, \tag{2.42}$$

where $\hat{X}_{k|k-1}$ is the predicted state vector and $P_{k|k-1}$ is the predicted covariance. J_f is the Jacobian, the partial derivatives, of the state model $f(X_{k-1})$. The Jacobian linearizes the function at the current estimate. Q_k is the covariance matrix of the process noise.

The correction step is described by

$$K_k = P_{k|k-1} J_h^T (J_h P_{k|k-1} J_h^T + R_k)^{-1}, \tag{2.43}$$

$$\hat{X}_{k|k} = \hat{X}_{k|k-1} + K_k(z_k - h(\hat{X}_{k|k-1})), \tag{2.44}$$

$$P_{k|k} = (I - K_k J_h) P_{k|k-1} (I - K_k J_h)^T + K_k R_k K_k^T, \tag{2.45}$$

where K_k is the Kalman gain. The gain becomes large ($K_k \rightarrow I_{n\times n}$, where n is the number of states) if the observation error matrix R_k is small, which means that the filter will trust the observation. Instead, if R_k is large, the gain will become small ($K_k \rightarrow 0_{n\times n}$) and the filter will mostly ignore the observation. $J_h = \frac{\delta h}{\delta \hat{X}}$ is the Jacobian of the measurement equation $h(X_k)$. $\hat{X}_{k|k}$ is the corrected state vector that updates $\hat{X}_{k|k-1}$. The expression inside the parenthesis in (2.44) is the innovation, which is the difference between the computed observation and the estimated. The state covariance $P_{k|k}$ is updated as well, by using Joseph's form in (2.45) since it does not assume that the Kalman gain is optimal and that it guarantees symmetry [38]. After the correction step is performed, the algorithm starts over at the prediction step.

2.6.2 Odometry

Odometry can be used to estimate the movement of a vehicle. It can either be used alone as dead-reckoning, or as state prediction in an EKF, Section 2.6.1. The odometry calculations are based on the angle and speed of the vehicle which can be computed from data provided by an inertial measurement unit (IMU), or through rotary encoders attached to the wheels. For each time step, a new position for the vehicle is estimated and added to the previous to create a path.

Sample odometry motion model

There are multiple motion models that describe the movement of a vehicle, one is sample odometry motion model [40]. The 2D method uses the state vector $X = [x_k \ y_k \ \theta_k]^T$, where θ is rotation around the z-axis. It uses previous calculated states $[x_{k-1} \ y_{k-1} \ \theta_{k-1}]^T$, and the difference between the previous and

current odometry measurements $[\Delta x_k^o \ \Delta y_k^o \ \Delta\theta_k^o]^T = [x^o{}_{k-1} \ y^o{}_{k-1} \ \theta^o{}_{k-1}]^T - [x^o{}_k \ y^o{}_k, \ \theta^o{}_k]^T$ to predict the next step. If odometry is used for example in an EKF, state prediction of this kind is needed. The odometry is used for calculating the relative measured motion parameters

$$\delta_{trans} = \sqrt{(\Delta x_k^o)^2 + (\Delta y_k^o)^2}, \tag{2.46}$$

$$\delta_{rot1} = \arctan2(\Delta y_k^o, \Delta x_k^o) - \theta^o{}_{k-1}, \tag{2.47}$$

$$\delta_{rot2} = \Delta\theta_k^o - \delta_{rot1}, \tag{2.48}$$

where arctan2 is defined in [21] and an illustration of the motion parameters is found in Figure 2.7.

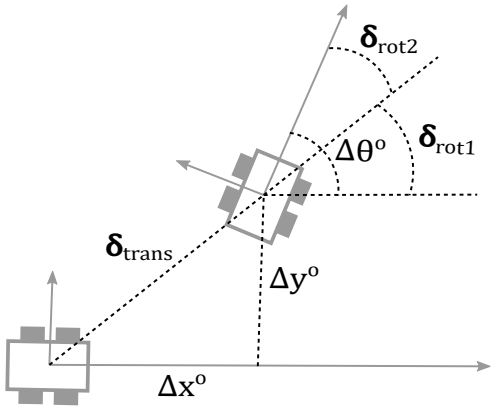


Figure 2.7: *Illustration of how the odometry motion parameters δ_{rot1}, δ_{rot2} and δ_{trans} relate to the motion of the vehicle.*

The noise of the motion parameters are assumed to be random normal distributed $\mathcal{N}(\mu, \sigma)$, where μ is the mean and σ is the standard deviation. More details about how the noise distribution can be modelled is found in [40]. The next coordinates $[x_k \ y_k \ \theta_k]^T$ are then predicted by

$$x_k = x_{k-1} + \delta_{trans} \ \cos(\theta_{k-1} + \delta_{rot1}), \tag{2.49}$$

$$y_k = y_{k-1} + \delta_{trans} \ \sin(\theta_{k-1} + \delta_{rot1}), \tag{2.50}$$

$$\theta_k = \theta_{k-1} + \delta_{rot1} + \delta_{rot2}, \tag{2.51}$$

which can be used as state prediction in (2.41).

2.7 Point Cloud Library

In this section, information about one important software library used in the global localization system is found. It is called Point Cloud Library (PCL) and is the main library for handling point clouds, which is a great part of this thesis work, it also contains ICP algorithms etc.

PCL is a large scale, free, BSD licensed library for processing of 2D and up to 4D point clouds or imagery and is fully integrated with the Robot Operating System, ROS. The library uses many other open-source libraries, for instance *Eigen*, *Intel Threading Building Blocks* (TBB) and *Fast Library for Approximate Nearest Neighbors* (FLANN) are a few of them. Point Cloud Library contains a multitude of state-of-the-art algorithms, such as filtering, model fitting, feature estimation, registration etc. With these algorithms it is possible to, for example, filter out outliers from noisy data, calculate the normals of a point cloud or stitch 3D clouds together. It is also possible to visualize the point clouds and surface data with its own visualization library, based on the open-source software system *Visualization ToolKit* (VTK) [37].

3

Pre-Processing

In this chapter, the data used in the designed global localization system is explained and it is described how it is pre-processed. This is the first step of the system, seen in Figure 1.1. Figure 3.1 shows the inputs and output of this step, along with the different parts which are further explained in the following sections.

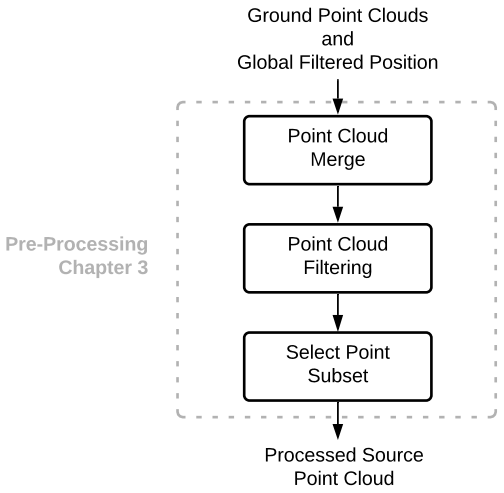


Figure 3.1: *The pre-processing step is divided into these three parts. The inputs are ground point clouds given by FOI's system and global position estimates representing the current position of the vehicle, computed by the EKF in the global position filtering step described in Chapter 5. The output of this step is a processed point cloud.*

3.1 The Data

Before the different parts of this *pre-processing* step are explained, the data used in the system is presented. The idea of using an ICP algorithm to locate a vehicle in a global map requires source and target point clouds, which is described more in the theory found in Section 2.2.

3.1.1 Point Clouds

In this application, the data used to build the source cloud is obtained from a Velodyne Puck VLP-16 LIDAR sensor [28] which is attached to the roof of the vehicle, see Figure 3.2. The sensor is elevated to avoid the laser being reflected on the car and has 16 channels with a 360°environmental view and 100 m range that allows the sensor to create point clouds that corresponds to the environment around the vehicle. The sensor has an integrated IMU xsens MTi-G-710 sensor, and also an integrated GNSS. A Sokkia GRX1 GNSS receiver [41], which uses *real time kinetic* (RTK) positioning, is also attached to the roof for ground-truth data. It is, during good conditions, a more accurate GNSS with only a few centimeters of error. Unfortunately, the Sokkia GRX1 did not provide accurate positions during the data collection, which is further explained in Section 6.3.1. CAN bus data from the vehicle is also measured, which is used in FOI's SLAM system to calculate odometry. A computer in the car saves all the data to a rosbag [31].

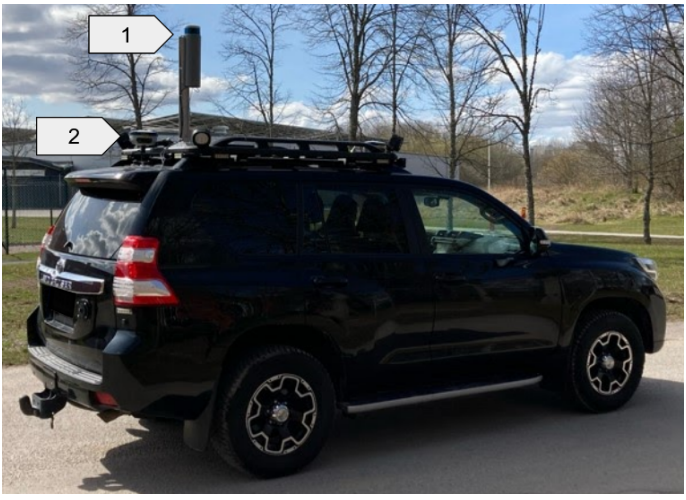


Figure 3.2: *The vehicle carrying the sensors. Marker one points to the LIDAR sensor, and marker two points to the Sokkia GRX1.*

Figure 3.3 shows a raw point cloud from one scan of the LIDAR sensor, where the vehicle carrying the sensor is facing right with trees on its left side, and a field on its right. This data is collected in collaboration with FOI.

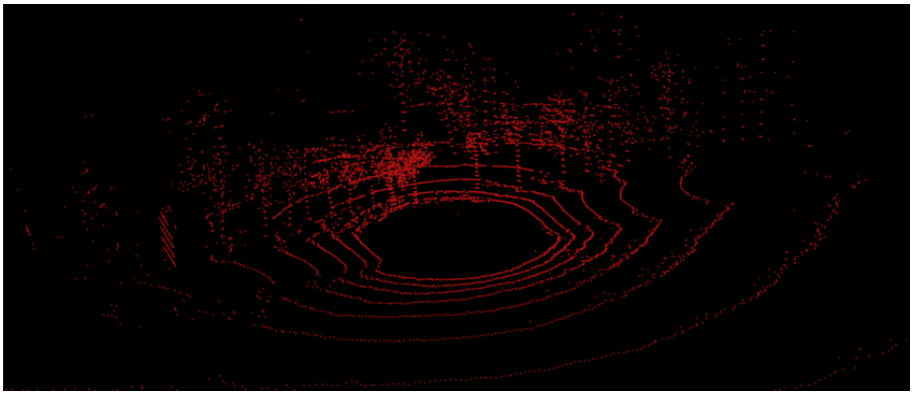


Figure 3.3: *A LIDAR point cloud. The vehicle is moving from left to right with trees on one side and an open field on the other.*

The target point cloud, that the source point cloud is going to be matched against, is an elevation map of the area. The target application is to use elevation maps distributed by the Swedish authority Lantmäteriet [17], where the data is collected from an airborne LIDAR sensor. The present data, already available at FOI, is from measurements by Linköping municipality who procured its measurements and post processing from a private company in 2013 [24]. This data is representative but has a higher density of up to 15-20 points per square-meter in open areas compared to the 0.5-1 points per square-meter available from Lantmäteriet. The elevation map only contains a ground-surface model of the area, while all the trees, road signs, houses etc. have been removed by approximating the surface underneath. In this case the private company did the classification of ground points while FOI provided the final ground model. The quality of the more generally available target data from Lantmäteriet, and how objects typically are removed are reported in [18].

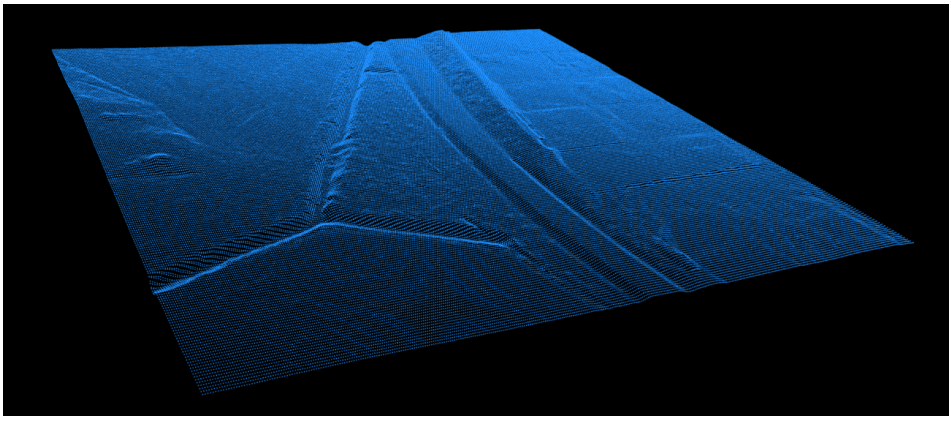


Figure 3.4: *A piece of an elevation map with a road going through it. It is used as target cloud in the system.*

Figure 3.4 shows a piece of an elevation map used in the system. A road is going through the map from top to bottom, and houses that in real life are located on the right side of the road are now gone. An elevation point map that contains a whole route that a vehicle travels can become very large and cause computational costs. Therefore, to make the system more time efficient, only a piece of the map around the estimated position of the vehicle is sent to the ICP algorithm. The position estimate is given by the EKF, in the *global position filtering* step, described in Section 5.1. This also provides some robustness to the system as the map is narrowed down.

To be able to achieve a match with the ICP algorithm, the source cloud must be as similar to the target cloud as possible. Therefore, only the points of the ground from the LIDAR sensor are used to create the source cloud. These points are extracted in an existing part of FOI's system, using code from [9]. The code is not completely accurate since some points representing objects can be mistaken for ground. These mistaken points are seen as noise and is discussed more below.

3.1.2 Noise

As mentioned previously, one source of noise that can clearly be seen in the point clouds is the unmapped objects that have been mistaken for ground when FOI's system extracted them. They can be seen as points randomly spread out in certain areas and floating above the rest of the point cloud where an unmapped object is known to be located. This noise can be classified as outliers and some of it can be removed with filtering.

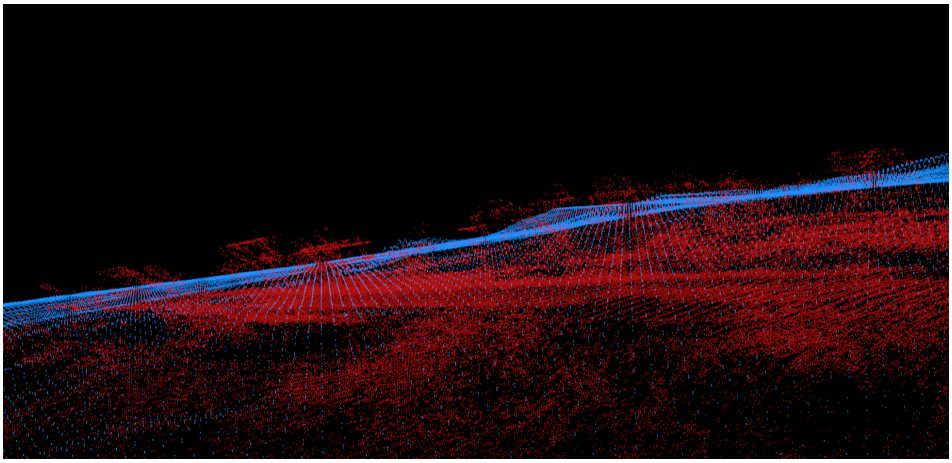


Figure 3.5: *A source cloud (in red), seen from the side, with noise floating above the more dense part of the cloud.*

Figure 3.5 shows an example of this, after FOI's system extracts the ground points, where the point cloud is seen from the side. This area is known to have plenty

of vegetation, which was observed when the data was recorded. The extraction has removed some of it but clearly not all, and especially not smaller less dense bushes etc. Points that are not removed are seen in the figure, where they are floating on the top part of the red point cloud (the less dense upper part of it). The noise, or outliers, are affecting the outcome of the ICP algorithm as it tries to match every point in the source cloud to the target cloud. This causes the point cloud to not be exactly placed against the target cloud and is instead pushed away from it. In Figure 3.5, this would mean that the cloud is being placed slightly below the target cloud because of the outliers floating on top. This can be handled by removing outliers again and matching with ICP, which is described more in Section 4.1.4.

All sources mentioned in Section 2.1.1 can affect the point cloud in the data set, but to know which ones are the most dominant ones and how they affect the point cloud can be difficult to determine. The measurement noise is usually modeled as sensor white noise but it is not so dominant since the LIDAR has an accuracy of ±3 cm.

3.2 Point Cloud Merge

This is the first part of the *pre-processing* step, where the goal is to create a source cloud to be used in the ICP algorithm. The source cloud must be filled with a suitable amount of information for the ICP algorithm to be able to find a position in the target cloud that matches.

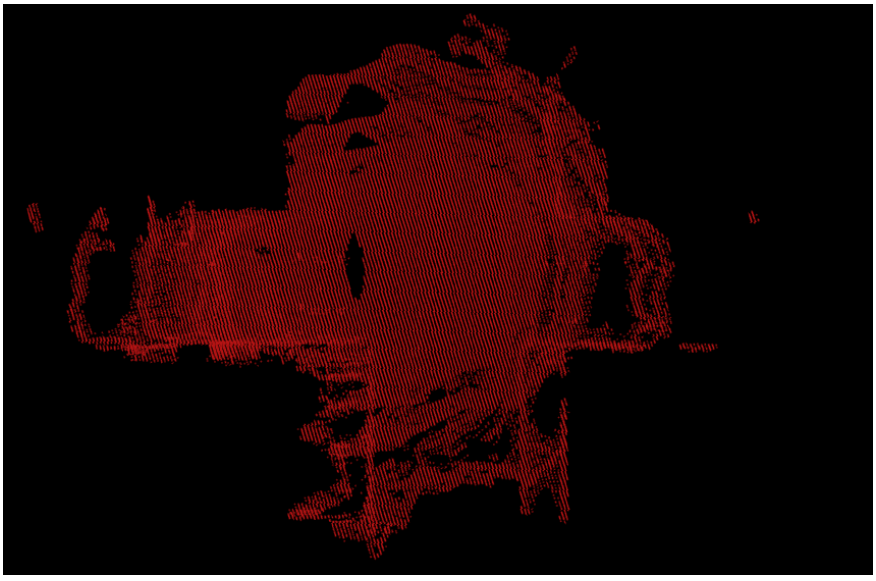


Figure 3.6: *A merged cloud at an intersection seen from above, that is containing 120 individual scans.*

To construct the source cloud, multiple scans of LIDAR sensor data taken over a period of time are merged. If, as in this case, the LIDAR sensor is attached to a moving vehicle, the vehicles translation and rotation between each scan must be taken into account.

When a scan from the LIDAR sensor is received, this individual point cloud is transformed to the current estimated global position of the vehicle which is calculated by the EKF. Each point cloud that is transformed, is also merged with the previous. This is iterated until a desired number of clouds have been merged. A source of error that may occur is that the position estimated by the EKF can drift over time, which is because the EKF is solely based on the odometry measurements while the clouds are being merged, as explained in Section 5.1. This can cause the later added individual point clouds to get an inaccurate position that leads to a slightly deformed merged point cloud. This noise is not modelled in the system and is therefore reflected in the covariance estimate. Figure 3.6 shows an example of a merged cloud containing 120 scans that form an intersection. This amount of scans is used every time a new merged point cloud is built and is roughly estimated based on three aspects; how informative the cloud becomes, the aforementioned error due to EKF drift, and computational cost. Having a high amount of scans increases the error and the computational cost. A low amount would risk not having enough information for the ICP algorithm to find a match, although the computational cost and error would decrease.

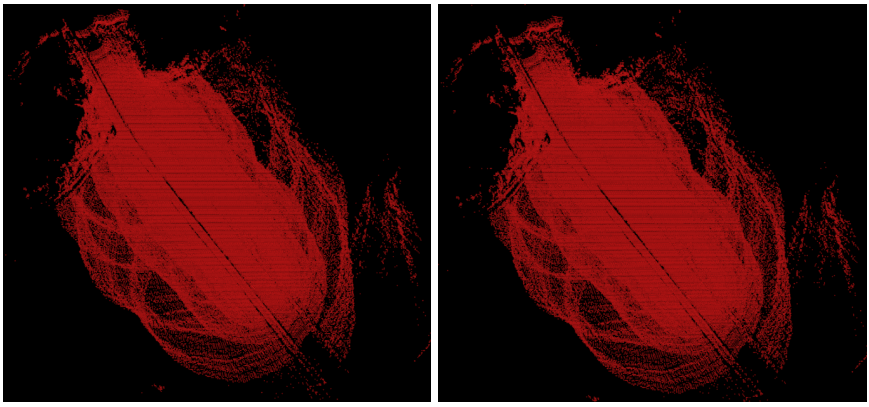
3.3 Point Cloud Filtering

This is the second part of the *pre-processing* step. The merged source cloud described in the previous section contains noise and a large amount of points. To reduce the noise and get a lower computational cost, the source cloud is downsampled and outliers are removed. Although, it becomes a trade-off as downsampling a point cloud lowers the computational cost but increases the noise with the approximations being made, while removal of outliers increase the computational cost but reduces the noise present. The noise from downsampling and the outliers are not modelled in the system and are therefore reflected in the covariance estimate. As mentioned in Section 1.3, previous work done on point cloud filtering is found in [15], and the methods they use are successful and very common. Therefore, those methods are chosen to be implemented in this system, and are presented below.

The downsampling method used in the system is *voxel grid sampling*, which is the most common method in PCL for downsampling and is chosen based on its simplicity and low computational cost. The size of the voxels are set to $0.06 \times 0.06 \times 0.06\ m^3$. More information in the cloud would be removed if the voxels were larger, while less points would be removed and the computational cost would increase if the voxels were smaller. When creating the merged source clouds, multiple scans are overlapped and many LIDAR points end up in the same position. Therefore, even if the voxel size is relatively small, a large amount of points

are removed anyways. The chosen voxel grid size retains the important information in the cloud, Figure 3.7 shows a source cloud before and after voxel grid sampling is applied. The two clouds look almost identical even though about 30 % of the points have been removed. Using voxel grid in the system reduces the computational cost by an average of 9 seconds, compared to without.

With this voxel grid size, the target cloud is still more sparse than the source cloud. If the same point density is desired, the voxel size would have to be about $0.4 \times 0.4 \times 0.4 \ m^3$, but since the ICP algorithm allows multiple points in the source cloud to be minimized towards the same point in the target cloud, it is not necessary to remove that large amount of information from the source cloud.



(a) *Before.* ***(b)*** *After.*

Figure 3.7: *The effect of voxel grid sampling. Approximately 30 % of the point cloud is removed.*

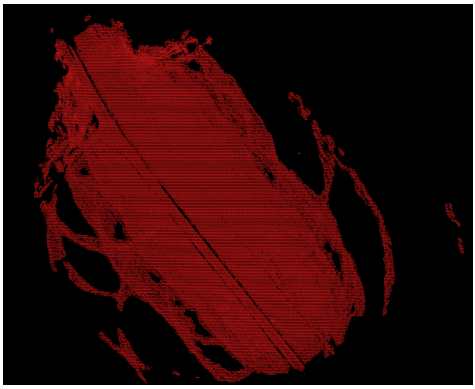


Figure 3.8: *After outliers are removed. The cloud before is shown in 3.7.*

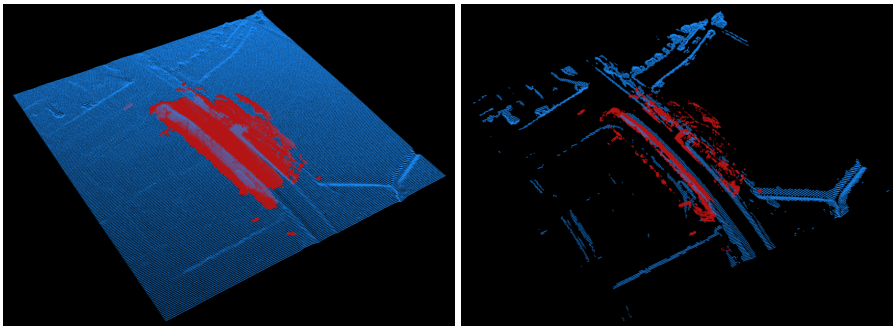
The outlier removal method used is *statistical outlier removal*, which is also chosen

based on its simplicity. It only requires one parameter to be set, which is the number of neighbours to take into account. This is set to 150 which is shown trough tests to reduce the outliers, where a higher value would remove more and possibly important points in the cloud and having a low value would keep too many unnecessary points such as noise from different sources. A result of the filtering is shown in Figure 3.8. The STD multiplier can be changed as well, but it is kept at its default value 1. Using this outlier method has an average computational time of 1.7 seconds, but at the same time it removes points which reduces the computational time of the system. It is impossible to remove all noise from a point cloud obtained from a LIDAR sensor, instead, it should be reflected as uncertainty in the covariance estimate.

3.4 Point Cloud Subset

The last part of the *pre-processing* step is to extract a subset of points from the merged and filtered source cloud, as well as from the target cloud. How and why these subsets are extracted are described in this section.

As mentioned in Section 2.2.2, the most common source of error is wrong convergence as the algorithm gets stuck in a local minimum. When performing ICP for localization, the source cloud corresponds to only a small part of the global target map. If the source cloud is given a starting position in the map that is far away from the correct position, the ICP algorithm will minimize the distances to the target points that are closest and is likely to get stuck in a local minimum. The preferred way would be if the algorithm could move the source cloud towards the global minimum, ignoring the incorrect local minimum on the way.



(a) *Whole clouds.* ***(b)*** *Subset clouds.*

Figure 3.9: *Target cloud in blue and merged source cloud in red. Two scenarios are compared,* ***(a)*** *shows the whole clouds, while* ***(b)*** *shows the same clouds but where only points with inclined surface normal directions have been chosen. In both images, the source cloud is correctly positioned.*

The number of local minimum can be reduced by choosing a clustered subset of points when creating the source and target cloud. This excludes some incorrect areas where the source cloud otherwise could converge to, and therefore allows the cloud to relocate itself easier during the ICP algorithm due to the empty spots in the target cloud. The points extracted to the subset must be informative and maintain the similarity of the two clouds, otherwise ICP becomes under-constrained (see Section 2.2.2) and a correct match is not possible. A way to choose a subset of points, that meets the previous mentioned requirements, is through sampling based on surface normals. The first step is therefore to construct surface normals as described in [36], which gives each point a direction in the form of a vector. Then, points whose normals have high inclination are added to the subset. This inclination limit is chosen based on the size of the height curves in the area. The limit can be higher in areas with a lot of height curves, than in less informative flat areas. This is because the goal of this method is to exclude local minimums, which is only achieved if a lot of points are removed. The data used in this system is collected from relatively flat areas and therefore the minimum inclination limit is set to 6°. The subset clouds would be filled with different information if the limit is increased or decreased with only a couple of degrees. Therefore, a future work would be to create a function that decides this minimum inclination automatically, that for example makes sure a certain percentage of all points are removed and checks that the remaining points are informative.

The result of this method is a new cloud with less points and that are, in best case scenario, more clustered, which often is the case when tested on the data used in this thesis. Due to the reduction of points, the computational cost is lower as well. Figure 3.9 compares a whole cloud with a subset cloud. Extracting these subset source and target clouds is the third and last step of the *pre-processing* step.

4

Global Position Estimation

This chapter presents an implementation of the ICP algorithm that matches a LIDAR point cloud to an elevation map, and is the second step in the system, seen in Figure 1.1. Figure 4.1 shows the corresponding input and output of this step. The first part of this step is the ICP algorithm. In this master's thesis, an own implementation of the algorithm is made where ICP is performed two times, further explained in the following sections. Three ICP extensions are presented as well, *outlier removal after ICP*, *stochastic initial positions* and *unscented transform based*. The second part of this step is uncertainty (covariance) estimation of ICP. The estimation is computed in 3D since the point clouds are in 3D, but only x and y coordinates along with the rotation around z is used to estimate the 2D global position.

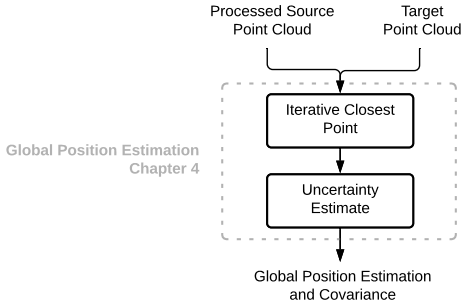


Figure 4.1: *The global position estimation step is divided into these two parts. The input is the processed source point cloud from the previous pre-processing step, as well as the target point cloud. The output from this step is a global position estimate and its associated covariance that in the next step (Chapter 5) is used as an observation in the EKF.*

4.1 Iterative Closest Point

The developed implementation of the ICP algorithm along with a method to evaluate its result are described in this section, which represents the first part in the *global position estimation* step. Some factors that affect the convergence are described as well. The green cloud in Figure 4.2 shows how the source cloud is initially positioned with the position estimated by the EKF, as explained in Section 3.2. The ICP algorithm aligns the source cloud to the global point cloud map seen in blue, and thereby adjusts it to the global coordinate system.

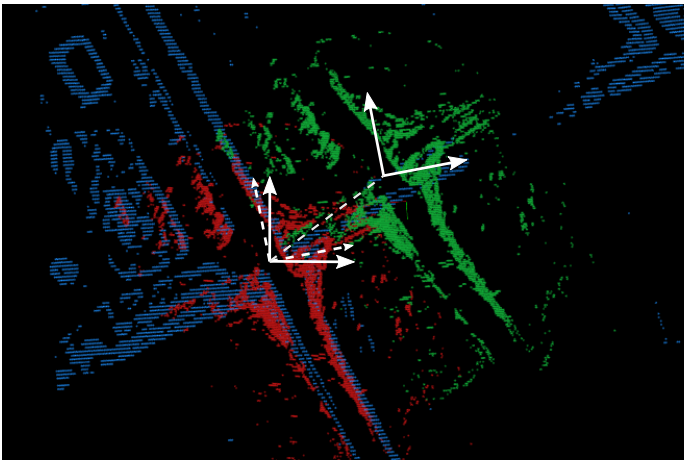


Figure 4.2: *Green source cloud is before ICP algorithm and red is after, blue is the target cloud. Subset clouds are used for better illustration. The clouds are seen from above in x and y coordinates.*

4.1.1 Implementation

The input data used in the ICP algorithm are the outputs from the *pre-processing* step, seen in Figure 1.1. In other words, the pre-processed source point cloud created from the LIDAR sensor mounted on the vehicle, and the target point cloud which in this system is equivalent to an elevation map of the area. As described in the previous chapter, subset of points are extracted. Both these subset clouds, as well as the whole clouds are used in this step.

In the previous *pre-processing* step, the source cloud is given an initial translation to the estimated position of the vehicle, described in Section 3.2, to give the ICP algorithm better conditions and decrease wrong convergence. The algorithm is applied in two steps in the developed system; first with the subset clouds and then with the whole clouds, as described below. There are some extensions to the algorithm as well, both found in Section 4.1.4.

With Selected Subset

First, an ICP algorithm (from PCL) is applied where the subset source and target clouds containing the points with inclined surface normals are used as input. This helps avoid wrong convergence as explained in Section 3.4. The point-to-plane variant of ICP is chosen since it has been shown in [32] to have better convergence. After finding the transformation matrix that minimizes the cost function, it is applied to the source cloud which in an ideal case moves it close to the global minimum.

Figure 4.3 shows the initial placement of the subset source point cloud in green and the result after ICP is performed in red. The two visible lines in the source cloud are road ditches that got extracted to the subset cloud since they have an inclined surface. They are important environment features since they provide sideways positioning.

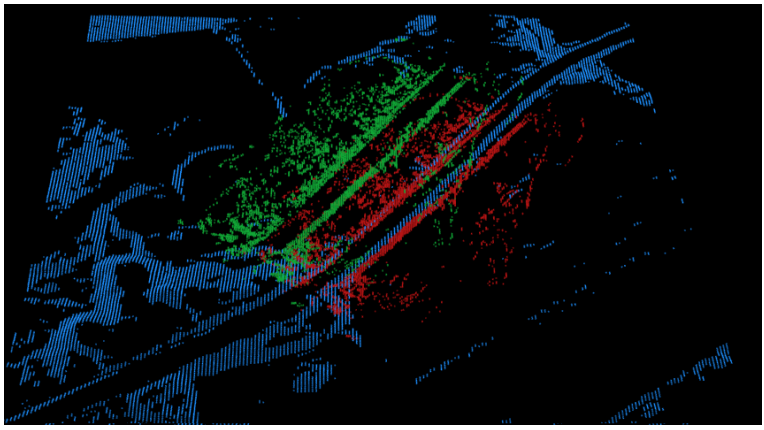


Figure 4.3: *The green cloud is the subset source cloud before the ICP algorithm is performed. The red is after ICP, and the blue is the fixed target subset cloud.*

With All Points

After the source cloud has been relocated with the optimal transformation matrix calculated by the ICP algorithm using the subset source and target clouds, seen in Figure 4.3, the algorithm is applied again but with the whole clouds as input to take all points into account. This adjusts the cloud in place and does not give a very visible transformation as the ICP with subset points results in, since the cloud is already translated to a local or global minimum. This step is mostly needed so that the ICP evaluation method called fitness score, explained in the next section, takes all points into account and becomes as accurate as possible, which makes it easier to decide how correct the match is. The output from the implemented ICP algorithm is the difference in global x, y and θ between the initial position of source cloud, and the position where it is aligned to.

4.1.2 Fitness Score

After performing an ICP algorithm, the match is evaluated by investigating the relative position between the source and target point clouds. In *Point Cloud Library*, described in Section 2.7, there is a function called *FitnessScore* which can be used for this purpose. The fitness score is found by calculating and summing up the Euclidean distances between each point in the source cloud, to their closest point in the target cloud. This score gives an idea of how close the two clouds are located, and then also how accurate the match is.

$$d(s, o) = \frac{1}{N_s} \sum_{i}^{N_s} |s_i - o_{i^*}|, \tag{4.1}$$

where s_i is the i^{th} point of the source cloud with its closest point o_{i^*} in the target cloud and N_s is the total amount of points. This evaluation method is used in the global localization system to determine if a match, i.e. a global position estimate, found by ICP should be accepted or discarded.

There are some drawbacks with the fitness score. Clouds that look like they have been correctly matched when visualized can be given a bad score due to some poorly positioned points in the source cloud. The noise from unmapped objects mentioned in Section 3.1.2 is most likely one of the factors contributing to the faulty score. The opposite issue, where a cloud that is not matched well visually but given a good score, could also happen because the ICP found a local minimum instead.

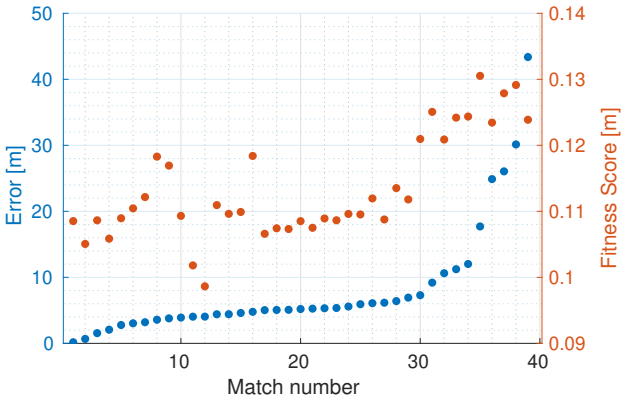


Figure 4.4: *The fitness score and error for all studied point clouds. The error is the euclidean distance from where the cloud should be positioned to where ICP found a match instead. Blue dots are error and corresponds to the left y-axis, red dots are fitness score and corresponds to the right y-axis.*

Since the data used to create the source and target clouds are not completely similar, a zero fitness score is impossible to achieve. A low fitness score only indicates

that the algorithm has found a good local minimum, it does not have to be the global minimum. By studying a large amount of matches, a fitness score limit is chosen. If a match has a score above the limit, it is seen as a bad match and is discarded. Figure 4.4 shows the fitness score along with the error to the real position of the studied matches and they are sorted by increased real error. Overall, the fitness score is increasing with an increased real error, but the drawbacks of fitness score mentioned previously is also seen in the figure, as some matches with low error have gotten a high fitness score and vice versa. By selecting the fitness limit to 0.12, all the matches with low error will be accepted, and the worst matches are discarded. This limit is only valid on the data set in this thesis and should be reconsidered on other sets of data. Note that the outlier removal method described further ahead in the thesis in Section 4.1.4 is used when these values are produced, which has lowered the fitness score.

A disclaimer is that the fitness score is only compared against x and y error in this test, and not rotational error. However, if a source cloud is wrongly rotated, is is most likely that it has drifted away from the correct x and y position as well since it will not fit into that position, unless the area is very flat. Therefore, if a match has got a high fitness score and low x and y error, it is taken into account that it can depend on wrong rotation of the source cloud.

4.1.3 Convergence factors

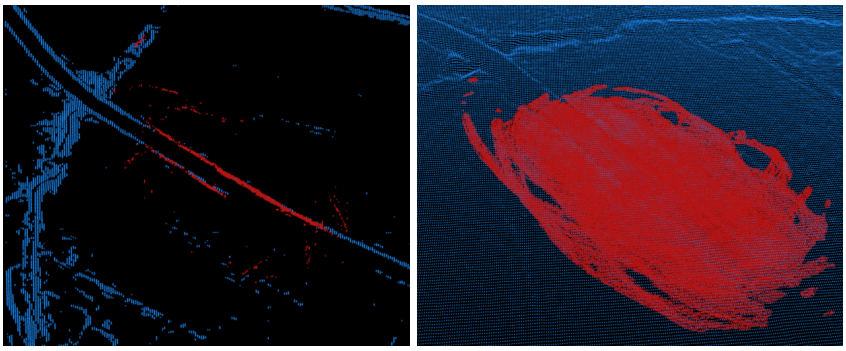
Except for how far away the source clouds are initially positioned, there are two main factors that affect the ICP algorithm's convergence; how informative the area that the point clouds represent is and how similar the source and target cloud used as input are. The purpose of this section is to evaluate the implemented ICP algorithm, which is the first part in the *global position estimation* step.

Informative Areas

A factor that affects the accuracy of the ICP results is how informative the area is. Since the target and source cloud only contains ground points, the only important information is height curves. In completely flat areas, a correct match through ICP is impossible since it becomes an under-constrained situation. The optimal condition is when the source and target cloud contains curves that together face all directions, which gives full constraint. An important environment feature is road ditches, which help provide the algorithm with sideways (of the vehicle) positioning since they often are located on each side of the roads.

Figure 4.5 shows a match where the vehicle is driving along a field with ditches on each side. **(a)** shows the subset clouds, seen from above, containing the inclined height curves. **(b)** shows the whole clouds in a slightly horizontal view. A road is seen in the diagonal in both figures. As mentioned, the ICP algorithm is first performed with the subset clouds, and then again with the whole cloud to take all points into account. It is clear in this situation that the algorithm can relocate the source cloud to the road sideways, but it does not have enough infor-

mation to position it along the road since there are no curves that gives guidance in that direction. The match is 8.5 meters away from the ground truth position, even though it is seemingly good positioned on the road sideways.



(a) *Subset clouds.* ***(b)*** *Whole clouds.*

Figure 4.5: *An example of an area, which is informative to the side but is uncertain along the road. Target cloud is in blue and merged source cloud is in red.*

To evaluate the performance of the subset clouds in this kind of area, another test is made. Instead of both using the subset and whole clouds, only the whole clouds seen in Figure 4.5 **(b)** are used as input. The result is that the source cloud gets stuck near the initial position next to the road with an error of 10.5 meters, which is worse than in the case where both subset and whole clouds are used. This is because the gaps in the subset clouds provides important information which is excluding incorrect positions, in other words local minimums. In summary, the subset clouds can help provide better estimates, even in less informative areas as in this case.

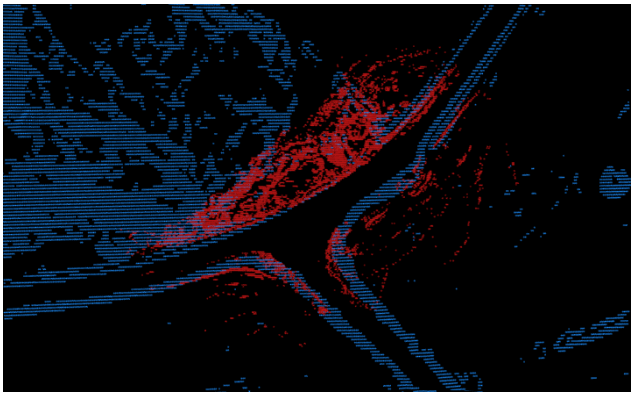


Figure 4.6: *Subset clouds containing the inclined surfaces, after ICP algorithm. The image shows a curved road and intersection. Target cloud is in blue and source cloud is in red.*

Figure 4.6 shows another match. Unlike the source cloud in Figure 4.5, this cloud contains more information because of the curved road and intersection that were scanned by the LIDAR. The match is 1.7 meters away from ground truth. The amount of information contained in a point cloud is preferred to be reflected in the covariance estimate.

Similarity of Input Clouds

It is important that the source and target clouds used as input to ICP is similar for the algorithm to be able to find an accurate match. As mentioned in Section 3.1, all the houses, trees and other objects have been removed from the two input clouds. A source of error that occurs when the elevation map is processed, mentioned in [18], is that some forest areas can be mistaken for hills. It is also difficult to approximate the surface underneath the trees, which leads to an inaccurate target map in forest environments. The same reasoning also applies to some extent in urban environments, but the surface underneath is easier to approximate since it is most often the same as the area around it.

The source clouds only contain ground points as well, as explained in Section 3.1. The ground point extraction algorithm works well in environments where there are clear objects, as houses, but some unwanted points often get mistaken for ground when there is a lot of vegetation. These points are removed as best as possible with point cloud filters, for example statistical outlier removal mentioned in Section 2.1.2.

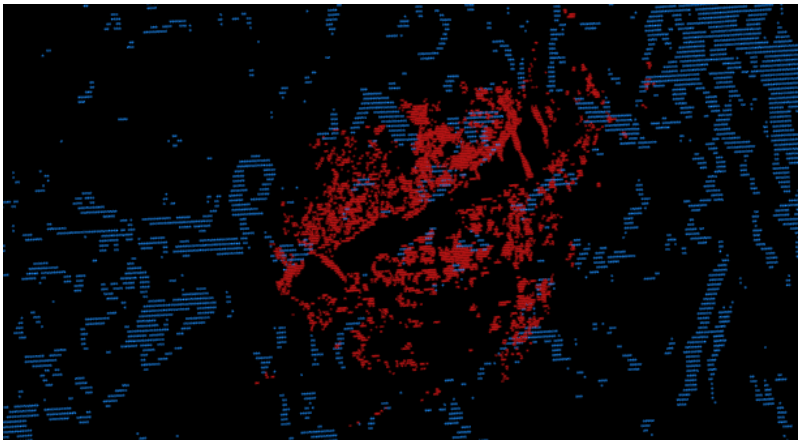


Figure 4.7: *Subset point clouds containing points with inclined surfaces, in forest route. Red is source cloud and blue is target cloud. Seen from above.*

Since the ground points are extracted in different ways in the source and target clouds, the similarity will decrease in areas that have a lot of objects. Therefore, the uncertainty in forest environments is large, leading to less accurate convergence. Figure 4.7 shows a match in a forest environment, where the source cloud

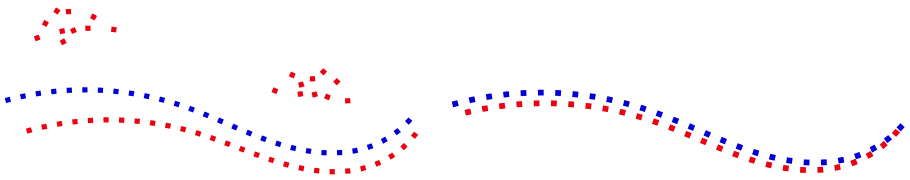
is positioned almost correctly. The two clouds have different height curves even though they represent the same position. The reason why ICP manages to position the source cloud on the road even though the height curves of the two clouds are different, is because of the gaps in the subset clouds where the road is. Since the algorithm minimizes the distances between each point in the source cloud to the closest point in the target cloud, all points strives not to be placed above the gap. The result of this is that the gap in the source subset cloud often is positioned above the gap in the target subset cloud.

4.1.4 Extensions of ICP

The implemented ICP can be altered with extensions to possibly improve the result. This section presents three different extensions.

Outlier Removal after ICP

As discussed in the previous section, the similarity of the source and target cloud is a convergence factor. A method that increases the similarity, and thereby increases correct convergence, is proposed in this section. This is seen as an extension of the implemented ICP algorithm.



(a) Before. *(b) After.*

Figure 4.8: *2D example of how removal of more outliers can improve the result. Red cloud is the source cloud and blue is the target, seen from the side.*

In the *pre-processing* step of this global localization system, outliers are removed from the source cloud. Since it is difficult to know which points that are outliers, a lot of them will remain in the cloud. When performing ICP as described in Section 4.1.1, the algorithm tries to minimize the distance from all points in the source cloud to the points in the target cloud, the outliers included. If the source cloud has outliers floating on the top part of it, as for example vegetation seen in Figure 3.5, the real ground points in the source cloud will be given a lower position than the ground points in the target cloud. This is because the algorithm tries to minimize the distance from the outliers to the target cloud as well. An illustration of this is seen in Figure 4.8 **(a)**.

The proposed method suggests that after the ICP algorithm is performed and the source cloud is positioned in a global or local minimum, the distance to the

closest point in the target cloud is found for each point in the source cloud. If this distance is larger than a certain value d_{max}, that point is seen as an outlier and is removed from the source cloud. After all outliers are removed, another round of the ICP algorithm is performed on the whole remaining source cloud. Since this is only an adjustment step, only a few iterations are needed in ICP. An illustration of the new aligned clouds, without outliers are seen in Figure 4.8 **(b)**. This also gives a more fair fitness score, since less outliers are added to the mean. In the system, d_{max} is set to a distance that is mostly meant to remove outliers in the form of vegetation, therefore, 0.25 m is chosen. Figure 4.9 shows an example of this on real data, in an evironment with a lot of vegetation.

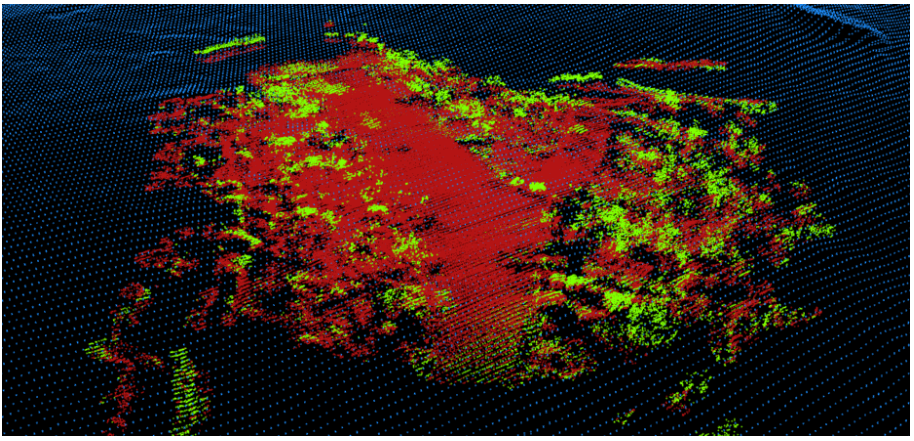


Figure 4.9: *Outlier removal after ICP, in a forest environment. The green points are more than 0.25 m away from the nearest point in the blue target cloud, and will be removed. The red points remain in the source cloud.*

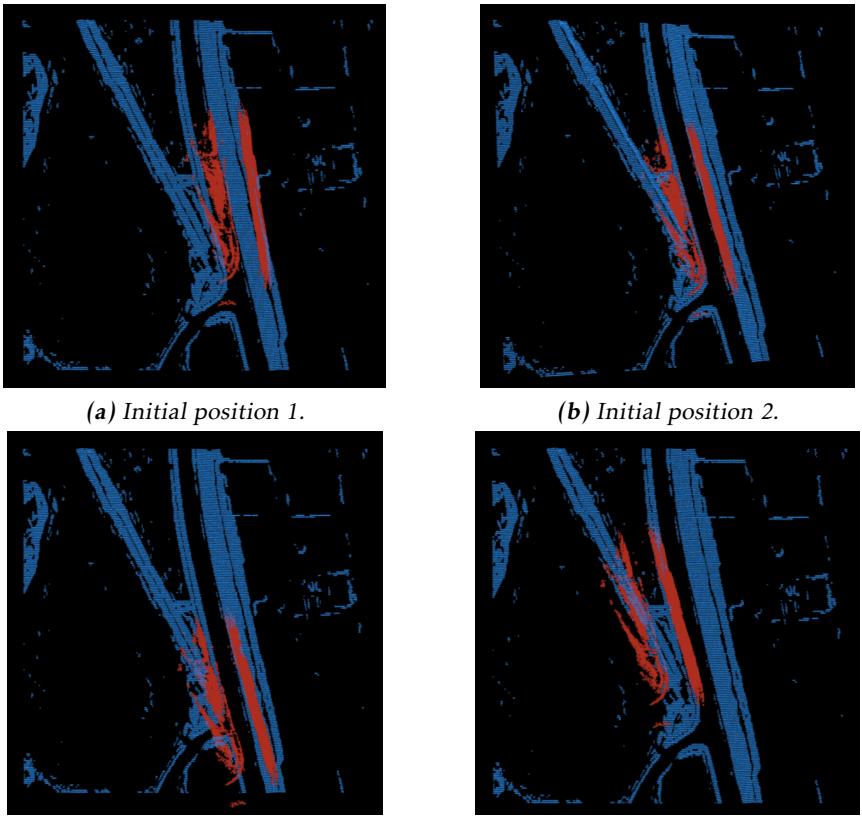
Stochastic Initial Position

To investigate the possibility to increase the chance of ICP converging to the global minimum, a method with this purpose is evaluated in this section.

An extension of the implemented ICP algorithm is inspired by the Stochastic ICP described in [25]. As the article describes, a random Gaussian noise translation is added to the source cloud before each iteration of the ICP algorithm. If the algorithm chooses about the same position more than once, it could indicate that it has found the global minimum and the translation noise added before the next iteration is reduced. This can cause the source cloud to be relocated far away from the initial position if it does not manage to find the same position more than once, which happens when tried in this system.

An altered method of the Stochastic ICP is developed in this master's thesis which puts more trust in the position estimated by the EKF, described in Chapter 5. The

method adds different random translations before the first iteration to give the source cloud multiple initial positions. The positions are chosen randomly from normal distribution with zero mean, in other words the position estimated by the EKF, and a variance of 10 meters to get a proper distribution. The computed covariance could be used instead, but as seen later in Section 4.2.2, the covariance becomes very optimistic and therefore a larger value is chosen to be used instead. The ICP algorithm is performed from each of the randomly chosen positions, and also from the EKF's estimated position without translation added, for comparison. If too many initial positions are chosen, the computational cost gets high. Therefore, only four initial positions are tried in this evaluation for exemplification. If the EKF's estimated position is near a wrong local minimum, this method can give the source cloud a chance to avoid wrong convergence when an initial translation is applied. The best result out of the performed ICP algorithms is chosen with the evaluation method fitness score described in Section 4.1.2.



(a) *Initial position 1.*

(b) *Initial position 2.*

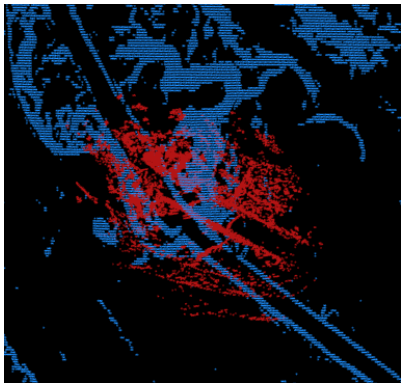
(c) *Initial position 3.*

(d) *Initial position 4.*

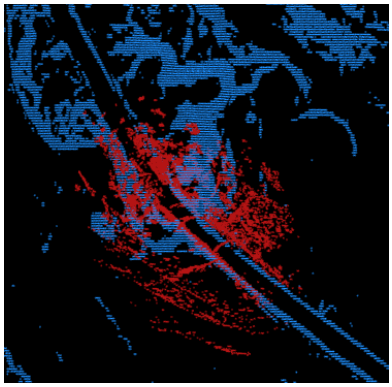
Figure 4.10: *Results of ICP with four different initial positions, where the algorithm chooses the seemingly correct match.*

Figure 4.10 shows a performance of this method where the ICP algorithm is per-

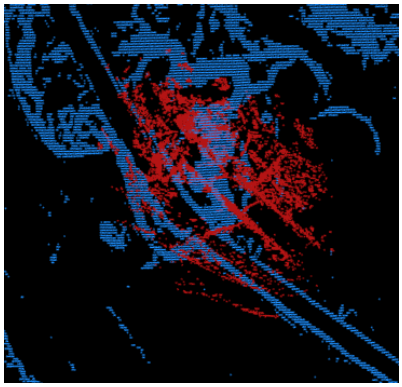
formed from four different initial positions. **(a)** is the result when the source cloud started from position estimated by the EKF, and the rest got a random translation applied. **(b)** got the best fitness score and is also seemingly most accurate. The system will register that position as a location of the vehicle. This is the ideal result of this method, since it managed to relocate itself from the local minimum that it otherwise would have converged to.



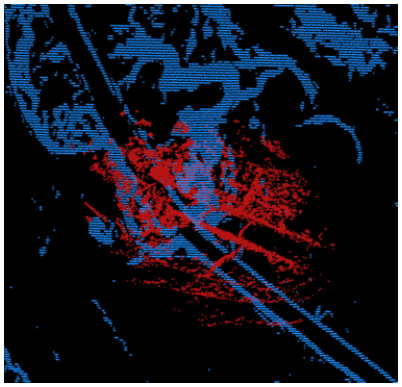
(a) Initial position 1.



(b) Initial position 2.



(c) Initial position 3.



(d) Initial position 4.

Figure 4.11: *Results of ICP with four different initial positions, where the algorithm chooses the seemingly incorrect match.*

Figure 4.11 shows another result. **(a)** is given the initial position estimated by the EKF, without translation added. The match that is seemingly nearest the global minimum is **(b)**, but the algorithm chose match **(c)** since it has the lowest fitness score. The problematic part of this method is therefore to evaluate the matches correctly all the time. Therefore, this method is only evaluated here and is not further used in the designed localization system. However, this evaluation highlights the risk of using fitness score to determine the accuracy of ICP results. If

an evaluation method with higher reliability than fitness score was found, this method could be improved. This evaluation also shows how easy the ICP algorithm finds local minimums, which is the main drawback of ICP.

UT variation

Another extension is to use the ICP based on the unscented transform, as described in Section 2.4. This method is chosen because it has the possibility to compute a covariance that would consider all the sources of error described in Section 2.2.2, and should therefore be better than the other covariance estimation methods.

The method is based on an amount, $2n+1$, of sigma points being calculated where n depends on the size of the state vector, which is 6 in this case. The sigma points are, unlike *Stochastic Initial Position*, chosen in a deterministic manner given a mean and covariance. The point cloud is transformed and ICP is performed (as described in Section 4.1.1) for each sigma point, giving different transformation matrices and the best matrix, based on the fitness score, is picked. This gives the same issue as for *Stochastic Initial Position* where the fitness score could choose an incorrect match. However, since this method chooses its initial positions in a deterministic manner and that it allows for covariance computations, it is chosen to be further evaluated in this global localization system. The covariance that belongs to this ICP with UT variation is evaluated in the next section. All the parameters that have been defined in Section 2.4 are chosen to be the same as the ones in [6], except for κ which is discussed more below in Section 4.2.2.

The computational cost of this method increases by a lot since the ICP has to be performed for all 13 sigma points. The ICP that is done on the whole point cloud is the source for the majority of the cost because of the high amount of points that the ICP has to go through. Therefore, this method cannot be used in real time.

4.2 Uncertainty Estimate

The second and last part of the *global position estimation* step is to find the uncertainty of the result provided by ICP, which is done by estimating the covariance. Different covariance methods and an evaluation of these methods are found in this section.

4.2.1 Covariance

The covariance is used in the EKF to weigh the position estimates given by ICP, described further in Chapter 5. The covariance methods that are tested are the UT based covariance estimate described in Section 2.4.3 (which requires the ICP with UT variation), covariance with correspondences, and covariance with Hessian where the last two are described in Section 2.3. The covariance is estimated

in 3D since the point clouds are in 3D, but the elements of x, y and θ (rotation around the z axis) in the covariance matrix are selected and placed in a 3×3 matrix instead, denoted R_k in Section 2.6.1. This is because the EKF only estimates 2D positions, since the odometry measurements from FOI's existing system, i.e. the local position estimates, are in 2D. The covariance is estimated after the ICP algorithm, when the source cloud has been aligned to the global coordinate system, seen in Figure 4.2.

The different covariance estimation methods are implemented and tested separately to be able to decide how well they capture the true variance. This is evaluated with the method called *NEES*, described in Section 2.5, and the evaluation is found below.

4.2.2 Evaluation of Covariance Methods

To evaluate the performance of the different covariance estimation methods mentioned above, a test is performed. The covariance matrix is estimated for approximately 20 source clouds (aligned with ICP) for each method to get an overall result, and the *NEES*-value is computed for each of these covariance matrices. The mean of the *NEES*-values for each method is used to compare its accuracy. The 20 source clouds used in the test are produced as in the *pre-process* step. They are from different environments such as field, forest, and urban, which is preferred since it gives a more general result. More information about the exact routes where the data is collected, is found in Chapter 6.

Table 4.1: *The mean NEES results for the different covariance estimation methods.*

Method	NEES
UT	$7.26 \cdot 10^3$
Hessian	$8.01 \cdot 10^3$
Correspondences	$2.15 \cdot 10^{12}$

The transformation error, $\tilde{x}$ in *NEES*, is computed by subtracting the position where the source cloud is aligned by ICP, from the ground truth position given by the GNSS. In other words the difference between estimated and real position. The same is done for the angle θ, which is rotation around z. The GNSS used as ground truth does not compute the real rotation of the vehicle. Therefore, it is estimated as the slope of the path provided by the GNSS (the slope in the ground truth x and y position). To take into account is that the GNSS used as ground truth has an accuracy of < 3 meters, therefore, the *NEES* results cannot be fully trusted. As mentioned in Section 2.5, for 2D cases the best *NEES*-value is three. Table 4.1 shows the results of *NEES* for all the different covariance estimation methods tested. The results show that all estimation methods are very optimistic, where the UT based method is the least optimistic and the method based on correspondences is the most. Optimistic covariance means that the es-

timated uncertainty is less than it should be. Therefore, it should be taken into account that the uncertainty is larger than it appears.

The common factor why all methods are optimistic is that there are sources of error, specific to this system, that are not considered. These sources could be outliers, deformation in the merged cloud caused by drift in the EKF (mentioned in Section 3.2), or dissimilarities in the source and target cloud caused by removed objects. Even though the UT based method takes the most sources of error into account, it is not enough in this system. Further, that the covariance method using Hessian is optimistic is expected from [4], due to the reasons stated in Section 2.3.2, which is that the Gaussian noise assumed in the error is negligible for a high amount of points and it only considers *sensor noise*. Another reason could be that the assumption of the cost function being linearized if around the convergence point is not an accurate representation. The covariance with correspondences method is the highest, which is because it, as the method based on Hessian, only takes *sensor noise* into consideration. It also uses a method that is known to be optimistic, which is stated in Section 2.3.3, as well as having the same assumption of the cost function as the method based on Hessian.

As mentioned in the introduction of this thesis, two different system variations are created and are further evaluated in the experiment presented in Chapter 6. The conclusion of this covariance evaluation is that the UT based covariance, used together with the UT based ICP, gets the best *NEES*-value and is therefore used as one of the system variations. The other system variation uses the estimate from the covariance with Hessian, since it is much less optimistic than covariance with correspondences. To have in mind is that none of the covariance estimates takes all possible sources of error into account and is therefore not fully realistic.

A future work would be to see if the results of the Hessian based method and UT based method could be improved by increasing the white noise design parameter σ, as seen in the theory chapter. In this evaluation, σ for the Hessian based method was set to 5 m to compensate for the lack of error sources that the method takes into account. For the UT based method σ was set to 0.1 m, which is the value used in the method's reference article. It was not set to a higher value since it was assumed that this method takes enough sources of error into account, which it clearly did not. Unfortunately, this design parameter is not available for the correspondence based method, which is why its *NEES*-value is still very high compared to the others. Another parameter that has also been altered in the UT based method is κ which is changed to 144. A lower value gives a higher covariance estimate (with a more accurate *NEES*-value), but this makes the EKF trust the position estimate less, while a higher value gives the opposite effect. The drawback of having a too high or low value on κ is that the system can become unstable.

5

Global Position Filtering

In this chapter, global position filtering obtained by an EKF is explained. This is the last step of the system, seen in Figure 1.1. Figure 5.1 shows the corresponding inputs and output of this step. The inputs are the global position estimate and covariance from the previous *global position estimation* step, also position estimation and covariance from the odometry computations in FOI's existing system. These inputs are weighted by an EKF that provides a merged global filtered position for each time step, which is the output of this step.

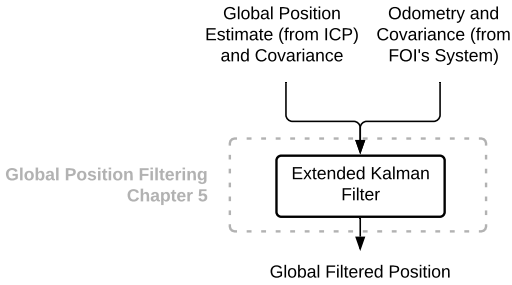


Figure 5.1: *The global position filtering step, with its inputs and output. The inputs are the estimations of a global position and covariance, as well as the odometry and its covariance from FOI's existing system. The output is a filtered position which is sent back to the pre-processing step, thus closing the loop.*

5.1 Extended Kalman Filter

As mentioned in Section 1.3, an EKF is used in the system to weigh the global position estimates, provided by ICP in the previous step, together with odometry estimates. In between the global position estimates (provided in average each 16.5 seconds), the EKF only relies on odometry (given in average each 0.05 seconds). Since odometry drifts, the position estimates from ICP is used to correct this drift as best as possible. Figure 5.2 shows an illustration of how the EKF corrects its position when a global position estimate is received.

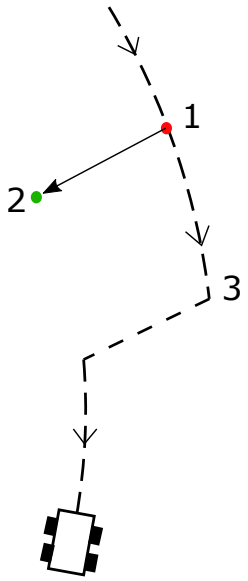
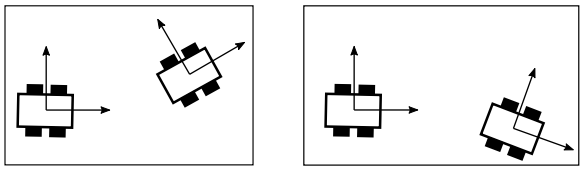


Figure 5.2: *Illustration of how the EKF corrects its filtered position when a global position estimate from ICP is received. 1 represents the position where the source cloud was initially positioned, 2 is the position on the reference map where the source cloud is aligned to. 3 is the position of the vehicle when the ICP computations are done (the vehicle is moving during these computations), which is when the EKF corrects its position.*

State Model

The odometry information that is used from FOI's system is the difference between the current position and rotation of the vehicle and the previous, which is expressed in the coordinate system of the vehicle. This is illustrated in Figure 5.3. To be able to use this information in this global localization system, it needs to be expressed in global coordinates, as seen in Figure 5.4.



(a) *Vehicle position for k-1 relative to k-2.*

(b) *Vehicle position for k relative to k-1.*

Figure 5.3: *Difference in vehicle position shown for two time steps. This, along with the covariance for each step, is the information provided by FOI's odometry system.*

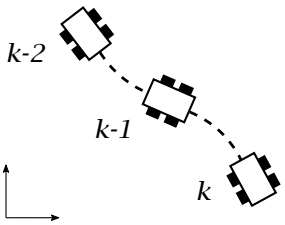


Figure 5.4: *The resulting odometry path from the state model, when a starting pose in $k-2$ is provided. This is the position differences seen in Figure 5.3, which has been transferred to the global coordinate system and combined.*

To transfer the relative positions in Figure 5.3 to global positions in Figure 5.4, a state model based on the odometry model found in (2.49)-(2.51) is used. The state model becomes

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + \delta_{trans_k} \cos(\theta_{k-1} + \delta_{rot1_k}) \\ y_{k-1} + \delta_{trans_k} \sin(\theta_{k-1} + \delta_{rot1_k}) \\ \theta_{k-1} + \delta_{rot1_k} + \delta_{rot2_k} \end{bmatrix} + w_k, \tag{5.1}$$

where δ_{trans_k}, δ_{rot1_k} and δ_{rot2_k} are assumed given input from the odometry. The odometry position differences from FOI's system are denoted $\Delta\hat{x}_k^o$, $\Delta\hat{y}_k^o$, and $\Delta\hat{\theta}_k^o$. Since they are in the coordinate frame of the vehicle, they must first be translated into relative motion parameters

$$\begin{bmatrix} \delta_{trans_k} \\ \delta_{rot1_k} \\ \delta_{rot2_k} \end{bmatrix} = \begin{bmatrix} \sqrt{(\Delta x_k^o)^2 + (\Delta y_k^o)^2} \\ \arctan2(\Delta y_k^o, \Delta x_k^o) - \theta_{k-1}^o \\ \Delta\theta_k^o - \delta_{rot1_k} \end{bmatrix}, \tag{5.2}$$

which can then be used in (5.1) to compute the global coordinates by taking the previous position and rotation into account. A more detailed illustration of how the position differences and the relative motion parameters correlate is found in Figure 2.7.

The noise w_k is assumed to be Gaussian distributed with mean zero and covariance Q_k ($w_k \sim \mathcal{N}(0, Q_k)$). The covariance matrix of $\Delta\hat{x}_k^o$ and $\Delta\hat{y}_k^o$ is given by FOI's system, denoted P_k^o. How it is computed is out of the scope of this thesis, but it is shown in a previous thesis [14] to provide successful results and is therefore used in this thesis as well. The given covariance is also in the coordinate frame of the vehicle. To be able to use it in Q_k, it must be rotated to the global coordinate frame (since Q_k is in global coordinates), which is done by rotating it by θ_k. The x and y part of Q_k, denoted Q_k^{xy}, is the left upper 2×2 corner of the 3×3 Q_k. It is obtained by $Q_k^{xy} = R^\theta P_k^o (R^\theta)^T$, where R^θ is the orthogonal 2D rotational matrix for rotation by θ_k. This transformation of covariance between different coordinate systems is motivated in [39]. Unfortunately, FOI's system only computes rotational uncertainties in quaternions, therefore, a simplification is made that the θ part of Q_k is set to 0.01 which is the error of the gyroscope in the IMU [42].

To be able to linearize the nonlinear state model around the current estimate, the Jacobian J_f containing the partial derivatives of (5.1) is needed

$$J_f = \begin{bmatrix} 1 & 0 & -\delta_{trans_{k|k-1}} \sin(\hat{\theta}_{k|k-1} + \delta_{rot1k|k-1}) \\ 0 & 1 & \delta_{trans_{k|k-1}} \cos(\hat{\theta}_{k|k-1} + \delta_{rot1k|k-1}) \\ 0 & 0 & 1 \end{bmatrix}. \tag{5.3}$$

Observation Model

The observation model is based on the output of the implemented ICP algorithm, and this output is a relative offset from the position where the source cloud was initially positioned before ICP. This can be seen as a global position if the offset is added to the initial position, i.e. the observation z_k obtained from ICP is

$$z_k = \begin{bmatrix} x^o \\ y^o \\ \theta^o \end{bmatrix} + \begin{bmatrix} \Delta x^{ICP} \\ \Delta y^{ICP} \\ \Delta\theta^{ICP} \end{bmatrix}, \tag{5.4}$$

where x^o, y^o and θ^o are the states where the source cloud is initially positioned in. ICP moves the source cloud by Δx^{ICP}, Δy^{ICP} and $\Delta\theta^{ICP}$, and by adding these differences to the initial position, the resulting global position is found. An illustration of this is seen in Figure 5.2 where x^o, y^o and θ^o represents position 1, and Δx^{ICP}, Δy^{ICP} and $\Delta\theta^{ICP}$ is the difference between position 1 and 2.

The observation model is therefore described as

$$z_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + v_k, \tag{5.5}$$

where z_k is the computed observation, i.e. the global position from ICP. The initial position of the source cloud $[x^o \ y^o \ \theta^o]^T$ is chosen as the estimate of the true position $[\hat{x}_{k|k-1} \ \hat{y}_{k|k-1} \ \hat{\theta}_{k|k-1}]^T$. The observed error v_k is seen as Gaussian distributed noise with mean zero and covariance R_k ($v_k \sim \mathcal{N}(0, R_k)$). As described in Section 4.2.2, two different system variations are created that uses different methods to estimate the covariance R_k. The first uses UT based covariance and the second uses covariance with Hessian, the theory about these methods are found in Section 2.4.3 and 2.3.2 respectively. Since the observation model is linear, the Jacobian J_h is the identity matrix (3×3).

Prediction and Correction Step

The prediction step is performed as in (2.41) and (2.42). The correction step begins with computing the Kalman gain K_k as in (2.43), which is a 3×3 matrix. It decides how much the EKF should trust the observed states, and how much it should trust the predicted states (5.1), based on their covariance estimates R_k and Q_k. If R_k is very small, i.e. the system trusts the ICP match, then K_k will go towards the identity matrix.

To compute the corrected state vector (2.44), an expression for the innovation is needed. The innovation is, as mentioned in the theory chapter, the difference between the computed observation and the estimated. Since $[x^o \ y^o \ \theta^o]^T$ is chosen as $[\hat{x}_{k|k-1} \ \hat{y}_{k|k-1} \ \hat{\theta}_{k|k-1}]^T$, the innovation can be found as

$$\begin{bmatrix} x^o \\ y^o \\ \theta^o \end{bmatrix} + \begin{bmatrix} \Delta x^{ICP} \\ \Delta y^{ICP} \\ \Delta \theta^{ICP} \end{bmatrix} - \begin{bmatrix} \hat{x}_{k|k-1} \\ \hat{y}_{k|k-1} \\ \hat{\theta}_{k|k-1} \end{bmatrix} = \begin{bmatrix} \Delta x^{ICP} \\ \Delta y^{ICP} \\ \Delta \theta^{ICP} \end{bmatrix}. \tag{5.6}$$

With this, the corrected state vector equation can be expressed as

$$\begin{bmatrix} \hat{x}_{k|k} \\ \hat{y}_{k|k} \\ \hat{\theta}_{k|k} \end{bmatrix} = \begin{bmatrix} \hat{x}_{k|k-1} \\ \hat{y}_{k|k-1} \\ \hat{\theta}_{k|k-1} \end{bmatrix} + K_k \begin{bmatrix} \Delta x^{ICP} \\ \Delta y^{ICP} \\ \Delta \theta^{ICP} \end{bmatrix}. \tag{5.7}$$

Since the vehicle travels during the ICP computations, an assumption is made that $[\Delta x^{ICP} \ \Delta y^{ICP} \ \Delta \theta^{ICP}]^T$ is still valid after the computations are done, which takes about 8 seconds. It is thus assumed that the odometry drifts very little during this time. If refereed to in Figure 5.2, the difference between position 1 and 2 is used to update position 3. A larger study how this assumption affects the system has not been made due to time limitations. The correct way to update a current state with older measurements is to use an out-of-sequence update step, as described in [1].

When the EKF has provided a global position (which is each time step k, between each output from ICP it only depends on the odometry), it is sent to the *pre-processing* step, which closes the system loop. There, it is used to merge a new source cloud that after processing is sent to the *global position estimation* step where the ICP algorithm is performed. There, a global position is estimated and sent, once again, to the *global position filtering* step. The loop continues as long as sensor data is provided.

6

Experimental Evaluation

The developed global localization system is evaluated through experiment. The purpose is to produce and evaluate results for the entire system and not just specific components. The data used in the experiment is collected by driving a vehicle, that carries the sensors, along different routes. The goal is to provide an estimated global position of the vehicle based on that data. First in this experiment chapter, the hardware and software of the system are described, then the routes and their results are presented. The results shows the ICP matches and their error to ground truth, along with the error of the EKF which estimates the vehicle position.

6.1 Hardware and Software Setup

The hardware setup is described in Section 3.1, and viewed in Figure 3.2. In summary, there is a Velodyne LIDAR sensor attached to the roof of the vehicle which has an integrated GNSS and IMU. There is also a Sokkia GRX1 attached to the roof. CAN bus data from the vehicle is measured as well.

The software is written in C++, where *Point Cloud Library* (PCL) described in Section 2.7 is partially used. The system is implemented as three different ROS nodes [29], see illustration in Figure 6.1. The nodes corresponds to the three steps in the system, *pre-processing* (Chapter 3), *global position estimation* (Chapter 4), and *global position filtering* (Chapter 5), which have been described in the previous chapters.

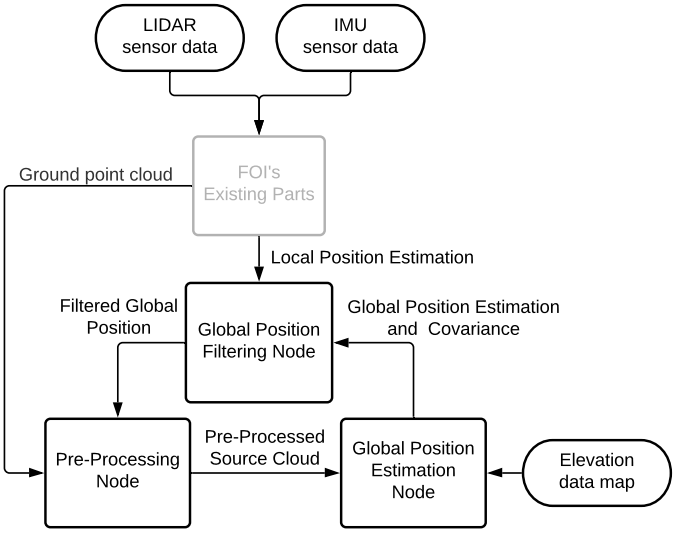


Figure 6.1: *Illustration of the node system.*

6.2 Global Localization System Variations

Two variations of the global localization system are created for comparison, and they are listed below. Both variations have the setup seen in Figure 6.1, but different algorithms in the *global position estimation* node. The parameters used in the two system variations are found in Table 6.1 and 6.3. Note that the parameter σ is used in both covariance with Hessian, (2.19), and in UT based covariance, (2.37). More details about the other parameters are found in the previous chapters.

- **System variation 1** uses the ICP implementation, described in Section 4.1.1, which performs the algorithm in two steps. First, with the subset clouds which helps avoid wrong convergence as described in Section 3.4 and 4.1.3. After that, ICP is performed again but with the whole clouds to take all points into account. The extension method that removes outliers after ICP, described in Section 4.1.4, is used as well to get a more accurate fitness score. ICP matches are either accepted or discarded depending on their fitness score, described in Section 4.1.2. The covariance is computed using the method based on Hessian, which has the second best *NEES* value out of the three tested methods, but it is still optimistic so the system will put a lot of trust in the position estimates from ICP. If more work would be put in optimizing the parameters, the covariance could be made less optimistic. As mentioned before, the ground truth used in the *NEES* computations have an accuracy of < 3 m. Therefore, it is possible that the covariance is less optimistic than it occurs. The parameters used in this method are listed in Table 6.1.

- **System variation** 2 uses the UT based ICP algorithm described in Section 4.1.4. It also has both the subset and whole clouds as input. The extension method that removes outliers after ICP, described in Section 4.1.4, is used as well. This system variation calculates the UT based covariance, described in Section 2.4.3 and motivated in Section 4.2, which is used by the EKF. Fitness score is used in this method as well. The parameters that are used are listed in Table 6.1 and 6.3.

An additional system variation is evaluated in the first route as well, this variation uses the exact same implementation as **System variation 1** with the exception that it only uses whole clouds instead of both subset and whole. This variation provides the same kind of result on all of the routes, and its result is therefore only presented in the first route.

Table 6.1: *Common parameters for both methods.*

Minimum inclination in subset cloud	$6°$
Fitness score threshold	$0.12\ m$
Voxel sampling box size	$0.06 \times 0.06 \times 0.06\ m^3$
Neighbours used in statistical outlier removal	150
STD multiplier used in statistical outlier removal	1
Radius search for estimating normals in source cloud	$1\ m$
Radius search for estimating normals in target cloud	$1.4\ m$
ICP iterations when first using subset clouds	100
ICP iterations when using whole clouds	10

Table 6.2: *Parameters used in system variation 1.*

White noise variance, σ	$5\ m$

Table 6.3: *Parameters used in system variation 2.*

Standard deviation of translation, σ_{cov,x_t}	$\frac{0.2}{\sqrt{3}}\ m$
Standard deviation of rotation, σ_{cov,x_r}	$\frac{10}{180 \cdot \sqrt{3}} \cdot \pi\ m$
α	1
β	2
κ	144
White noise variance, σ	$0.1\ m$

6.3 Routes

In this section, three different routes and their results are presented. The vehicle is stationary at the start and end of the route, in between the speed is kept at 20-30 km/h except at intersections. The localization results using the two system variations, described in Section 6.2, are presented in the subsections belonging to each of the routes. The starting position of each route is assumed to be known and evaluating the localization along the route is the focus.

6.3.1 Ground Truth

For ground truth, the current GNSS position is saved for each given EKF estimate and each ICP match. As mentioned before, both a Sokkia GRX1 and an integrated GNSS measures the position of the vehicle. The integrated GNSS updates its position four times each second, while the Sokkia GRX1 only updates once each second. During good conditions, the Sokkia GRX1 usually provides very accurate positioning. In this experiment, it unfortunately did not find a fix solution very often. The cause is probably that the used Sokkia GRX1 is rather old and has access to fewer satellites than the more modern ones. By comparing positions from the Sokkia GRX1 that received a fix solution with the corresponding positions provided by the integrated GNSS, it is shown that the error is < 3 m. Therefore, the integrated GNSS is seen as more suitable to use as ground truth.

6.3.2 Urban/Field

The first route to be evaluated is shown in Figure 6.2. The route starts in an urban environment and then goes along a field. This route is chosen to observe the behaviour of the localization system in an area where houses have been removed. This often causes dissimilarity in the source and target cloud, discussed in Section 4.1.3. There are also some field areas in the route with ditches on each side of the road, which as mentioned before is a helpful environment feature.

Figure 6.2: *Aerial map with route 1 marked, the vehicle moves from point A to B. The map is from the website Eniro, and is produced by Visma Optiway.*

Figure 6.3 shows the graphic result as a trajectory seen from above when using system variation 1 that uses the ICP algorithm without extensions, and the method based on Hessian to compute the covariance. Figure 6.4 is the result from system variation 2 where the UT based ICP is used to compute covariance, which affects how much the EKF trusts accepted ICP matches.

In the figures, the dark blue line is the global position filtered by the EKF. The ICP matches used as observations in the EKF are green (if accepted match) and yellow (if discarded match) dots. If a match is accepted the EKF will update its position. As seen in the figures, the EKF updates its position a while after the green matches. This is because the vehicle travels while the ICP algorithm is

computed. The dots seen in the figure also represents the middle of each merged cloud, and not the end. The odometry path provided by FOI's system, which is used as input to the dynamical model in the EKF, is the light blue line. Since these are provided in local coordinates, the path is positioned so that the starting position is aligned with the red GNSS path. The grey map used as background represents the elevation map used as target cloud, where objects like houses and trees have been removed. The layout is the same for all figures in the different types of routes.

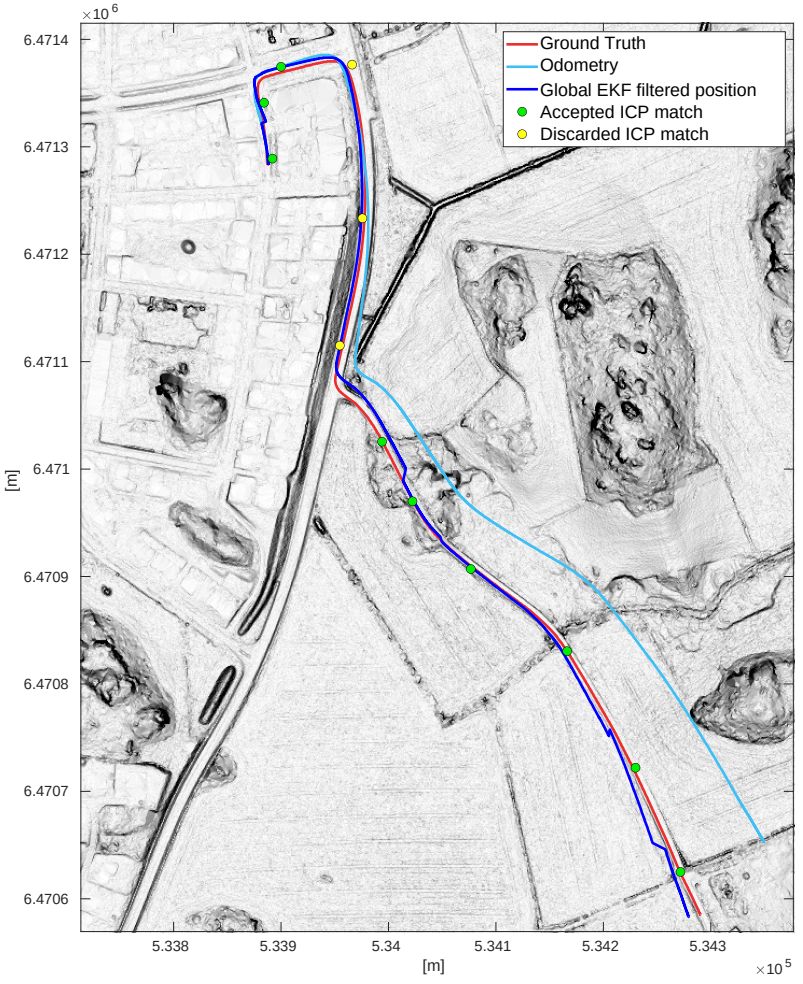


Figure 6.3: *Urban/field route using system variation 1.*

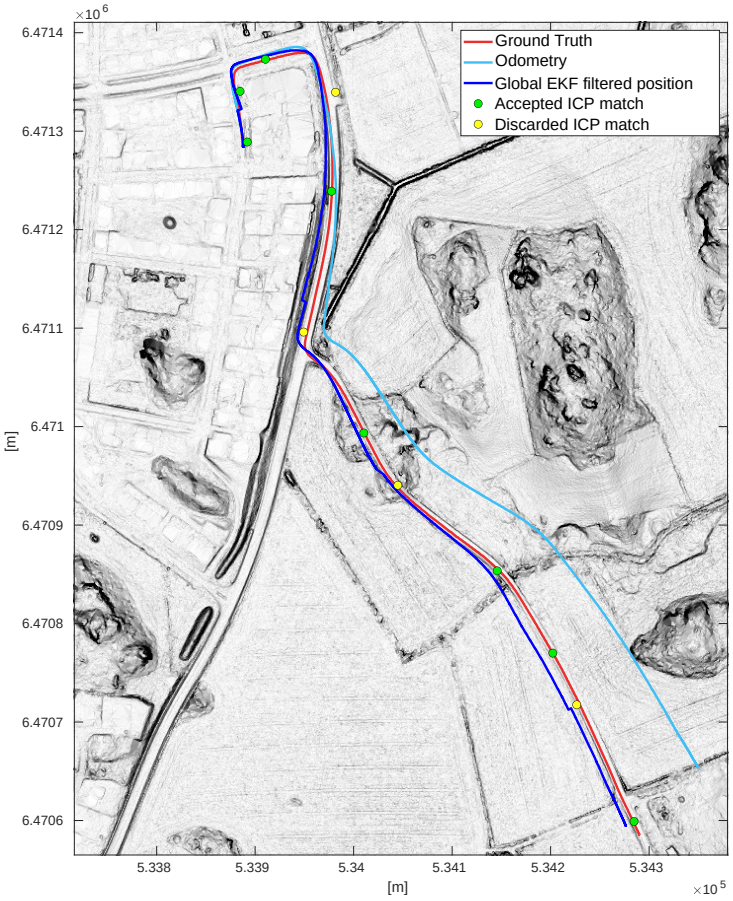


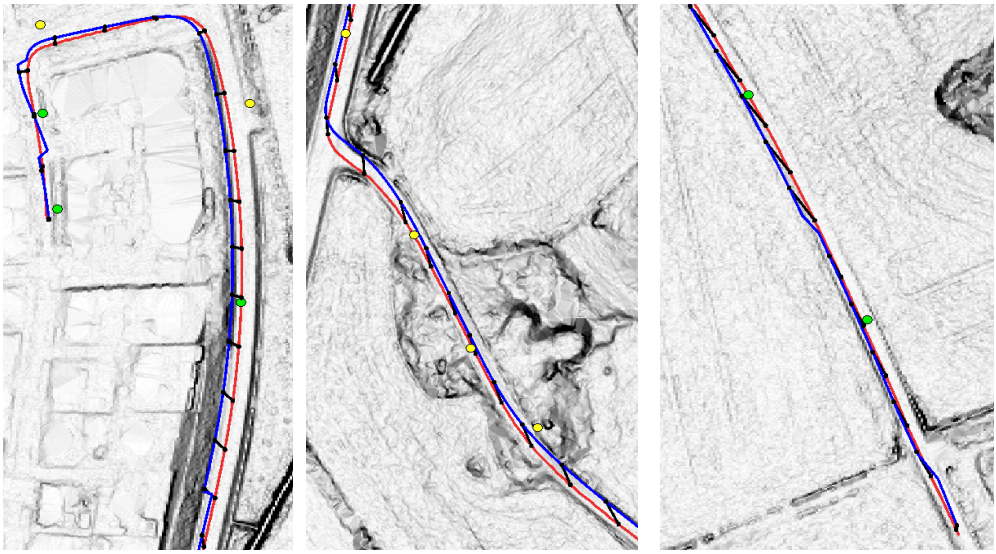
Figure 6.4: *Urban/field route using system variation 2.*

Table 6.4: *Numeric results of urban/field route using system variation 1 and 2.*

Description	Variation 1	Variation 2
Mean error of EKF path [m]	5.61	8.19
Mean rotational error of EKF path [rad]	0.038	0.039
Mean error of all accepted matches [m]	4.05	5.17
Mean error of all matches [m]	5.85	7.48
Mean computational time per ICP algorithm [s]	8.07	59.01

As seen in Figure 6.3 and 6.4, not all matches are accepted. This can be explained by dissimilarity in the source and target cloud which often is the case in areas where landmarks, such as houses and trees, have been removed, which are discussed in Section 4.1.3. The 4^{th}, 5^{th} and 6^{th} match using variation 1 and the 4^{th}

and 6^{th} match using variation 2 are not accepted, and are all in an area where there are houses in real life.



(a) First part. *(b) Middle part.* *(c) Last part.*

Figure 6.5: *Parts of the urban/field route, where the black lines represents the ground truth for that specific filtered position. These figures do not compare to any of the results presented above, it is only used as an example.*

The mean error of the EKF filtered position is 5.61 meters and 8.30 meters in variation 1 and 2 respectively, which can seem like much if looking at the paths but Figure 6.5 shows that even though the blue filtered position path is close to the red ground truth, it sometimes falls behind. It is because the EKF only has access to odometry measurements during that time since no matches are accepted, which causes drift. This can be seen in **(b)**. In **(c)**, two matches are accepted which put the filtered position on the correct path again. The rotational error is fairly small in all of the routes, which is also seen in the figures.

If comparing the two system variations, variation 1 performs better with regards to the mean error of the EKF path and the matches. As mentioned, in the UT based ICP, the ICP algorithm is performed from 13 different initial positions to be able to compute the covariance. The best result out of these 13 performances is chosen based on fitness score, however, fitness score is shown to be a risky evaluation method. This can cause the UT based ICP to choose incorrect positions some times, which can be the cause of the bad performance in this route. If a more reliable method for evaluating the results were to be found, the performance of the UT based ICP could be improved. System variation 2 also has a much higher computational time, which is because of the 13 ICP algorithms that has to be performed. For the system to keep up with the calculations, the rosbag containing

the data was played at a lower speed than real time. System variation 1 on the other hand can be used in real time since it has lower computational time.

To take into consideration when comparing the two system variations is that because of the different computational time, the source clouds are not completely the same in the two variations even though it is the same route. The initial positions of the source clouds before ICP is performed are different as well, since the EKF in the different system variations estimates different paths.

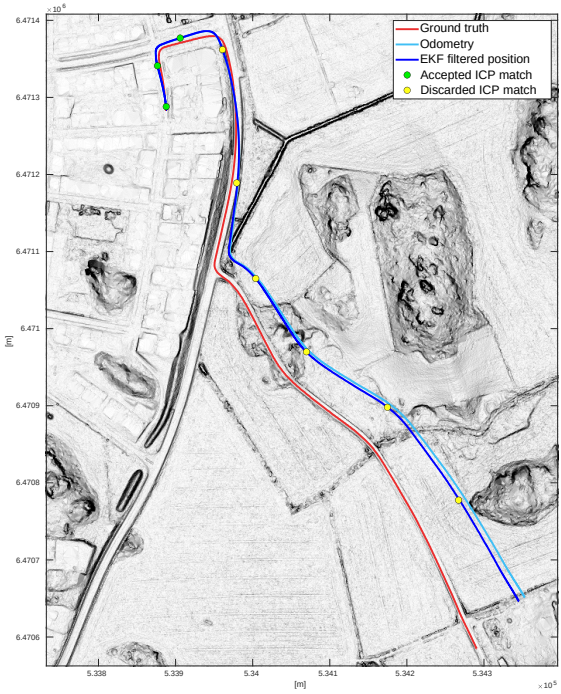


Figure 6.6: *Urban/field route using system variation 1, but only whole clouds are used as input and not the subset clouds, which gives bad result.*

The performance of the subset clouds, which are used in both of the system variations, are evaluated in this route as well as an additional system variation. As mentioned in Section 3.4, a way to increases the chance of avoiding incorrect local minimums is to choose a source and target cloud subset containing points whose normals have high inclination, as input. It is used in both of the system variations that are evaluated in this experiment. Figure 6.6 shows the additional system that is similar to System variation 1 but it uses whole clouds. If compared to the result in Figure 6.3 when subset clouds are used, the performance is drastically degraded. This result shows that using the subset clouds increases convergence to the global minimum. If using only the whole source clouds, they do not move towards the correct position since they get stuck in an incorrect local minimum

very quickly. This is the case since the whole target cloud contains much more local minimums than the subset target cloud and this causes the global position path, estimated by the EKF, to only follow the odometry path.

6.3.3 Field

The second route to be evaluated is shown in Figure 6.7. The route goes along a field, where a single house and some forest groves are passed. This route is chosen to test the performance of the localization system in an area with mostly clear ground and no landmarks. There are road ditches through out the whole route, which is an important environment feature. Figure 6.8 shows the graphic result of the field route using system variation 1, and Figure 6.9 shows the same route but using system variation 2.

Figure 6.7: *Aerial map with route 2 marked, the vehicle moves from point A to B. The map is from the website Eniro, and is produced by Visma Optiway.*

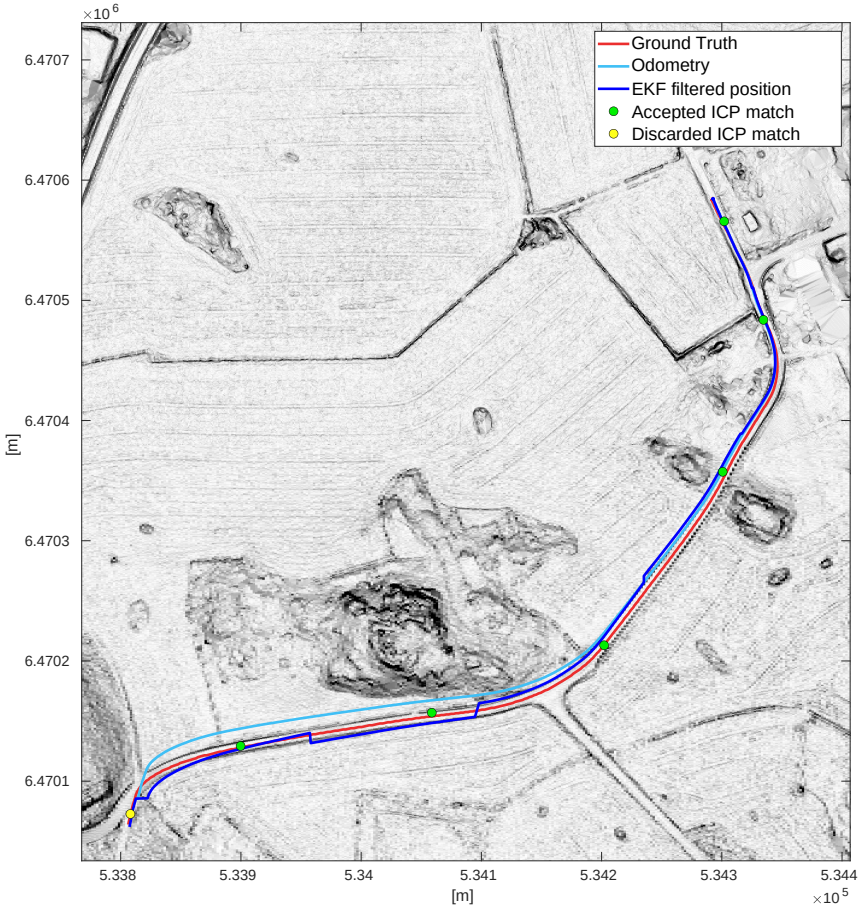


Figure 6.8: *Field route using system variation 1.*

Table 6.5: *Numeric results of field route using system variation 1 and 2.*

Description	Variation 1	Variation 2
Mean error of EKF path [m]	3.40	3.45
Mean rotational error of EKF path [rad]	0.025	0.02
Mean error of all accepted matches [m]	3.24	4.04
Mean error of all matches [m]	2.86	4.11
Mean computational time per ICP algorithm [s]	9.26	69.34

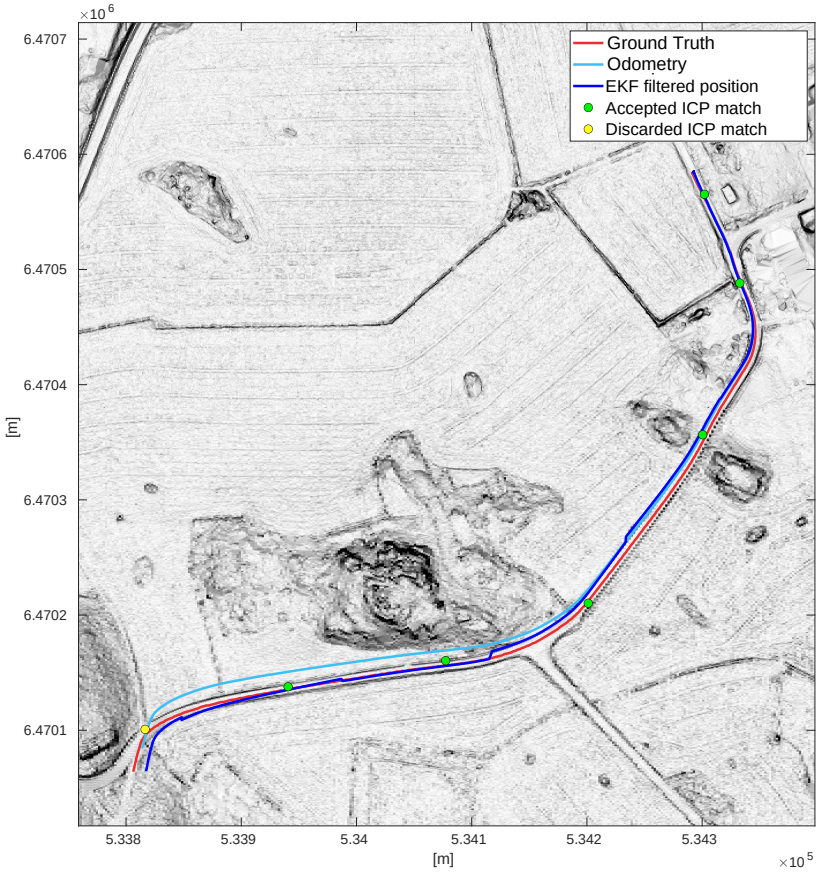


Figure 6.9: *Field route using system variation 2.*

In this route, the performance of both system variations is rather equal, but to have in mind is that the error of the GNSS used as ground truth is < 3 meter. Since a lot of matches are accepted, a lot of observations are sent to the EKF which corrects the estimated position. If comparing the EKF path in the result figures, it can be seen that system variation 1 trusts the matches more, this is because it uses the more optimistic Hessian method to compute the covariance. Therefore, the estimated position from the EKF overshoots some times as seen in Figure 6.8. System variation 2 trust the matches less, therefore, it does not overshoot as much as system variation 1.

The fitness scores compared to the actual error are found in Figure 6.10 and 6.11. The fitness score correlates to the error fairly good, but not always. The 6^{th} match in Figure 6.9 is seemingly good positioned on the road, but the error is 7.5 meters. This case is similar to the one described in Section 4.1.3, Figure 4.5, since both

areas only contains height curves in the form of ditches. It provides good positioning sideways on the road, but both areas are missing height curves in other directions. This makes it almost impossible for ICP to provide positioning along the road, since there is no information in that direction. The reason why fitness score became low is because ICP has found a local minimum which is very similar to the global minimum. The conclusion of this is therefore that fitness score is not a fully reliable method to determine the precision of the ICP results, since it can not differentiate between local and global minimums. The correlation between fitness score and error is similar in the first route and is therefore not presented there as it would not add any new insights.

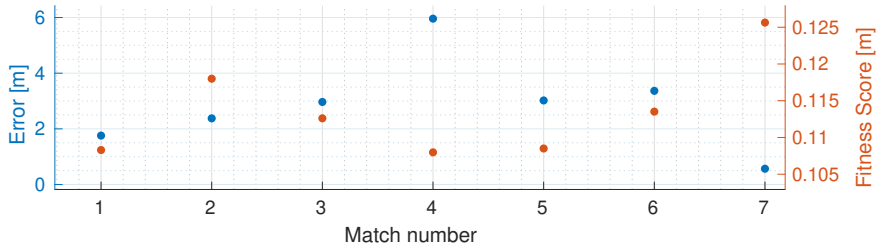


Figure 6.10: *Blue dots are error and corresponds to the left y-axis, red dots are fitness score and corresponds to the right y-axis. System variation 1.*

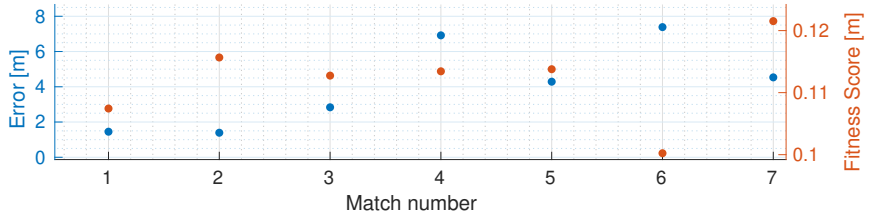


Figure 6.11: *Blue dots are error and corresponds to the left y-axis, red dots are fitness score and corresponds to the right y-axis. System variation 2.*

6.3.4 Forest

The last route to be evaluated is shown in Figure 6.12. This route is chosen to be in the experiment to evaluate the performance of the localization system in a dense forest area. Based on discussions about similarity of point clouds, Section 4.1.3, this is the worst condition for ICP. Figure 6.13 shows the graphic result of the forest route using system variation 1, and Figure 6.14 shows the results using system variation 2.

Figure 6.12: *Aerial map with route 3 marked, the vehicle moves from point A to B. The map is from the website Eniro, and is produced by Visma Optiway.*

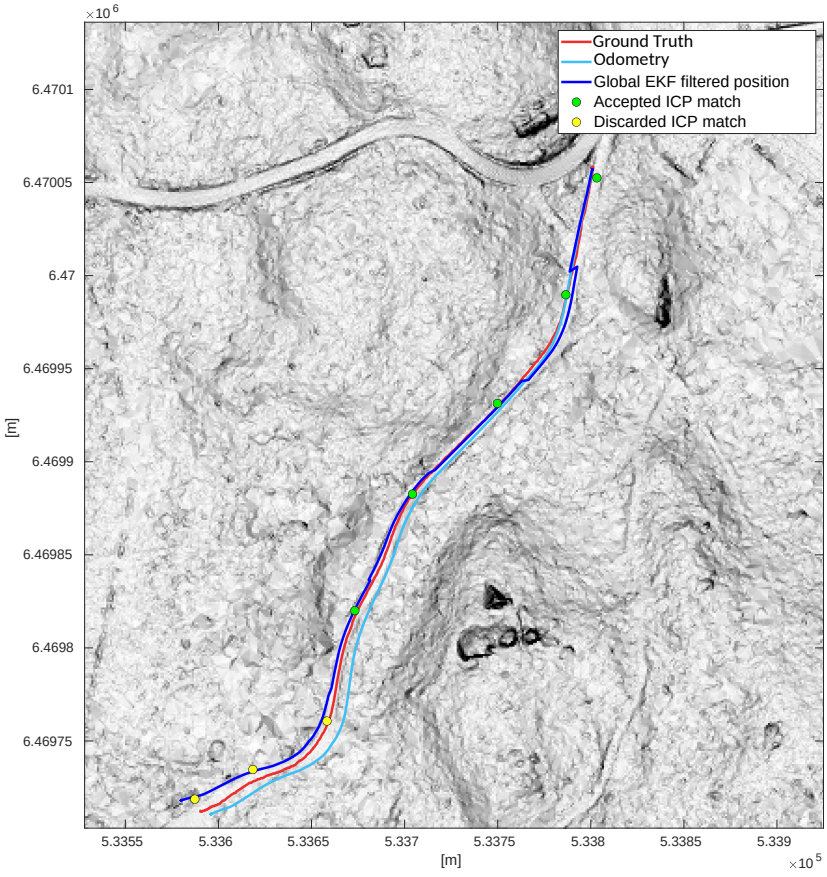


Figure 6.13: *Forest route using system variation 1.*

Table 6.6: *Numeric results of forest route using system variation 1 and 2.*

Description	Variation 1	Variation 2
Mean error of EKF path [m]	3.35	2.92
Mean rotational error of EKF path [rad]	0.073	0.069
Mean error of all accepted matches [m]	3.08	2.18
Mean error of all matches [m]	4.72	1.82
Mean computational time per ICP algorithm [s]	6.56	72.54

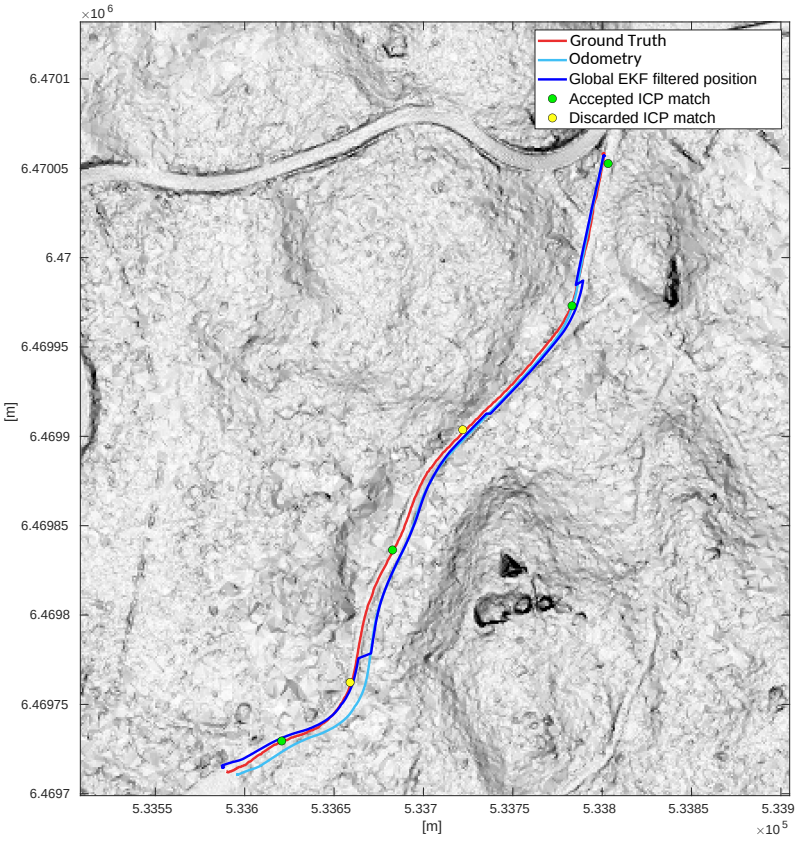


Figure 6.14: *Forest route using system variation 2.*

As discussed in Section 4.1.3, the uncertainty in dense forest areas is large because of the dissimilarity in source and target cloud, caused by the different ground approximations. This is improved by using the extension method that removes outliers after ICP, described in Section 4.1.4.

Even though some matches have found the global minimum, they are not accepted. This is because they got a high fitness score, seen in Figure 6.15 and 6.16. For example, the 3^{rd} match in Figure 6.14 has found the global minimum, but is discarded. If comparing the fitness score and error figures in the forest route, with those in the field route, there are less correlations in the forest route. If the fitness score limit is increased more matches would be accepted, but it also adds more uncertainties since wrong converged matches could be accepted.

Variation 2 performed better than variation 1 in this route. The possible reason is that variation 2 uses a less optimistic covariance estimation method, and because of the many uncertainties that a dense forest environment brings, it is wise to trust the matches less.

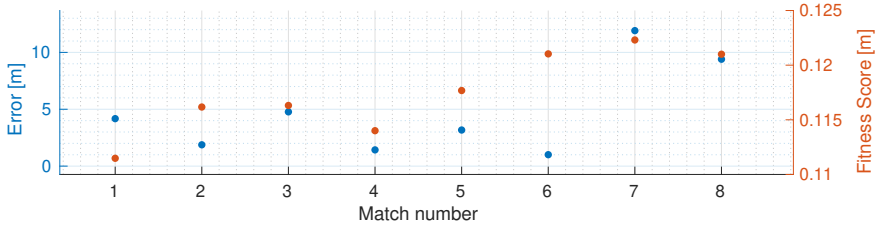


Figure 6.15: *Blue dots are error and corresponds to the left y-axis, red dots are fitness score and corresponds to the right y-axis. System variation 1.*

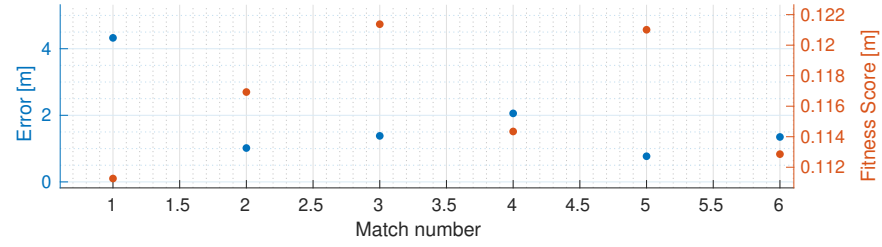


Figure 6.16: *Blue dots are error and corresponds to the left y-axis, red dots are fitness score and corresponds to the right y-axis. System variation 2.*

6.3.5 System Variation Recommendation

It is difficult to determine which system variations performs best in the overall case since different environments requires different solutions. Also, as mentioned before, the variations do not have the exact same source clouds as input, which means that they cannot be fully compared since they have different prerequisites. If comparing them separately and solely focusing on how they perform in finding the actual position and "follow" the GNSS track, then both variations are viable. Although, variation 1 has the computational advantage and had smaller errors in the urban/field and field route. In these kind of areas, the system profits from using variation 1 which trusts the matches more. It can indicate that even if a match has some uncertainties, it can still provide a better estimate than odometry. In areas with higher uncertainties, such as dense forests, system variation 2 performs better since it computes less optimistic covariance estimates. To have in mind is that the GNSS used as ground truth has an accuracy of < 3 meter. Therefore, when comparing the two variations, it is not completely certain that

that system variation 1 actually performed better at the first two routes, and vice versa.

The UT based ICP tests 13 different initial positions to be able to compute the UT covariance. The best ICP result out of the 13 initial positions is chosen based on fitness score. Since fitness score is a risky evaluation method, it is not entirely certain that the best is always chosen. Therefore, if a more reliable evaluation method was found, system variation 2 would be improved. If a graphics card were to be used, then the computational time could be decreased, since it can speed up the process by relieving some of the heavy computations off the CPU.

7

Conclusion and Future Work

In this chapter, the conclusions related to the purpose and questions that this thesis addresses are presented. Future work is found as well.

7.1 Conclusion

An ICP based global localization system that provides the position of a driving vehicle, has been developed in this thesis. The purpose of the system is to be an alternative to GNSS, which can be inaccessible or provide low precision in some environments. The developed system has been evaluated in a real time experiment where the goal was to localize a vehicle while it was driving different routes.

It has been shown that it is difficult to provide an accurate uncertainty estimate of ICP. The different methods that have been evaluated are UT based covariance, covariance with Hessian, and covariance with correspondences. The result showed that the UT based method was the least optimistic, the Hessian method was more optimistic and the correspondences method was the worst. The result from covariance with Hessian and covariance with correspondences are as expected since they only consider *sensor noise* as the source of error, while the UT based variant considers *convergence* and *under-constrained* as well. Since some sources of error, for example outliers in the source cloud or dissimilarity in source and target cloud, are not represented in any of the covariance estimates, none of the methods provide fully correct estimates. Even though the UT based covariance gave the least optimistic result, it can not be fully recommended if choosing between the three investigated covariance methods. This since it results in a very high computational time which makes it unsuitable to use in real time. If the computational time was not an issue or if the UT based system was faster, then the only thing that needs to be corrected is the optimistic covariance estimate. This can be done by adjusting the parameters to fit the system better.

The results provided by ICP have shown to be rather unreliable some times, and are not provided frequently enough to be the only source of position estimates in the localization system. Therefore, odometry position estimates were used as a second source. To merge these estimates, an EKF was implemented in the system which weights the estimates based on their covariance. As mentioned, it has been shown that the covariance of the ICP results are not fully accurate. Therefore, the performance of the EKF is decreased since it can not weight the estimates correctly. However, the experiment shows that the overall estimated vehicle positions that the EKF provides are viable and localization within a couple of meters from ground truth is provided.

The largest downside with the ICP algorithm is that it does not differentiate between local and global minimums. No solution has been found that guarantees convergence to the global minimum, but one successful method that increases this chance has been developed. The approach is to exclude local minimums by removing a large amount of points and only keeping the most informative. This because by removing points from the target cloud, some local minimums are removed as well. By removing the same points from the source cloud, the similarity is retained, and by keeping the most informative points, the ICP algorithm will not be under-constrained.

The developed global localization system is far from perfect and there are several improvements that would increase the localization performance. Using ICP with a global point cloud map as reference together with odometry to provide the position estimates works, but is not optimal at all times. Although it provides robustness in such way that the system works regardless of landmarks. However, too many landmarks, as in a dense forest, can decrease the performance of ICP since the source and target cloud becomes less similar in these areas. It is believed that the best approach to increase robustness to the system would be to integrate it with FOI's existing feature based SLAM system. It would result in even more sources of position estimates that the system can rely on. Other improvements are found in the next section about future work.

7.2 Future Work

In the experiment, the starting position of each route was assumed to be known. Therefore, a future improvement would be to develop a method that can find the starting position by only using the point cloud data and evaluate how plausible it is that the position is correct.

A future work would be to improve the covariance estimates by including more sources of error, so that the EKF is provided with a more accurate uncertainty of ICP. Another improvement to the system, as discussed a lot in this thesis, would be to find a better way to evaluate ICP matches than with fitness score.

Since a lot of parameters are used in the system, for example number of scans used in the merged cloud, and the size of the voxels when downsampling the cloud, more work could be done on optimizing these which probably would improve the system.

The outlier removal method that is used after ICP, described in Section 4.1.4, could be further evaluated. For example look into the relationship between the number of removed points and the accuracy of the match. If a low amount of points are removed, it could indicate that there is a good match.

As mentioned previously, an integration of this global localization system to FOI's existing system could increase robustness and give better position estimate to the EKF, but there is also the possibility that it could be changed to an unscented Kalman filter or that the EKF is not needed at all.

Bibliography

[1] Y. Bar-Shalom. Update with out-of-sequence measurements in tracking: exact solution. *IEEE Transactions on Aerospace and Electronic Systems*, 38(3): 769–777, 2002. doi: 10.1109/TAES.2002.1039398.

[2] Timothy D. Barfoot and Paul T. Furgale. Associating uncertainty with three-dimensional poses for use in estimation problems. *IEEE Transactions on Robotics*, 30(3):679–693, 2014. doi: 10.1109/TRO.2014.2298059.

[3] P. J. Besl and N. D. McKay. A method for registration of 3D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992. doi: 10.1109/34.121791.

[4] S. Bonnabel, M. Barczyk, and F. Goulette. Observability, covariance and uncertainty of ICP scan matching. *ArXiv*, 2014.

[5] S. Bonnabel, M. Barczyk, and F. Goulette. On the covariance of ICP-based scan-matching techniques. In *2016 American Control Conference (ACC)*, pages 5498–5503, 2016. doi: 10.1109/ACC.2016.7526532.

[6] M. Brossard, S. Bonnabel, and A. Barrau. A new approach to 3D ICP covariance estimation. In *IEEE Robotics and Automation Letters*, pages 1–1, 2020.

[7] Andrea Censi. An accurate closed-form estimate of ICP's covariance. *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007.

[8] Zhaozhong Chen, Christoffer Heckman, Simon Julier, and Ahmed Nisar. Weak in the NEES?: Auto-tuning Kalman filters with Bayesian optimization, 2018.

[9] David Claridge and Michael Quinlan. Height map, 2013. URL `https://wiki.ros.org/velodyne_height_map`.

[10] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. URL `http://www.blender.org`.

[11] Nguyen Dinh-Van, Fawzi Nashashibi, Nguyen Thanh-Huong, and Eric Castelli. Indoor intelligent vehicle localization using wifi received signal strength indicator. In *2017 IEEE MTT-S international conference on microwaves for intelligent mobility (ICMIM)*, pages 33–36. IEEE, 2017.

[12] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part I. *IEEE Robotics Automation Magazine*, 13(2):99–110, 2006. doi: 10.1109/MRA.2006.1638022.

[13] N. Gelfand, L. Ikemoto, S. Rusinkiewicz, and M. Levoy. Geometrically stable sampling for the ICP algorithm. In *Fourth International Conference on 3D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings.*, pages 260–267, 2003. doi: 10.1109/IM.2003.1240258.

[14] Max Holmberg. Development and evaluation of a robocentric SLAM algorithm. *Uppsala University*, 2018.

[15] Cheng-Tiao Hsieh. An efficient development of 3d surface registration by point cloud library (PCL). In *2012 International Symposium on Intelligent Signal Processing and Communications Systems*, pages 729–734, 2012. doi: 10.1109/ISPACS.2012.6473587.

[16] Simon J Julier and Jeffrey K Uhlmann. New extension of the Kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI*, volume 3068, pages 182–193. International Society for Optics and Photonics, 1997.

[17] Lantmäteriet. Laser data, 2020. URL `https://www.lantmateriet.se/globalassets/kartor-och-geografisk-information/hojddata/lidar_data_nh_v2.6.pdf`.

[18] Lantmäteriet. Laser data quality description, 2020. URL `https://www.lantmateriet.se/globalassets/kartor-och-geografisk-information/hojddata/quality_description_lidar.pdf`.

[19] Jesse Levinson, Michael Montemerlo, and Sebastian Thrun. Map-based precision vehicle localization in urban environments. In *Robotics: science and systems*, volume 4, page 1. Citeseer, 2007.

[20] John X Liu. *New Developments in Robotics Research*. Nova Science Publishers, Inc, 2005. ISBN 1-59454-593-6. Chapter 5.

[21] MedCalc. Definition of arctan2, 2021. URL `https://www.medcalc.org/manual/atan2-function.php`.

[22] Ellon Mendes, Pierrick Koch, and Simon Lacroix. ICP-based pose-graph SLAM. In *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 195–200, 2016. doi: 10.1109/SSRR.2016.7784298.

[23] Marius Miknis, Ross Davies, Peter Plassmann, and Andrew Ware. Near real-time point cloud processing using the PCL. In *2015 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 153–156, 2015. doi: 10.1109/IWSSIP.2015.7314200.

[24] Linköping municipality. Maps and map services, 2020. URL https://www.linkoping.se/bygga-bo-och-miljo/fastigheter-och-lantmateri/kartor-och-karttjanster.

[25] G.P Penney, P.J Edwards, A.P. King, J.M. Blackall, P.G. Batchelor, and D.J. Hawkes. A stochastic iterative closest point algorithm. *Division of Radiological Sciences and Medical Engineering, The Guy's, King's and St. Thomas'Schools of Medicine and Dentistry, Guy's Hospital, London, SE1 9RT, UK. MICCAI 2001, LNCS 2208*, page 762–769, 2001.

[26] Jeff Phillips, Ran Liu, and Carlo Tomasi. Outlier robust ICP for minimizing fractional RMSD. pages 427 – 434, 09 2007. ISBN 978-0-7695-2939-4. doi: 10.1109/3DIM.2007.39.

[27] Sai Manoj Prakhya, Bingbing Liu, Rui Yan, and Lin Weisi. A closed-form estimate of 3D ICP covariance. *2015 14th IAPR International Conference on Machine Vision Applications (MVA)*, pages 526–529, 2015.

[28] Velodyne Puck VLP-16. URL https://velodynelidar.com/products/puck/.

[29] ROS. Nodes, 2018. URL http://wiki.ros.org/Nodes.

[30] ROS. Robot Operating System, 2020. URL http://wiki.ros.org/.

[31] ROS. rosbag, 2020. URL http://wiki.ros.org/rosbag.

[32] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001. doi: 10.1109/IM.2001.924423.

[33] Radu B. Rusu. Removing outliers using a statistical outlier removal filter, 2021. URL https://pcl.readthedocs.io/projects/tutorials/en/latest/statistical_outlier.html#statistical-outlier-removal.

[34] Radu B. Rusu. Estimating surface normals in a PointCloud, 2021. URL https://pcl.readthedocs.io/projects/tutorials/en/latest/normal_estimation.html#normal-estimation.

[35] Radu B. Rusu. Downsampling a PointCloud using a VoxelGrid filter, 2021. URL https://pcl.readthedocs.io/projects/tutorials/en/latest/voxel_grid.html#voxelgrid.

[36] Radu Bogdan Rusu. Semantic 3D object maps for everyday manipulation in human living environments. *KI-Künstliche Intelligenz*, 24(4):345–348, 2010.

[37] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, 2011.

[38] Martin Skoglund, Fredrik Gustafsson, and Gustaf Hendeby. On iterative unscented Kalman filter using optimization. *22th International Conference on Information Fusion (FUSION)*, 2019.

[39] Tomas Soler and Miranda Chin. On transformation of covariance matrices between local Cartesian coordinate systems and commutative diagrams. In *Proceedings of the ASP-ACSM Convention*, page 393–406, 1985.

[40] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623.

[41] Sokkia GRX1 GNSS Receiver user manual, 2009. URL `https://www.sokkia.com.sg/products/GNSS/uploads/GRX1_brochure.pdf`.

[42] Xsens MTi user manual, 2020. URL `https://www.xsens.com/hubfs/Downloads/usermanual/MTi_usermanual.pdf`. Page 37.

[43] Eric Wan and Ronell Merwe. The unscented Kalman filter for nonlinear estimation. In *The Unscented Kalman Filter for Nonlinear Estimation*, volume 153-158, pages 153 – 158, 02 2000. ISBN 0-7803-5800-7. doi: 10.1109/ASSPCC.2000.882463.

[44] Zhe Wang, Yang Liu, Qinghai Liao, Haoyang Ye, Ming Liu, and Lujia Wang. Characterization of a RS-LIDAR for 3D perception. In *2018 IEEE 8th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 564–569, 2018. doi: 10.1109/CYBER.2018.8688235.