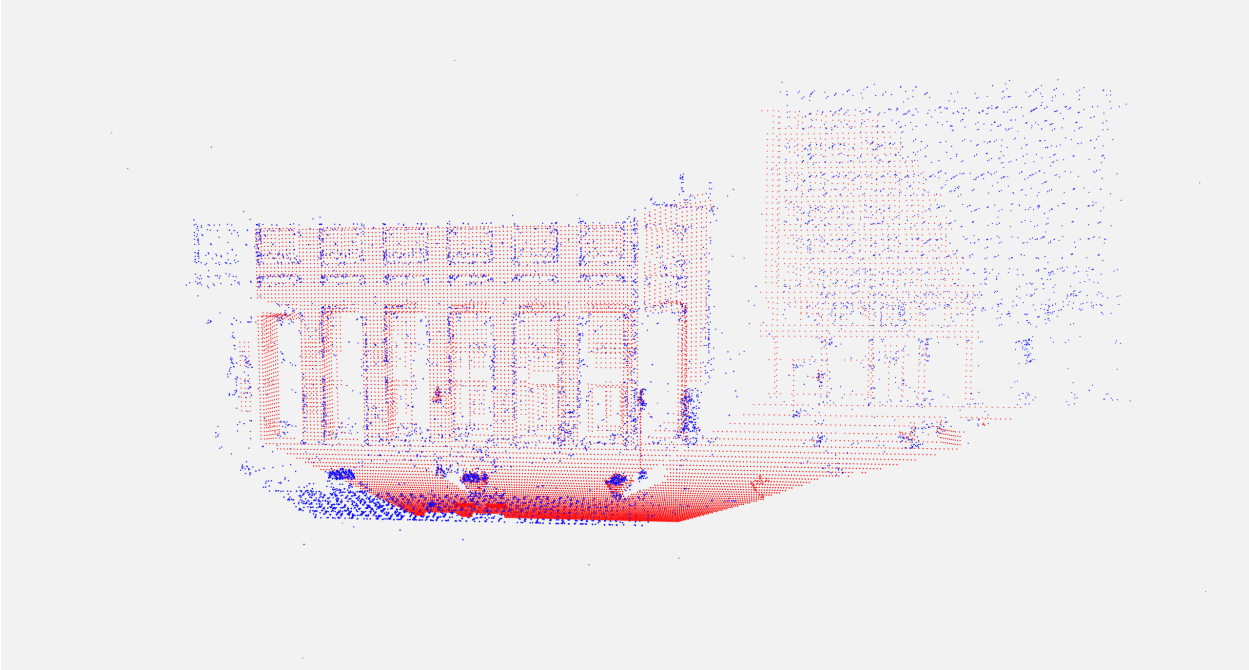


Automatic LiDAR-camera calibration

Extrinsic calibration for a LiDAR-camera pair using structure from motion and stochastic optimization

Master's thesis in Systems, Control and Mechatronics

THERESE DAHLBERG
VALTER STRÖMBERG

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022:27

Automatic LiDAR-camera calibration

Extrinsic calibration for a LiDAR-camera pair using structure from motion and stochastic optimization

THERESE DAHLBERG
VALTER STRÖMBERG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2022

Automatic LiDAR-camera calibration
Extrinsic calibration for a LiDAR-camera pair using structure from motion and stochastic optimization
THERESE DAHLBERG
VALTER STRÖMBERG

Advisor: Torgeir Langfjord Nordgård, CPAC Systems AB
Examiner: Peter Forsberg, Department of Mechanics and Maritime Sciences

Master's Thesis 2022:27
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Point clouds generated from structure from motion and LiDAR visualized together.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Automatic LiDAR-camera calibration
Extrinsic calibration for a LiDAR-camera pair using structure from motion and stochastic optimization
THERESE DAHLBERG
VALTER STRÖMBERG
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

This thesis presents an approach to automatically and simultaneously perform extrinsic calibration of a LiDAR and a camera. Nowadays, a multitude of sensors are used in a majority of vehicles. Having correctly calibrated sensors is essential for attaining accurate data to use in various sensor dependent applications. Today's LiDAR-camera calibration methods are often performed manually or require externally introduced calibration objects. However, the method proposed in this thesis is only dependent on 3D LiDAR point clouds and camera images. The method consists of two major parts. Firstly, the camera images were converted to 3D point clouds using a structure from motion pipeline, ensuring that the data from both sensors were comparable. Secondly, a genetic algorithm with an objective function based upon a 3D voxel grid filter was used to iteratively compare the overlap of the point clouds until convergence. The method proved to be successful in creating 3D point clouds from camera images and accurately estimating the rotational parameters for both sensors. However, it was not as robust and accurate as anticipated when estimating the sensor positions.

Keywords: LiDAR-camera calibration, stochastic optimization, genetic algorithm, structure from motion, point clouds.

Acknowledgements

This thesis was carried out in the spring of 2022. We would like to thank our supervisor Torgier Langfjord Nordgård for the many interesting and insightful conversations during the period. We would also like to thank our examiner Peter Forsberg for the rewarding tips that he provided. Lastly, we would like to thank CPAC Systems AB for the opportunity to do this thesis.

Therese Dahlberg & Valter Strömberg, Gothenburg, June 2022

Thesis advisor: Torgeir Langfjord Nordgård, CPAC Systems AB
Thesis examiner: Peter Forsberg, Department of Mechanics and Maritime Sciences

List of acronyms

2D	Two dimensions
3D	Three dimensions
DoF	Degrees of freedom
FoV	Field of view
GA	Genetic algorithm
GNSS	Global navigation satellite system
GPS	Global positioning system
IMU	Inertial measurement unit
LiDAR	Light detection and ranging
Radar	Radio detection and ranging
RANSAC	Random sample consensus
SIFT	Scale-invariant feature transform
SfM	Structure from motion

Contents

1

Introduction

Nowadays, a multitude of sensors is used in autonomous vehicles to collect data about the state of the vehicle and its surroundings. The sensor data has plenty of purposes and is used in various applications and systems of the vehicle. Ranging sensors are widely used in adaptive cruise control, cameras have become an extensive aid in parking assistance and positioning sensors play a huge role in navigation. Even more critical applications, such as active safety features, depend heavily on the use of various sensors.

The most prominent sensors used today are: ranging sensors - such as LiDARs and radars, imaging sensors - such as cameras, inertia sensors - such as IMUs, and positioning sensors - such as GNSS and GPS. Different types of sensors have different purposes. However, they all have one important aspect in common; the requirement of delivering accurate and reliable data for further data manipulation. For example, if the ranging sensor in an active safety system indicates incorrect distances to surrounding objects, serious complications could follow, such as activating the emergency break in the wrong situation. Consequently, counteracting the purpose of the active safety system.

To ensure that the sensors correctly record the surrounding data they need to be *calibrated*. Properly calibrated sensors are the basis for all further data manipulation and fusing, such as the examples mentioned above, but also object detection, image segmentation, Simultaneous localization and mapping (SLAM) [1], adaptive cruise control, various safety functions and more.

This thesis will focus on two types of sensors - cameras and LiDARs. Cameras and LiDARs have become increasingly popular, especially in the automotive industry. Nowadays, even other industries are starting to make use of cameras and LiDARs in their products, for example, robotic lawn mowers and autonomous vacuum cleaners. Even the boat industry is starting to implement adaptive cruise control and assisted docking algorithms, both highly dependent on being aware of the surroundings. Without a human to gather information, a camera and LiDAR can be seen as replacements for the human eye.

1.1 Calibration

Sensor calibration can be divided into two categories, *intrinsic* and *extrinsic* calibration. Both methods are quite different but equally crucial for reliably being able to use the sensor data. Intrinsic calibration aims to determine the internal parameters of a sensor. What these parameters are is different depending on the sensor. For a camera, it is related to how the image is altered in relation to reality, for example, distortion or resolution. For a LiDAR, it is for instance the range it can detect or the field of view (FoV). However, the shared property of intrinsic parameters is that they are often used in the extrinsic calibration process. Intrinsic parameters are often given by the manufacturer of the sensor, but can sometimes change over time.

Extrinsic calibration, on the other hand, intends to determine a sensor's pose or the calibration parameters, i.e. how it is placed in relation to some other actor, for example, its host vehicle. In Figure 1.1 an intuition of extrinsic calibration is depicted, where the aim is to find the transformation T (rotation R and translation t), that relates the sensor coordinate system S, to the vehicle coordinate system W. Knowing T, the sensor data can be transformed into the vehicle's coordinate system W, and consequently turned into usable information for the vehicle. For example, if a LiDAR gives a distance to a certain object, the measurement is given in relation to the LiDAR itself, i.e. in the LiDAR's coordinate system S. However, when transforming the LiDAR measurements into the vehicle coordinate system W, the distance to the object is possibly changed, which can be explained by the measurement now being in relation to the vehicle and not the LiDAR. In most cases, it is only of interest to have information in relation to the vehicle coordinate system, since it is the vehicle that has to avoid obstacles. Also, all integrated systems and applications are most commonly based upon the vehicle's coordinate system.

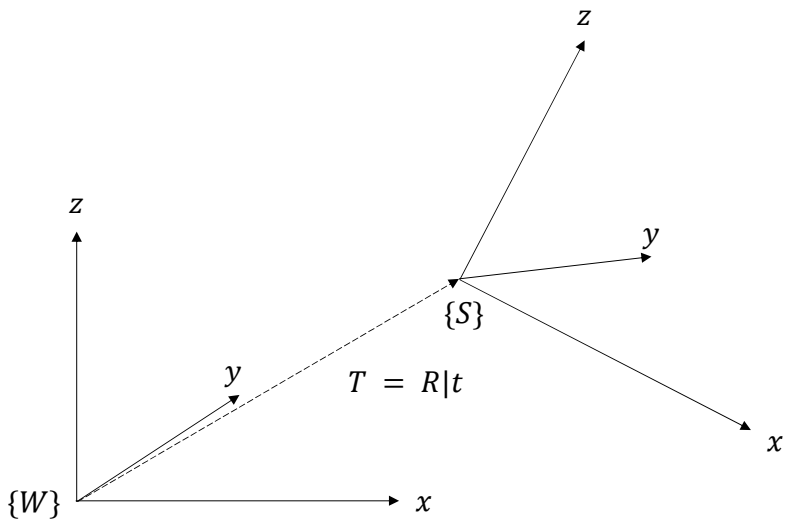


Figure 1.1: Calibration is the process of finding the Transformation T that maps the sensor's coordinate system S to the vehicle coordinate system.

Extrinsic calibration can be done manually, i.e. the distance and rotation of the sensor from the origin of the vehicle are measured by hand. The calibration is performed either during the manufacturing of the vehicle or afterward. However, it is not an easy task due to the usually obscure and inconvenient sensor placements. Extrinsic calibration can also be done automatically, i.e. using an algorithm to calculate the sensor positions.

Automatic sensor calibration algorithms can be done either online, i.e the calibration happens as the vehicle is operated, or offline, i.e the sensor data is collected as the vehicle is operated but the calibration is done afterward. An advantage of online calibration is that the sensors can be re-calibrated if their pose is altered due to external influences. However, offline calibration enables the use of more powerful computers, and thus more computationally heavy algorithms can be used. Another difference is the time constraint imposed on the algorithm in online calibration, while this is usually not an issue for algorithms that run offline.

1.2 Aim

The aim of this thesis is to develop, test and validate a fully automatic calibration algorithm for LiDAR-camera extrinsic co-calibration, thus eliminating the need for manual interference. Co-calibration means that only one calibration algorithm is used to simultaneously calibrate both camera and LiDAR. Given a rough estimation of the calibration parameters, the algorithm should be able to find the relation between an arbitrary number of cameras and LiDARs that has an overlapping FoV.

1.3 Related work

The subject of extrinsic sensor calibration is well studied and there are a great variety of approaches to take inspiration from. The biggest complication when it comes to LiDAR-camera calibration is to properly process and compare the data collected by each of the sensor types. LiDAR data consists of clusters of points in 3D, point clouds, whereas camera data is depicted in 2D. This section will introduce the most relevant sources of LiDAR-camera calibration along with some content on relevant optimization algorithms.

1.3.1 LiDAR-camera calibration

The most prominent dilemma when working with cameras and LiDARs simultaneously is that they operate in different dimensions, making it hard to compare the raw data of the two sensor types. A solution to this problem is to either remove one dimension from the LiDAR data or add one dimension to the camera data. It is important in both cases to find shared information as it is the baseline for producing a satisfactory calibration result. A common way to find shared information is to extract interesting regions in the scene called *features*.

3D to 2D conversion of LiDAR data

The technique of converting 3D point clouds to 2D starts by finding the same features in the point cloud and image. The point cloud is then projected down on the image and thus removing one dimension. The features in the image and the projected LiDAR data can now be compared in the same dimension. To make sure that the data align as well as possible, the data is iterated through an optimization algorithm. There exist different methods for extracting equivalent features in both point clouds and images.

Firstly, *edge detection* is a method for finding so-called edges, e.g. corners or natural occurring lines resulting from shadows in an image. The edges are then converted to lines and can be compared with the same occurring lines in the point cloud data. By applying an optimization algorithm the best match of overlapping lines can be found, which in turn can be used to find the extrinsic calibration parameters [2].

Secondly, [3] describes a *line correspondence* calibration method, which utilizes the concept of so-called infinity lines to achieve a similar comparison as [2]. Lastly, calibration objects or targets can be used as a fixed feature with known dimensions. Calibration targets introduce known information in both the LiDAR and camera data, thereby simplifying the optimization process [4].

2D to 3D reconstruction of camera data

The method of converting the 2D camera data to 3D requires more than one image of the scene. The reason for this is that it is possible to extract the depth information given two or more images of the same scene. In *stereo vision*, a stereo camera - two parallel cameras mounted on a rigid frame - produce image pairs of the scene, whereupon the depth can be extracted. The depth data is then used to create 3D points depicting the scene. These points can then be compared with the LiDAR 3D points. A common approach in calibration using 2D to 3D reconstruction is to introduce calibration targets to ensure that the LiDAR and camera have one common known object in their vicinity. The calibration target is then used to perfectly overlap the point clouds. From there the sensor poses can be extracted [5], [6]. Optimization is then used to find the most accurate overlap of the compared sensor data.

Another method is structure from motion (SfM) which does a 3D reconstruction of camera images taken of the same scene. The approach is similar to stereo vision. Images are used to construct a 3D point cloud of a scene by estimating depth data from two or more images. It is, however, different from stereo vision since only one camera is needed to collect data. In [7] an SfM pipeline was used to reconstruct camera data into 3D points for comparison with LiDAR data on the fly. One advantage of SfM is that it can be done with only one camera and thus the calibration is not dependent on including a stereo camera. It is also worth noting that a densification algorithm was used on the reconstructed point clouds to increase the amount of information extracted from the images. An example of the SfM pipeline is presented in [8], where the authors reconstructed the city of Rome

in 3D based on only images.

1.3.2 Biologically inspired optimization algorithms

Previous work, namely [9], shows that a calibration method for 2D LiDARs has been successfully developed using a genetic algorithm (GA). This algorithm was compared with the more common method of Iterative closest point (ICP) and was shown to be superior in almost all cases. In [10], a fully automatic calibration method using 3D LiDARs and biologically inspired algorithms was developed. It produced satisfactory results using both Particle swarm optimization (PSO) and a genetic algorithm. Also, the method should work in any environment and with any number of sensors, and all sensors can be calibrated simultaneously. [10] will be used as a baseline for this thesis.

1.4 Research questions and limitations

Since biologically inspired optimization has proven effective in previous work, mainly [9] and [10], it is a suitable foundation for this thesis as well. Based upon this reasoning the following research questions are investigated:

- *How can an SfM pipeline be used to convert camera images to 3D point clouds of satisfactory standards when it comes to comparison with 3D LiDAR data?*
- *How will an objective function that has been proven successful in LiDAR-LiDAR calibration [10], perform if used in a LiDAR-camera co-calibration built on the same optimization method?*
- *How does the calibration algorithm perform with regards to robustness, specifically disturbances, and how will it affect the quality of the estimated parameters?*

1.4.1 Limitations

This work will assume that all sensors produce noise-free data. Additionally, it is assumed that vehicle data collected from IMU and GNSS sensors are available, meaning that ground truth values for simplifying the testing are accessible. All sensor data will be collected from simulations exclusively. It is also assumed that all sensors used for calibration have the same number of degrees of freedom (DoF). Moreover, the sensors are assumed to already be intrinsically calibrated and there will be no need to re-calibrate during simulations and tests. Additionally, it is assumed that the scenes used to collect data are rich in features and that objects are in the range of both sensors. Furthermore, the simulated data will only be tested during linear movement, i.e. no sharp turns or fast changes in direction will be enforced on the sensors used.

Only an offline calibration algorithm will be in focus in this thesis, removing time constraints and computational complexity from the implementation. Lastly, the

calibration parameters will only be determined in relation to each other and not the vehicle.

2

Theory

In this chapter, the relevant theory is presented. First, each of the sensors used - LiDAR and camera - are introduced, followed by methods for converting 2D camera images to 3D points. Lastly, the optimization method used, a genetic algorithm, and all its related elements are presented.

2.1 LiDAR properties

LiDAR stands for Light Detection And Ranging. By emitting and detecting light at a specific wavelength only, LiDARs measure distances by timing the return of the reflected light rays. Therefore, LiDARs only capture specific points, meaning that the more points captured the higher resolution of the collected data. The clusters of points generated by LiDARs are often referred to as point clouds. A single laser will produce a point cloud in 2D, showing the distance to objects from the sensor but not their height. If multiple lasers at different angles are used, the point cloud will be in 3D.

Rotating LiDARs have lasers mounted on a revolving frame. The rotational element is used to collect data from a 360° field of view. One important aspect to take into consideration when it comes to rotating LiDARs is that a time distortion will be introduced in the data if the LiDAR is moving while capturing data.

Solid state LiDARs, on the other hand, use a single laser beam to illuminate the surroundings. An array of receivers mounted both horizontally and vertically, detect the reflected beams. The removal of rotating parts has the advantage of being cheaper to manufacture but also to be more robust [11]. However, solid state LiDARs cannot capture a 360 ° field of view like the rotating LiDAR.

2.2 Camera properties

There are plenty of different types of cameras but all of them share the same basic characteristics. To illustrate and introduce some important concepts, a simplification of a pinhole camera is shown in Figure 2.1. A camera captures light rays from the vicinity to create a 2D image. The rays come from different angles but will eventually meet at one point - the projection centre O of the camera. The light

beam perpendicular to the image plane and with a zero-degree angle to the projection centre will pass through the principal point p of the image plane. The distance between the image plane and the projection centre is called the focal length f and is often measured in pixels. The focal length can be found through the following equation

$$f = \frac{w}{2\tan(\frac{fov}{2})}, \tag{2.1}$$

where fov is the field of view and w is the width of the image in pixels.

The camera properties are expressed in the intrinsic matrix K, shown in equation (2.2). K is important in a lot of applications where cameras are used and is usually estimated by intrinsic calibration.

$$K = \begin{bmatrix} f & \gamma & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

p_x and p_y are the x and y coordinate of the principle point and γ is a skew coefficient which often is 0.

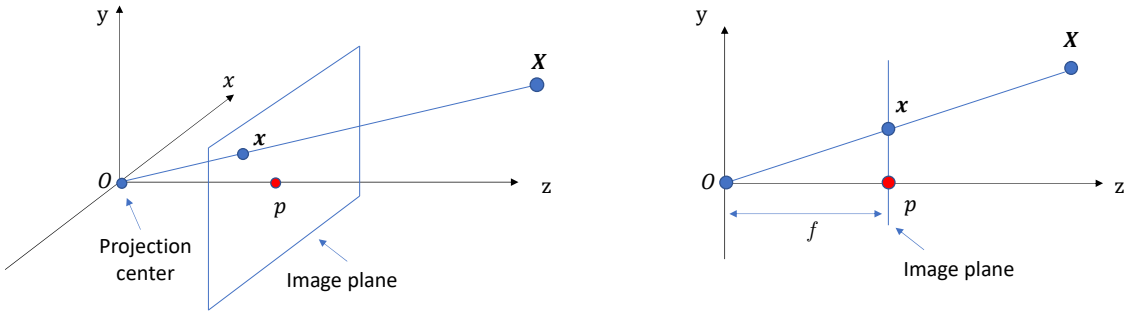


Figure 2.1: Basic illustration of camera mechanics. **x** is the point in the image plane where **X** projects to. **X** is a point in the world and is represented by Cartesian coordinates. **x** on the other hand, is described with image coordinates. The image coordinate system often has its origin in the middle of the image plane.

2.3 SIFT

Scale-invariant feature transform, or SIFT for short, is an algorithm to extract image features [12]. SIFT features are invariant to some basic image manipulations, such as scale, rotation and illumination, which makes it a robust method for feature extracting. In addition to key feature points, e.g. corners, certain shapes, or other distinct features in an image, SIFT also provides some useful quantitative information about the extracted features. The steps in SIFT are here described briefly. The first step is to apply a difference of Gaussians on the image, followed by extracting local extrema to create a scale-space. Next, gradients are computed around each found extrema. The orientations of the gradients are then downsampled and put into a histogram. The most prominent direction and the corresponding scale are

chosen as the descriptor of a particular key feature point. The next step after having extracted SIFT features is often to match the features, which can be done by comparing descriptors.

2.4 RANSAC

Random sample consensus (RANSAC), first presented in [13], is an algorithm that performs binary characterization (inliers and outliers) on data points to fit the best possible model. RANSAC consists of three main steps. The first is to randomly sample the number of necessary data points needed to fit the model. The next step is to fit the sampled points to the model and lastly, depending on how accurate the resulting match is, assign a score. A data point is considered an inlier if it lies within a certain predefined threshold. Otherwise, it is considered an outlier. These steps are iterated until the best model has most likely been found. RANSAC is often used in line-fitting problems.

2.5 Epipolar geometry

The material presented in this section is accessed from [14, Ch. 12] and [15, Ch. 7], and concern the subject of epipolar geometry. Two cameras in 3D space give rise to certain geometrical constraints. If the projection centres of the two cameras are at O and O' respectively, the geometry shown in Figure 2.2 is valid. If $\mathbf{X}$ is a point in 3D-space, it will together with the two camera projection centres O and O' form the *epipolar plane*. Epipolar geometry provides the basic tools for computing the projection of $\mathbf{X}$ onto the second image plane (i_2 in Figure 2.2) from the projection of $\mathbf{X}$ in the first image plane i_1, called $\mathbf{x}'$ and $\mathbf{x}$ respectively.

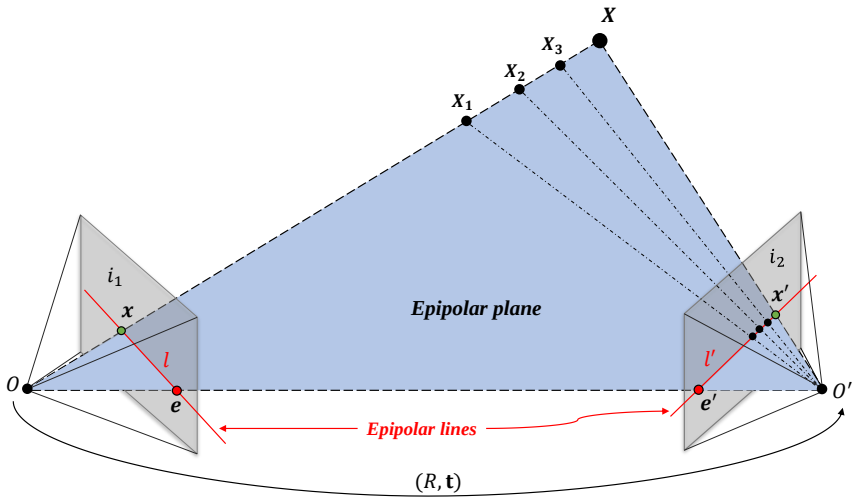


Figure 2.2: Visualization of epipolar geometry. O and O' are the projection centres of the first and second cameras respectively.

The viewing ray from the camera projection centre O, i.e. the vector $\mathbf{X} - O$, will

pass all possible scene points that projects onto $\mathbf{x}$. In Figure 2.2, $\mathbf{X}_1$, $\mathbf{X}_2$ and $\mathbf{X}_3$ are examples of this. The vector $\mathbf{X} - O$ projects on the second image plane (right image plane in Figure 2.2) as the *epipolar line*, l', marked in red. Since the points $\mathbf{X}_1$, $\mathbf{X}_2$ and $\mathbf{X}_3$ all lie on $\mathbf{X} - O$ their projections in the second image have to lie somewhere on l', thus limiting the search space from the entire image to the epipolar line exclusively.

The projection of O onto i_2 is known as the *epipole* $\mathbf{e}'$, and it also lies on the epipolar line, l'. In fact, for all points on i_1, their formed epipolar line will pass through the epipole on i_2 (and vice versa). Moreover, the condition that all possible points - which can correspond to some point in the other image plane - have to lie on the epipolar line is called the *epipolar constraint*. A parameterization describing the epipolar line can be formulated as

$$dR\mathbf{x} + \mathbf{t} \sim l \tag{2.3}$$

where d is a scalar, R a rotation and $\mathbf{t}$ a translation.

The epipolar line of $\mathbf{x}$ could alternatively be represented by the line equation

$$\mathbf{l}^\top \mathbf{x} = 0. \tag{2.4}$$

To find the equation of this line, two points on the line must be chosen and inserted in the line equation, i.e.

$$\begin{cases} \mathbf{l}^\top (R\mathbf{x} + \mathbf{t}) = 0 \\ \mathbf{l}^\top \mathbf{t} = 0 \end{cases}. \tag{2.5}$$

Equation (2.5) can be rewritten as

$$\begin{cases} \mathbf{l} \bullet (R\mathbf{x} + \mathbf{t}) = 0 \\ \mathbf{l} \bullet \mathbf{t} = 0 \end{cases}, \tag{2.6}$$

indicating that $\mathbf{l}$ needs to be perpendicular to both points since if the dot product of two vectors is zero they are perpendicular. Taking the cross product of the two chosen points, a third vector perpendicular to the two points is created as

$$\mathbf{l} = \mathbf{t} \times (R\mathbf{x} + \mathbf{t}) = \underbrace{\mathbf{t} \times \mathbf{t}}_{=0} + \mathbf{t} \times (R\mathbf{x}), \tag{2.7}$$

and thus the sought after $\mathbf{l}$ is found. Additionally, since $\mathbf{t} \sim \mathbf{e}'$, equation (2.7) can be formulated as

$$\mathbf{l} = \mathbf{e}' \times (R\mathbf{x}) = \left[\mathbf{e}'\right]_\times R\mathbf{x}. \tag{2.8}$$

where $\left[\mathbf{e}'\right]_\times$ is the cross product matrix operator of $\mathbf{e}'$:

$$\left[\mathbf{e}'\right]_\times = \begin{bmatrix} 0 & -e'_3 & e'_2 \\ e'_3 & 0 & -e'_1 \\ -e'_2 & e'_1 & 0 \end{bmatrix} \tag{2.9}$$

The epipolar constraint can now be written as

$$\mathbf{x}'^{\top} \left[\mathbf{e}'\right]_{\times} R\mathbf{x} = 0, \tag{2.10}$$

if $\mathbf{x}'$ is the corresponding projection of $\mathbf{x}$ in i_2. What $\left[\mathbf{e}'\right]_{\times} R$ actually does is mapping any point $\mathbf{x}$ in the first image to the corresponding epipolar line in the second image. The matrix $[\mathbf{e}']_{\times}R$ is called the fundamental matrix and is further explained in Section 2.6.

2.6 Essential and fundamental matrix

The essential matrix E, introduced by [16], just as the fundamental matrix, F, are widely used in camera geometry. It is with E or F that the epipolar lines can be found, as described in Section 2.5. Both E and F fulfill the epipolar constraint as $\mathbf{x}'^{\top}E\mathbf{x}$ or $\mathbf{x}'^{\top}F\mathbf{x}$, compare with equation (2.10).

The essential matrix depends only on the extrinsic camera parameters, meaning that to find E, the intrinsic matrix K of both cameras must be known. If the intrinsic parameters of the cameras are unknown, it is not possible to perform normalization of the image coordinates, which is a requirement for using E. E is defined by

$$E = [\mathbf{t}]_{\times}R \tag{2.11}$$

or alternatively

$$E = [\mathbf{e}']_{\times}R \tag{2.12}$$

to coincide with the notation used in Section 2.5.

F on the other hand, can be seen as a generalization of E since it does not require the intrinsic parameters to be known. F is dependent on both the intrinsic and extrinsic parameters of the cameras, which means that the image points will be in pixel coordinates. It is also pixels that are related to epipolar lines in the epipolar constraint. F is related to E through equation (2.13)

$$E = K'^{\top}FK, \tag{2.13}$$

and the epipoles are the left and right null spaces of both F and E.

Both E and F are rank deficient and have a determinant of zero. However, unlike the essential matrix which has five degrees of freedom, the fundamental matrix has seven. To find F and E it is common to use the N-point algorithm described in Section 2.7.1.

2.7 Structure from motion

Structure from motion (SfM) is a technique used to reconstruct 3D point clouds from a set of 2D image correspondences along with recovering the 3D camera poses,

also known as Euclidean reconstruction. One important property of SfM is that the method can only reconstruct 3D views up to a scale, meaning that only the ratio and direction of the reconstructed points are given. If the scale parameter is to be decided, further information about the environment is needed. A simple example is shown in Figure 2.3. The information presented in the following section is retrieved from [14, Ch. 11].

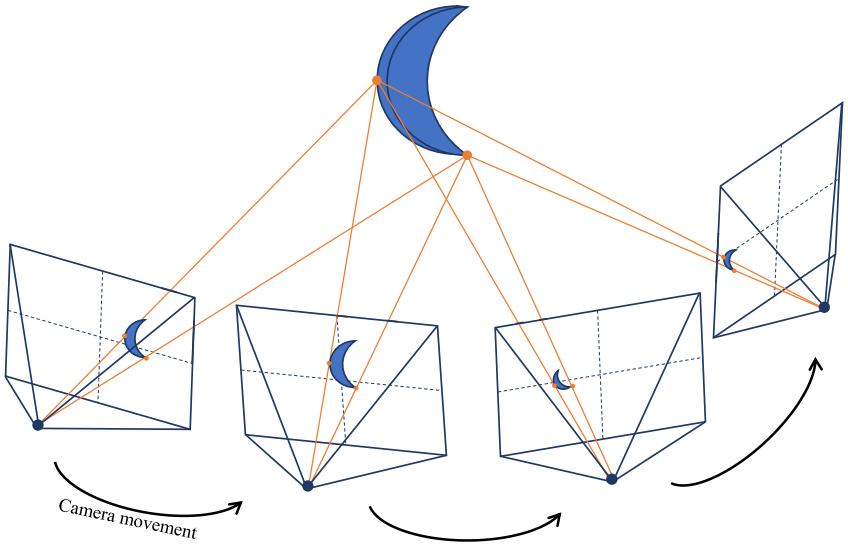


Figure 2.3: A simple structure from motion example. The blue moon object projects to different places in the image plane depending on the camera pose the moment the image is taken. This information can be used to find the camera movement and a set of 3D points of the object.

The first step of performing SfM is to find corresponding key feature points between images, which can be done by using any feature extracting method, e.g. SIFT, as mentioned in Section 2.3. The key feature points are then matched and the outliers are often found and rejected by using RANSAC, described in Section 2.4. Knowing a set of matching key feature points in two images, the relative pose between the cameras can be established by forming the following camera matrix P

$$P = K \begin{bmatrix} R & | & \mathbf{t} \end{bmatrix}. \tag{2.14}$$

K is the intrinsic matrix, R a rotation and $\mathbf{t}$ a translation to the other camera. More specifically, P relate 3D points to 2D image points as

$$\mathbf{x} = P\mathbf{X}, \tag{2.15}$$

a property used to establish the relative relationship between two cameras. Assuming two cameras in $\mathbb{R}^3$, the respective camera matrices, P and P', can be represented by

$$P = \begin{bmatrix} I & | & \mathbf{0} \end{bmatrix} \tag{2.16}$$

and

$$P' = \begin{bmatrix} R & | & \mathbf{t} \end{bmatrix}, \tag{2.17}$$

since the pose between the cameras is relative. Using equation (2.3), a point $\mathbf{x}$ in the first image, is projected onto the second image by

$$\mathbf{x}' = dR\mathbf{x} + \mathbf{t}, \tag{2.18}$$

where d is a scalar. Furthermore, equation (2.18) indicates that $\mathbf{x}'$ has to lie somewhere on the epipolar line of the second image, as described in Section 2.5.

Now, if both sides of equation (2.18) are multiplied with the cross product matrix operator of $\mathbf{t}$

$$\left[\mathbf{t}\right]_\times = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix}, \tag{2.19}$$

the following equation is formed

$$\left[\mathbf{t}\right]_\times \mathbf{x}' = d\left[\mathbf{t}\right]_\times R\mathbf{x} + \underbrace{\left[\mathbf{t}\right]_\times \mathbf{t}}_{=0}. \tag{2.20}$$

Multiplying both sides with $\mathbf{x}'^\top$ gives

$$0 = \mathbf{x}'^\top \left[\mathbf{t}\right]_\times \mathbf{x}' = d\mathbf{x}'^\top (\left[\mathbf{t}\right]_\times R)\mathbf{x}, \tag{2.21}$$

which follows from $\left[\mathbf{t}\right]_\times$ being a skew-symmetric matrix and therefore the left hand side of the equation above cancels out to zero. What is left on the right-hand side of the equation is now the epipolar constraint, see equation (2.10) for reference,

$$\mathbf{x}'^\top \underbrace{(\left[\mathbf{t}\right]_\times R)}_{=F} \mathbf{x} = 0 \tag{2.22}$$

with d omitted since it is only a scalar. From the above derivations it is clear that to find P and P', F or alternatively E, has to be known.

2.7.1 N-point algorithms

The task of finding F or E is done by using some N-point algorithm, where N is the minimal points needed to solve an equation system. Expanding equation (2.22) with $\mathbf{x}_i = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^\top$ and $\mathbf{x}'_i = \begin{bmatrix} x'_i & y'_i & z'_i \end{bmatrix}^\top$ gives

$$\begin{aligned} \mathbf{x}'^\top_i F\mathbf{x}_i = &F_{11}x'_ix_i + F_{12}x'_iy_i + F_{13}x'_iz_i + \\ &F_{21}y'_ix_i + F_{22}y'_iy_i + F_{23}y'_iz_i + \\ &F_{31}z'_ix_i + F_{32}z'_iy_i + F_{33}z'_iz_i \end{aligned} \tag{2.23}$$

which can be written on matrix form for N entries as

$$\underbrace{\begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1z_1 & \dots & z'_1z_1 \\ x'_2x_2 & x'_2y_2 & x'_2z_2 & \dots & z'_2z_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x'_nx_n & x'_ny_n & x'_nz_n & \dots & z'_nz_n \end{bmatrix}}_{A} \begin{bmatrix} F_{11} \\ F_{12} \\ F_{13} \\ \vdots \\ F_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.24}$$

Generally, the number of points needed to solve the system of equations for F and E has to be at least the same as the number of unknown parameters in each respective matrix. However, even if the fundamental matrix has nine entries, it only takes eight different matched points, i.e. $N \geq 8$. The eight degrees of freedom derive from the camera poses only being determined up to a scale factor, i.e. only the direction and ratio between the reconstructed points and camera poses can be found. To solve for E, only five points have to be used, since the intrinsic parameters are already known and thus do not need to be estimated.

By using Single Value Decomposition (SVD) the system of equations in equation (2.24) can be solved. Equation (2.25) describes the SVD of A and the entries of F can be found in the last column of V.

$$A = USV^{\top} \tag{2.25}$$

Having found F (or E), the next step is to retrieve the camera matrices. Since $\mathbf{e}'$ and $\mathbf{t}$ are equivalent, P' can also be described as:

$$P' = \left[R \;\middle|\; \mathbf{e}' \right]. \tag{2.26}$$

The next to last step is to calculate $\mathbf{e}'$, which is done by solving for the null space of F, and then R can finally be found by using equations (2.12) and (2.13). When P and P' are known the 3D points can be recovered with triangulation as described in Section 2.7.2.

2.7.2 Triangulation

Triangulation is a method used to find a 3D point given a set of images of the same object [17]. To use this method, the camera matrices P and P' for the cameras involved must be known, see Section 2.7. In its simplest form, triangulation can be described by calculating where the projection rays of the image points $\mathbf{x}$ and $\mathbf{x}$' intersect. Figure 2.4 shows an illustration of triangulation. Equation (2.27) - (2.30) shows an example of how to mathematically solve for the sought after 3D position. This simple form of triangulation is only applicable when the images have epipolar geometry.

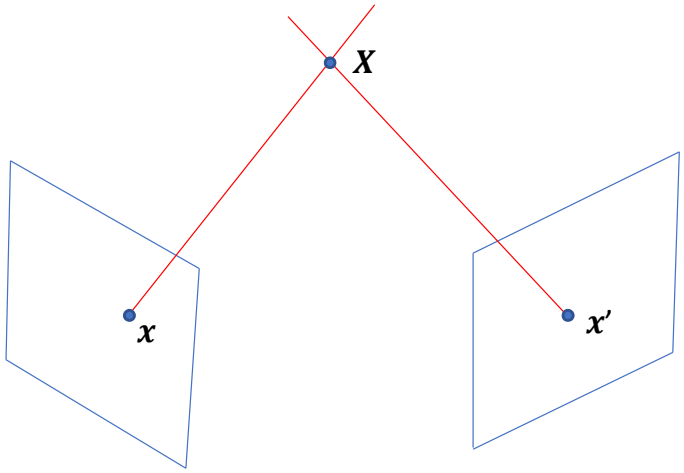


Figure 2.4: By projecting two points $\mathbf{x}$ and $\mathbf{x}'$ from two cameras it is possible to find the corresponding 3D point $\mathbf{X}$ by calculating their intersection point.

Expanding equation (2.15) in Section 2.7 for both $\mathbf{x}$ and $\mathbf{x}'$ yields

$$\begin{aligned} \mathbf{x} = P\mathbf{X} &= \begin{bmatrix} p_1\mathbf{X} \\ p_2\mathbf{X} \\ p_3\mathbf{X} \end{bmatrix} \\ \mathbf{x}' = P'\mathbf{X} &= \begin{bmatrix} p'_1\mathbf{X} \\ p'_2\mathbf{X} \\ p'_3\mathbf{X} \end{bmatrix}, \end{aligned} \tag{2.27}$$

where p_1-p_3 are the row vectors of P. The goal is to find the point $\mathbf{X}$ that satisfies equation (2.27). The first step to solve for $\mathbf{X}$ is to take the cross product of the image point $\mathbf{x}$ on both sides of equation (2.27), and since the vectors $\mathbf{x}$ and $P\mathbf{X}$ are of the same direction the cross-product is equal to zero, i.e.

$$\mathbf{x} \times \begin{bmatrix} p_1\mathbf{X} \\ p_2\mathbf{X} \\ p_3\mathbf{X} \end{bmatrix} = \begin{bmatrix} x_2p_3P\mathbf{X} - p_2\mathbf{X} \\ -x_1p_3\mathbf{X} + p_1\mathbf{X} \\ x_1p_2\mathbf{X} - x_2p_1\mathbf{X} \end{bmatrix} = 0. \tag{2.28}$$

Since the last row is a linear combination of the first two, it can be left out. With the last row removed and after a simplification, the final equation can be written as:

$$\begin{bmatrix} x_1p_3 - p_1 \\ x_2p_3 - p_2 \end{bmatrix} \mathbf{X} = 0. \tag{2.29}$$

The same steps can be done to x', resulting in the final relationship shown in equation (2.30) from which $\mathbf{X}$ can be solved.

$$\begin{bmatrix} x_1p_3 - p_1 \\ x_2p_3 - p_2 \\ x'_1p'_3 - p'_1 \\ x'_2p'_3 - p'_2 \end{bmatrix} \mathbf{X} = 0 \tag{2.30}$$

Cheirality

When it comes to camera geometry, all points found on the epipolar lines are not necessarily projected from in front of the cameras. In [18] it is stated that for a point X to be in front of the cameras it has to fulfill the following condition

$$r_3(\mathbf{X} - O) > 0. \tag{2.31}$$

Where r_3 is the third row of the rows and columns in P that represents the camera rotation matrix, R. Cheirality is an important aspect to take into consideration, for example in SfM, since points found projecting from behind the camera are obviously outliers and should not be taken into consideration.

2.8 Z-score

The *Z-score* or the *standard score* is used to measure how many standard deviations away a certain measurement is from the mean. The Z-score is useful for identifying outliers in a large data set. In a normal distribution, 99.7% of the values lies within the $\pm 3\sigma$ interval, meaning that a Z-score greater than three is highly implausible, and can therefore be considered an outlier. The Z-score is calculated as follows

$$Z = \frac{x - \mu}{\sigma} \tag{2.32}$$

where x is the measurement, μ is the mean of the collected measurements and σ is the standard deviation.

2.9 Genetic algorithm

A genetic algorithm or GA is an optimization algorithm inspired by evolution and natural selection. The theory in this section is based on [19, Ch. 3]. The GA can be summarized in three distinctive steps - initialization, evaluation, and evolution, where the latter two are repeated until termination. All steps will be further explained in this section.

2.9.1 Initialization

As in all optimization, there is a set of parameters to be optimized, $x_1, x_2, ..., x_n$ where n is the number of parameters. In a genetic algorithm, one of these sets i.e. $x_1 \dots x_n$ is called an individual. Each individual has a chromosome made out of genes that describes the set of parameters. The value of a gene can be described with equation (2.33). Where C is a chromosome and g_i is the gene that represents the i:th parameter in $x_1 \dots x_n$.

$$x_i = C(g_i) \quad \forall i \in x_1 \dots x_n \tag{2.33}$$

The values that the genes are allowed to take are restricted to a pre-defined range, called the search space. It is within this range that the algorithm will search for solutions. All the genes do not need to have the same search space.

The algorithm process starts by initializing a population of individuals with a population size N_s. A higher population size yields a higher computational complexity but does not necessarily lead to better results. The optimal population size is dependent on the application. Furthermore, to initialize all the individuals' chromosomes, each gene is given a value by uniformly drawing a random value from the specific gene's search space. The initialization is described in equation (2.34).

$$C_i(g) \sim U(lb(g), ub(g)) \quad \forall i \in N_s \tag{2.34}$$

ub and lb are the upper and lower bounds of the search space for a specific gene. It is crucial to choose a proper search space. If set too small, the solution might fall outside the search space, if too big, the algorithm might never converge.

2.9.2 Evaluation

The quality of each individual has to be evaluated during the optimization. The evaluation occurs after the individuals have evolved. The evolution process will be described in Section 2.9.3. The quality measurement is called *fitness* and is calculated with a fitness function. When all individuals have been tested for fitness, the value can be used to determine the strongest individual and thus which to favor in the next evolution process.

2.9.3 Evolution

After the evaluation step, all individuals are evolved to create a new population for the next generation. The evaluation process usually consists of the following steps.

Selection

Selection is the process of determining which individuals to be used when forming the new generation. There are many methods to perform selection, but roulette-wheel selection and tournament selection are the two most common. In roulette-wheel selection, an individual is chosen proportional to their fitness. An individual with higher fitness will have a higher chance of being selected. The method has gotten its name from the similarities with a roulette wheel, as illustrated in Figure 2.5. The probability P_i of choosing an individual is calculated as

$$P_i = \frac{F_i}{\sum_{J=1}^{N} F_j}, \tag{2.35}$$

where F_i is the fitness of individual i.

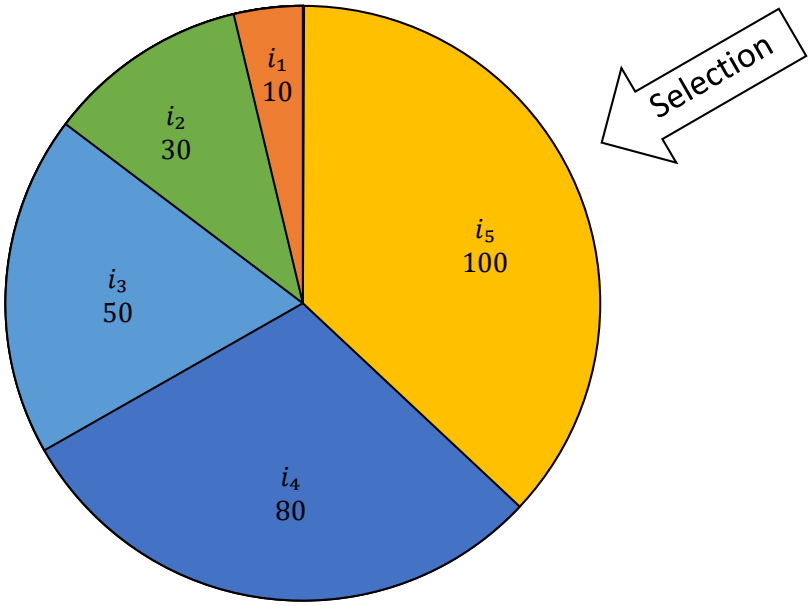


Figure 2.5: Roulette wheel selection can be interpreted as a spin of the wheel, where the size of each section is determined by the fitness of the individuals participating in the selection process.

Tournament selection, on the other hand, chooses individuals randomly to participate in a tournament. The amount of individuals selected is determined by the tournament size, j - a pre-defined parameter. From the selected individuals, a winner is determined based on its fitness. However, to give individuals with less fitness a chance, the winner is selected with the probability pattern described in equation (2.36). P_j is the probability to choose individual j where $j = 1$ represents the individual with the highest fitness of the selected individuals and so on. p_t is the probability to choose the individual with the highest fitness. Figure 2.6 is a visualization of tournament selection.

$$\begin{aligned} P_1 &= p_t \\ P_2 &= p_t(1-p_t) \\ P_3 &= p_t(1-p_t)^2 \\ &\vdots \\ P_j &= p_t(1-p_t)^{j-1} \end{aligned} \tag{2.36}$$

Figure 2.6: Tournament selection with tournament size 3. Note that, the best individual does not always win.

Crossover

Crossover is a process that mimics the reproduction process in nature. The idea is that two individuals' genes are mixed up in the hope that the best of each carries over to the new individual. Crossover is a very effective method, but can easily lead to local optima if not regulated. Therefore, crossover is only done to a portion of the individuals, determined by the probability p_c. To perform crossover, the chromosome of the selected individuals is split at a random gene. The chromosomes then swap genes at the splitting point, see Figure 2.7 where the line represents the splitting point.

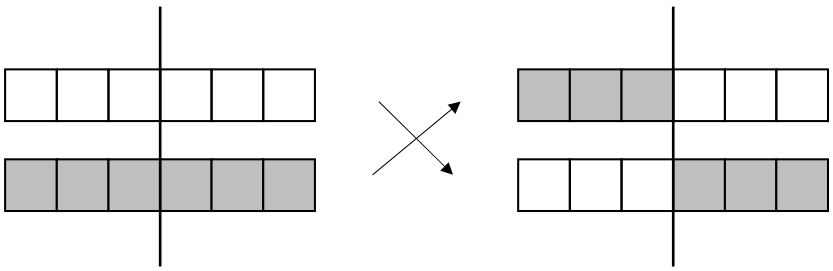


Figure 2.7: Visualisation of crossover. The line represents the crossover point where the individuals swap genes. The result is two new individuals.

Mutation

The process of mutation makes sure that the individuals evolve and new solutions are explored. Even though mutation does not always lead to better fitness it is important for the purpose of exploration. Mutation works by randomly changing some of the genes to another value in the valid search space. The probability that a gene mutates p_m is often set to m/n where m is the number of mutations per individual and n is the number of genes in one individual. A constant mutation rate is often used since it is hard to decide how the mutation rate should vary.

When a mutation occurs, the new value is drawn from a uniform distribution. The width of the distribution is determined by the creep rate c_{rate}, which regulates how far from its former value a gene can mutate. If the new mutated value would lie outside the search space, it is important to limit the value and instead set the new mutation value to the search space limit. The mutation process is illustrated in Figure 2.8.

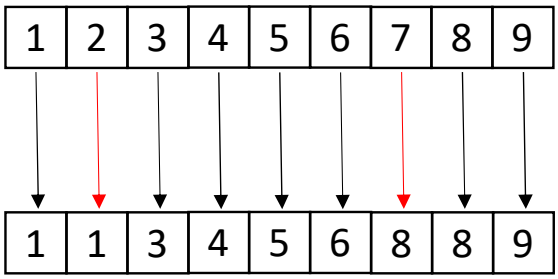


Figure 2.8: Illustration of mutation. The red arrows represent the genes being mutated. In this example the number of mutated genes are two.

Elitism

Even though the best individual is likely to be selected for the next generation, it is not a certainty. Even if the best individual is included in the next generation, it cannot be guaranteed that the individual will not undergo mutation. *Elitism* fixes this problem by always including the best individual in the next generation without any changes, ensuring that the best individual of each generation will not be lost.

Termination

The evaluation - evolution circle will continue until a termination criterion has been fulfilled, which for instance can be a certain number of generations. Finally, the individual with the highest fitness in the last generation is returned as the solution of the optimization.

3

Methods

This chapter introduces and explains all the methods used to arrive at the result. The overarching task of the method was to find the poses of one camera and one LiDAR mounted on the same car with overlapping FoV. The method can in short terms be described by the following steps:

1. Collection of simulated sensor data.
2. Conversion of camera images to 3D point clouds to be able to accurately combine and compare with the LiDAR data in a fitness model.
3. Definition of a fitness model.
4. Definition of an objective function to maximize for the highest fitness and thus also the best sensor poses.
5. Optimization.

The flow from beginning to end can also be seen in Figure 3.1.

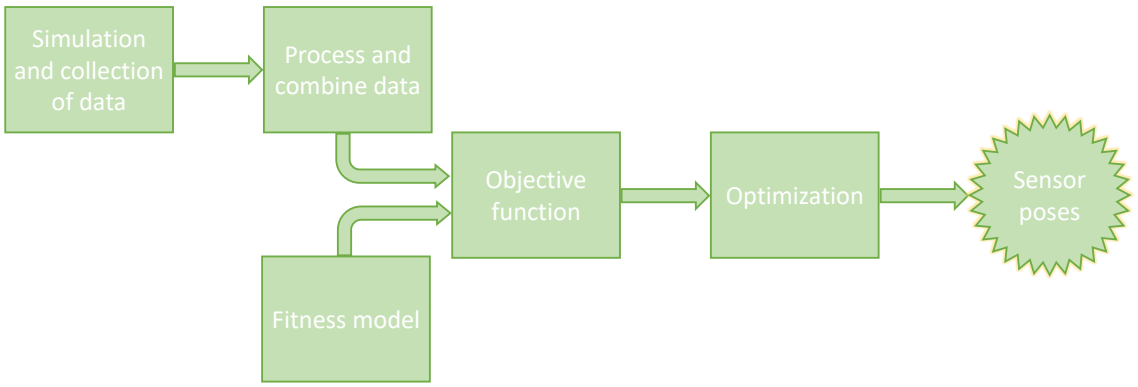


Figure 3.1: Overview of the method.

3.1 Simulation and collection of data

To simulate and collect useful data for the project, the CARLA simulator was used [20]. CARLA is a simulator built upon Unreal Engine for vehicle simulation and has a good interface for integrating different sensors. There were two main reasons why a simulator was needed and used, the first one being that it was fast and convenient for producing different scenarios and tests. Secondly, no real sensor data which satisfied our requirements were available.

The LiDAR type used was solid state. Based on the presented theory in Section 2.1, introducing calibration methods for solid state LiDARs will be of great importance in the near future. The FoV of the LIDAR used was 90 °. Since the only LiDAR model included in the CARLA library was a rotating LiDAR, a solid state one had to be emulated. It was possible to emulate solid state LiDAR characteristics in CARLA by using depth cameras. The same approach was used in [10] with great success. The range of the LiDAR was set to 50 meters. In addition to a LiDAR, a camera was used in the simulation. The camera was an RGB pinhole camera with a resolution of 1920×1080 and an FoV of 90 °. The camera had no distortion. The sensors' ground truth position in relation to the vehicle coordinate system can be seen in Table 3.1.

Table 3.1: Values of true calibration parameters for the LiDAR and the camera.

Sensor	**Translation (m)**			**Rotation (°)**		
	x	y	z	φ	θ	ψ
Camera	1	-1.6	2.8	0	0	-90
LiDAR	1.25	-1.6	2.8	0	0	-90

To evaluate the algorithm's robustness to disturbances while collecting data, we decided to simulate five different scenarios where we release some of the vehicle's DoF to account for human imperfection and uneven ground. The different scenarios are listed in Table 3.2. During the simulation a random number was drawn from a uniform distribution, then the vehicle's pose was altered accordingly. The distribution bounds were set to $[-2, 2]^\circ$ since we considered it to be a fair approximation of human error.

3.2 Process and combine data

After having collected the simulated data, the next step was to perform data fusion of the collected point clouds and images. In this thesis, it was desirable to keep the 3D properties of the LiDAR point cloud and therefore the camera data was converted to 3D point clouds with an SfM pipeline. To make the collected data comparable, some pre-processing was needed. The goal of this section is to explain how the 2D images collected by the camera were converted to 3D points, enabling

3D to 3D comparison of two point clouds. Figure 3.2 displays the steps presented in this section.

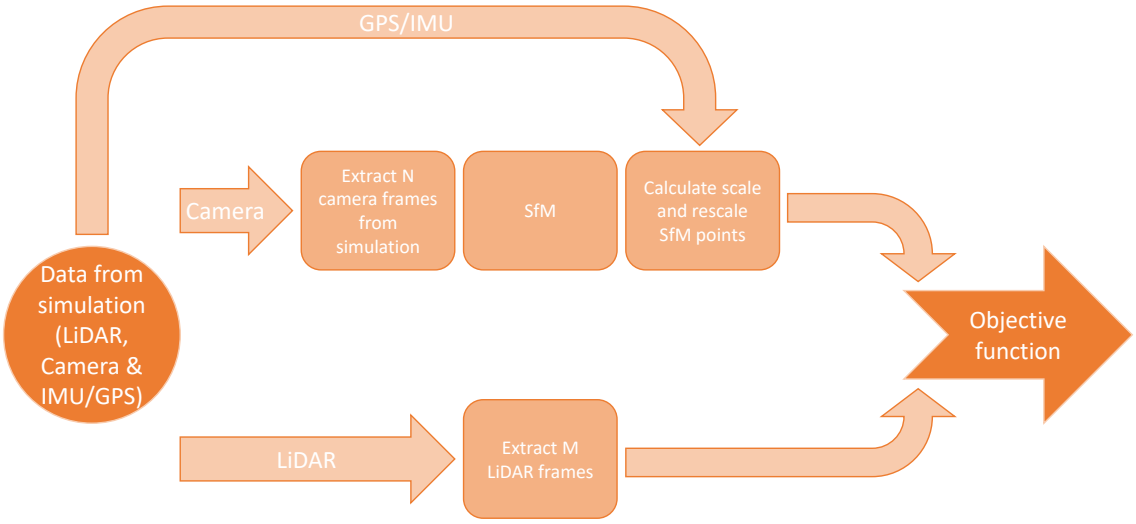


Figure 3.2: The steps performed in order to be able to fairly compare the data from the camera and LiDAR.

The camera images were converted to 3D point clouds by using SfM. Moreover, the number of camera frames and the scale factor to use in the SfM pipeline had to be chosen. The scale factor was needed to ensure that the SfM point cloud and the LiDAR point cloud had the same size. In the case of the LiDAR data, only the number of frames to use had to be decided.

Table 3.2: The different car test case scenarios used in the simulations. The rotational parameters are offsets from their respective ground truth values. The translation parameters (x,y,z) were not exposed to any external disturbances. Note that the parameters are related to the car and not the sensors.

Scenario	φ (°)	θ (°)	ψ (°)
Base	0	0	0
Roll	±2	0	0
Pitch	0	±2	0
Yaw	0	0	±2
Combined	±2	±2	±2

3.2.1 Conversion of camera images to 3D point clouds

SfM, described in Section 2.7, was used to convert the camera images to 3D points, see Figure 3.3 and Algorithm 1 for reference. After the 3D points had been reconstructed, the points behind the camera are identified and removed, by utilizing the cheirality condition mentioned in Section 2.7.2. Finally, an outlier filter is applied

based on the *z-score*, presented in Section 2.8, of each point's distance to the estimated camera position. A point is considered an outlier if it has a Z-score of $Z > 3$ or $Z < -3$, i.e. if a point lies beyond the $\pm 3\sigma$ interval of the normal distribution of the reconstructed 3D points.

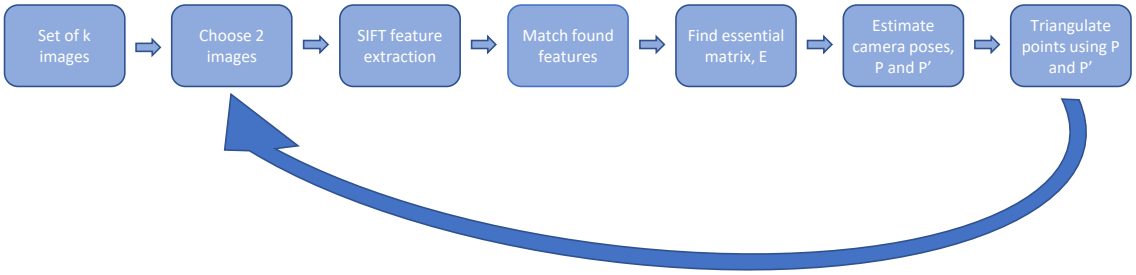


Figure 3.3: Flow chart displaying the most prominent steps of SfM

Algorithm 1: Structure from motion

Input: $imageSet \leftarrow$ Set of k images from simulation.
Output: $3Dpoints$, 3D point cloud constructed from the input image set
for $I \in imageSet$ **do**
  if *first iteration* **then**
    $SIFTpoints_1 \leftarrow$ Detect SIFT key feature points points and extract descriptors for first image I_1
  else
    $SIFTpoints_k \leftarrow$ Detect SIFT key feature points and descriptors for current image, I_k
    $matches \leftarrow$ Match features for current and previous image, I_k and I_{k-1}
    $E \leftarrow$ Estimate the essential matrix with 5-point algorithm
    $relPose \leftarrow$ Estimate relative pose of camera from E
    $points \leftarrow$ Find corresponding points across **all** previous matches
    $3Dpoints \leftarrow$ Triangulate($points$)
    $3Dpoints \leftarrow$ Remove points behind camera and outliers
  end
  $SIFTpoints_{k-1} \leftarrow SIFTpoints_k$
end

3.2.2 Scale estimation

Since SfM only reconstructs 3D points up to ratio and direction but not scale, a scale factor needs to be estimated to be able to properly compare the LiDAR point cloud with the one constructed by SfM.

As mentioned in Section 1.4 it is assumed that the vehicle is equipped with positioning sensors such as GPS/GNSS and IMU. The data collected by these sensors was used to estimate the car pose in the simulation, and thereafter calculate an estimation of the pose of the camera in world coordinates for each frame. Assuming that the resulting camera poses from SfM are correct in direction and ratio, they

can be used to calculate the scale factor between the point cloud generated by SfM and the point cloud from the LiDAR sensor. The scale factor between frame i and $i+1$ is calculated as:

$$s = \frac{\|\mathbf{t}_{i+1} - \mathbf{t}_i\|}{\|\mathbf{C}_{i+1} - \mathbf{C}_i\|}. \tag{3.1}$$

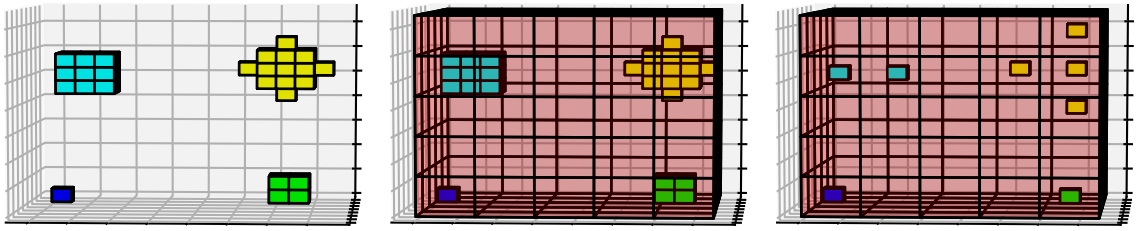
Where $\mathbf{C}_i$ is the position of the camera extracted from the camera matrix P from SfM. $\mathbf{t}_i$ is the position of the camera calculated from the car pose as

$$\mathbf{t}_i = \mathbf{t}_{c,i} + R_{c,i}\mathbf{v}. \tag{3.2}$$

$\mathbf{t}_{c,i}$ is the position of the car, $R_{c,i}$ is the rotation matrix of the car at frame i, and $\mathbf{v}$ is the translation between the origin of the car and the camera. Theoretically, the scale factor should not change between frames, but to ensure robustness the scale factor is chosen as the median of all calculated scale factors.

3.3 Fitness model

A fitness model was needed for measuring how well two point clouds overlap. The more overlap the better and thus a higher fitness score was given out. The fitness score was important for defining the objective function of the optimization described in Sections 3.4 and 3.5. The fitness model used was based on a voxel grid filter, i.e. a cubical grid filter that downsamples a point cloud. Each voxel that contained at least one 3D point was reduced to only having one 3D point in the center of the voxel, as shown in Figure 3.4.



(a) A simple point cloud before filtering.

(b) Applying the filter on the point cloud.

(c) Point cloud after filtering.

Figure 3.4: Illustration of applying a voxel grid filter to a simple point cloud. Images are taken from [10] with permission from the authors.

The downsampling feature of the voxel grid filter was utilized when defining the fitness score as

$$f_{score} = \text{points before filtering} - \text{points after filtering}. \tag{3.3}$$

It follows from equation (3.3) that a greater overlap between point clouds results in higher fitness score.

3.4 Objective function

Since the calibration parameters were to be found by optimization, an objective function was needed. The objective was to find the best possible overlap of the LiDAR and SfM point clouds. Therefore, the objective function had to be defined in a way that takes the overlap into consideration. The fitness model from Section 3.3 was used to define the objective function as

$$\max f(\text{point clouds}, \text{calibration parameters}). \tag{3.4}$$

The calibration parameters consisted of the pose of each sensor used, see Figure 3.5 for reference. The function f was defined as the fitness score f_{score} in equation (3.3), meaning the objective function maximizes the fitness score.

x_l	y_l	z_l	φ_l	θ_l	ψ_l	x_c	y_c	z_c	φ_c	θ_c	ψ_c

Figure 3.5: An array holding the calibration parameters of the sensors. The subscript l or c indicates a LiDAR or camera sensor respectively.

An important part of choosing the objective function was to choose the size of the voxels. The point cloud registered from the LiDAR was bounded by its lowest resolution and thus the voxel size had to be decided with this taken into consideration. Equation (3.5) states that at a distance r from the LiDAR, objects with a width greater than or equal to res_{low} will be detected, which is the lowest resolution of the LiDAR. α is the angular resolution of the LiDAR. The size of the voxels was set to res_{low}, to correctly filter the points far away.

$$res_{low} = 2\pi r \frac{\alpha}{360} \tag{3.5}$$

3.5 Optimization

The optimization problem was solved using a GA, which was described in Section 2.9. Each degree of freedom for each sensor was represented by a gene, illustrated in Figure 3.5, adding up to twelve genes per chromosome and individual. The SfM point cloud had to be re-scaled according to equation (3.1) before being used in the fitness function. The objective function used was the one described in Section 3.4, and after the GA had terminated the best individual of all generations was returned. The termination was decided to occur after a certain number of generations. The algorithm is described in Algorithm 2.

3.6 Sensor poses

After the optimization was terminated, the fittest individual's genes were returned as the sought-after sensor poses. The sensor poses were however only defined in

relation to each other since no information about the car, or how it was related to the sensors, was incorporated into the optimization. Thus, one sensor had to be constrained to its ground truth values, thereby enabling the comparison between the pose of the other sensor and its ground truth value.

Algorithm 2: GA optimization

Input: *point_cloud_list* ← All point clouds
Output: *overall_best*, Resulting best sensor poses.
Initialization: Generate N tuples of chromosomes - each of length kn, where k is the number of DoF for a sensor and n is the number of sensors used - representing an individual in the population.
$termination \leftarrow FALSE$
$n\_gen \leftarrow 0$ Create a counter holding the number of generations evolved.
while $\neg termination$ **do**
  for $i \in individuals$ **do**
    for j *in range(N)* **do**
      $transformed\_point\_clouds \leftarrow$ Transform point clouds according to the pose encoded in the current individual, i.
      $f_{score} \leftarrow$ evaluate($transformed\_point\_clouds$) `/* Save score */`
      .
    end
  end
  if $n\_gen \geq max\_gen$ **then**
    $termination \leftarrow TRUE$ `/*` *max_gen* `is the number of generations to run before termination */`
  else
    while $n\_iter < N$ **do**
      $j_1, j_2 \leftarrow$ Selection of two individuals from the evaluated population.
      $j_1, j_2 \leftarrow$ Crossover of the two chromosomes from the two individuals selected.
      $j_1, j_2 \leftarrow$ Mutation of the newly crossed chromosomes.
      $N_{new} \leftarrow$ Add generated individuals.
    end
  end
  $N \leftarrow N_{new}$
  $best\_individuals \leftarrow$ Save best individual.
  $overall\_best \leftarrow max(best\_individuals)$
  n_gen++
end

4

Results

This chapter presents the results achieved from implementing the method. The results are produced by simulating the five different cases presented in Section 3.1. The cases are compared to each other with respect to robustness, accuracy and their overall performance. First, the results of using the SfM pipeline to reconstruct camera images to 3D are presented. Thereafter, the results produced by applying the GA to the different cases will be presented along with the used tuning parameters.

4.1 Structure from motion pipeline

The SfM result was visually evaluated due to the lack of a proper statistical validation method. The point clouds generated with SfM were compared to an image of the scene to evaluate if a satisfying representation of the scene had been achieved. Another validation test for SfM was to compare the SfM point cloud and the LiDAR point cloud with respect to depth and scale. The relevant plots can be seen in Figures 4.1 and 4.2.

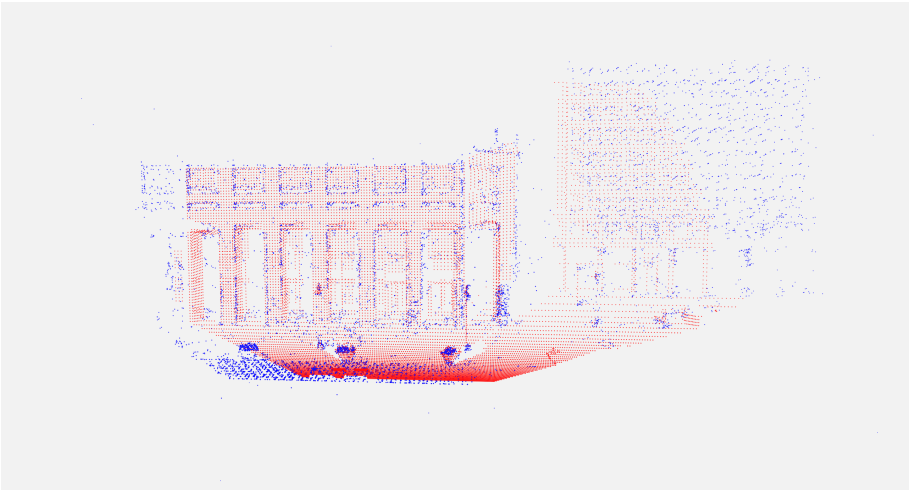
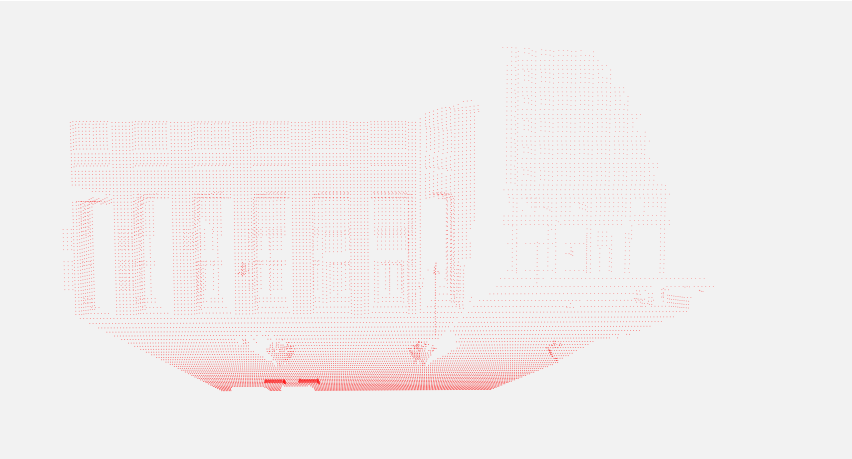
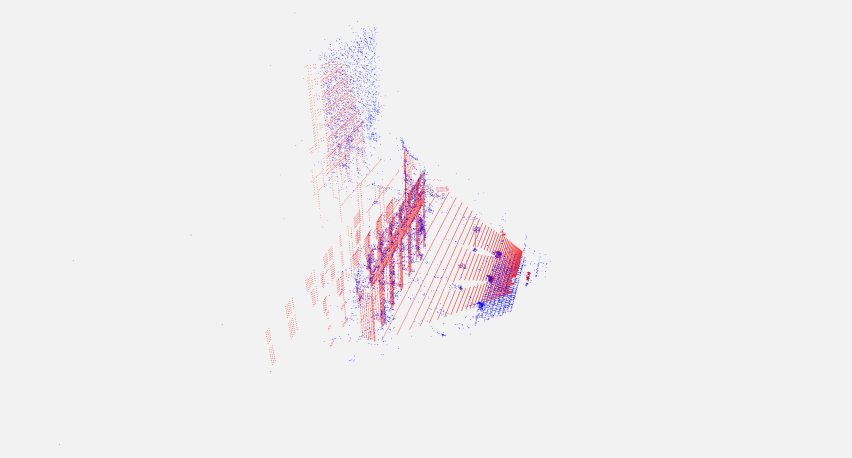


Figure 4.1: SfM point cloud (blue) and LiDAR point cloud (red) in the same plot, with correct overlap. It is visible that the proportions an depth of the SfM point cloud matches the LiDAR point cloud. The SfM point clouds consisted of a fewer number of points than the LiDAR point clouds.



(a) LiDAR point cloud only.



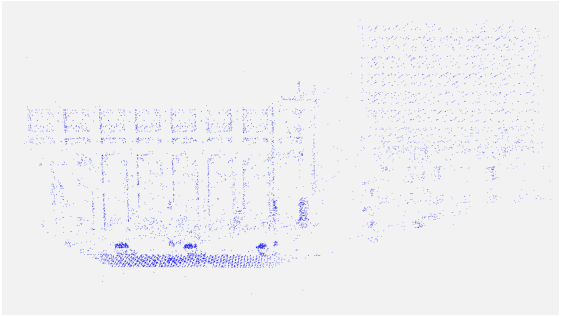
(b) SfM point cloud and LiDAR point viewed from the side.

Figure 4.2: LiDAR point cloud in (a) and SfM and LiDAR point cloud from a side view.

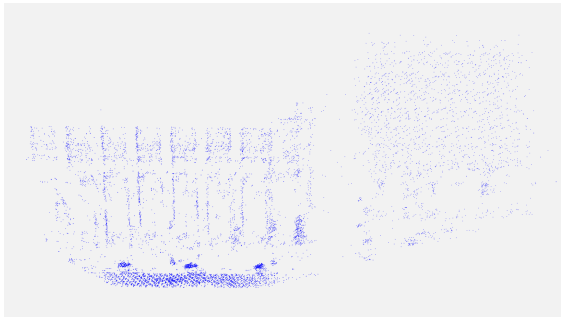
Each point cloud was generated from a set of ten images. The size of the image set was chosen both with consideration of the size of the LiDAR point cloud and to ensure that a large number of features were detected. Furthermore, only one LiDAR frame was used to not increase the complexity of the problem.

The resulting point clouds generated from SfM as well as an image of the scene are depicted in Figure 4.3. The base case was used as a baseline of what the SfM point cloud should look like and was compared with all other cases. The roll case showed many similarities to the base case, only a slight noise was the exception from the base case. However, a significant amount of noise was observed in the pitch case, resulting in a not as well reconstructed scene. The yaw case had a slightly lower level of noise than the pitch case. Subsequently, the combined case suffered from the bad characteristics of the pitch and yaw case, creating a noisy point cloud in the process. Moreover, all point clouds had a tendency to duplicate objects. The duplications were most prominent in the yaw, pitch and combined cases.

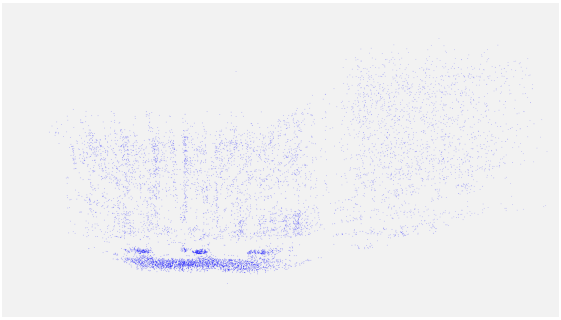
(a) Picture of the scene.



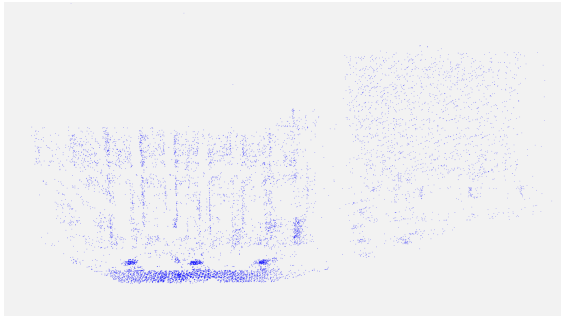
(b) Base case.



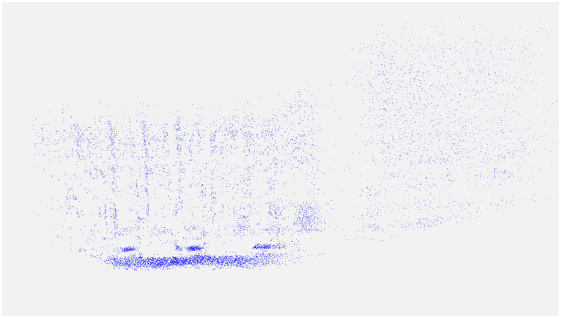
(c) Roll case.



(d) Pitch case.



(e) Yaw case.



(f) Combined case.

Figure 4.3: The result from SfM. The same scene was used to test all five cases, generating five different image sequences over the same scene. Each point cloud was then generated by the corresponding sequence of images.

4.2 GA parameters

The tuning of the GA proved to be hard, not only due to a large number of parameters but also because it was difficult to determine if the GA had converged or not. Without knowledge about the GA's convergence, we decided to base the tuning on the calibration result.

The algorithm consistently found better results while using tournament selection rather than roulette wheel selection, indicating that a better result correlates with the increase of exploration that tournament selection gives. A high tournament probability and low crossover probability proved to yield the best result. The tournament size was found to be best at around $\frac{1}{3}$ of the population size. We also

decided to use a decreasing creep rate enabling exploration at the beginning and exploitation in the later stages. Changing the mutation probability did not result in any drastic changes. We therefore decided to use the standard value of $1/n$, where n is the number of genes. A population size of 16 together with 4000 generations was used to achieve a total of 64000 evaluations. The rest of the tuning parameters are presented in Table 4.1.

Table 4.1: Chosen GA parameters

Parameter	Value
Population size	16
Generations	4000
Selection method	Tournament
Tournament prob.	0.7
Tournament size	5
Crossover prob.	0.25
Mutation prob.	$\frac{1}{n}$
Small creep rate pos.	0.2 m
Small creep rate ang.	5 °
Large creep rate pos.	0.4 m
Large creep rate ang.	10 °
Final creep rate pos.	0.02 m
Final creep rate ang.	0.3 °

4.3 GA calibration results

The five simulated scenes mentioned in Section 3.1 were tested with a small and large search space shown in Table 4.2. The GA optimization was decided to have produced a successful result if the translation distance in each direction was ≤ 0.1 m and the rotation about each axis $\leq 1.5°$ for both search spaces. The average error for each scenario is presented in Tables 4.3 and 4.4 for the small and large search space respectively. Table 4.5 shows the number of successful calibrations of each simulated case. Lastly, in Tables 4.6 and 4.7 the best individuals and their gene error of the different cases and search spaces are presented.

Table 4.2: Ranges of the two different search spaces used. Each range is defined as offsets from the ground truth values.

	Small search space	Large search space
Translation (m)	-0.2 to 0.2	-0.4 to 0.4
Rotation (°)	-5 to 5	-10 to 10

Table 4.3: Average gene error over 50 iterations for the small search space.

Case	x (m)	y (m)	z (m)	φ (°)	θ (°)	ψ (°)
Base	0.10271	0.10214	0.0917	0.5186	0.43337	0.36453
Roll	0.1072	0.06835	0.10608	0.62706	0.53484	0.637
Pitch	0.10232	0.19293	0.13557	0.9062	0.54228	0.43143
Yaw	0.11988	0.24481	0.11443	0.41859	0.95017	0.3637
Combined	0.11757	0.11608	0.14224	0.33499	1.26116	0.64808

Table 4.4: Average gene error over 50 iterations for the large search space.

Case	x (m)	y (m)	z (m)	φ (°)	θ (°)	ψ (°)
Base	0.15648	0.11302	0.16834	0.55368	0.56454	0.47714
Roll	0.20837	0.18403	0.23106	0.67023	0.93962	0.77826
Pitch	0.24845	0.35643	0.24677	0.96736	1.03174	2.82658
Yaw	0.19258	0.3064	0.17277	0.44325	0.64249	0.50601
Combined	0.19852	0.24436	0.21896	0.4234	1.42513	2.58035

Table 4.5: Number of successful calibrations for each search space and case. Both search spaces ran for 50 iterations each for every case.

Case	Success small search space	Success large search space
Base	8	3
Roll	9	3
Pitch	2	0
Yaw	0	0
Combined	5	2

Table 4.6: Best gene error of 50 iterations in the small search space. The values in parenthesis are the defined successful bounds for each parameter.

Case	x ($\pm$0.1 m)	y ($\pm$0.1 m)	z ($\pm$0.1 m)	φ ($\pm$1.5°)	θ ($\pm$1.5°)	ψ ($\pm$1.5°)	Success
Base	-0.05756	0.04017	-0.03869	-0.11712	0.04437	-0.05622	Yes
Roll	-0.00014	-0.03709	0.00521	0.69075	0.65235	-0.6388	Yes
Pitch	0.00161	0.07728	-0.15436	-0.14994	0.92051	0.04268	No
Yaw	-0.04373	0.11843	-0.08408	0.42493	0.04871	0.05618	No
Combined	-0.05228	-0.03663	0.02607	0.00133	1.03496	-0.82478	Yes

Table 4.7: Best gene error of 50 iterations in the large search space. The values in parenthesis are the defined successful bounds for each parameter.

Case	x (± 0.1 m)	y (± 0.1 m)	z (± 0.1 m)	φ ($\pm 1.5°$)	θ ($\pm 1.5°$)	ψ ($\pm 1.5°$)	Success
Base	0.01993	0.05776	0.02387	0.66528	0.64115	-0.27976	Yes
Roll	-0.01065	-0.05963	0.04416	0.93475	0.79486	-0.44219	Yes
Pitch	-0.20421	0.09324	-0.04029	1.1714	-0.26305	0.52156	No
Yaw	-0.12259	0.16861	-0.1152	0.01276	-0.21716	0.15345	No
Combined	-0.06394	0.02919	-0.01791	0.06172	1.44898	-0.89724	Yes

4.4 Defect objective function

For an objective function to work as intended, the ground truth parameters must generate the highest possible fitness score. However, the proposed objective function could generate higher fitness scores than the ground truth, which can be seen in Figure 4.4. This phenomenon caused the algorithm to never converge at ground truth, as it should, and made it hard to determine if convergence had been achieved.

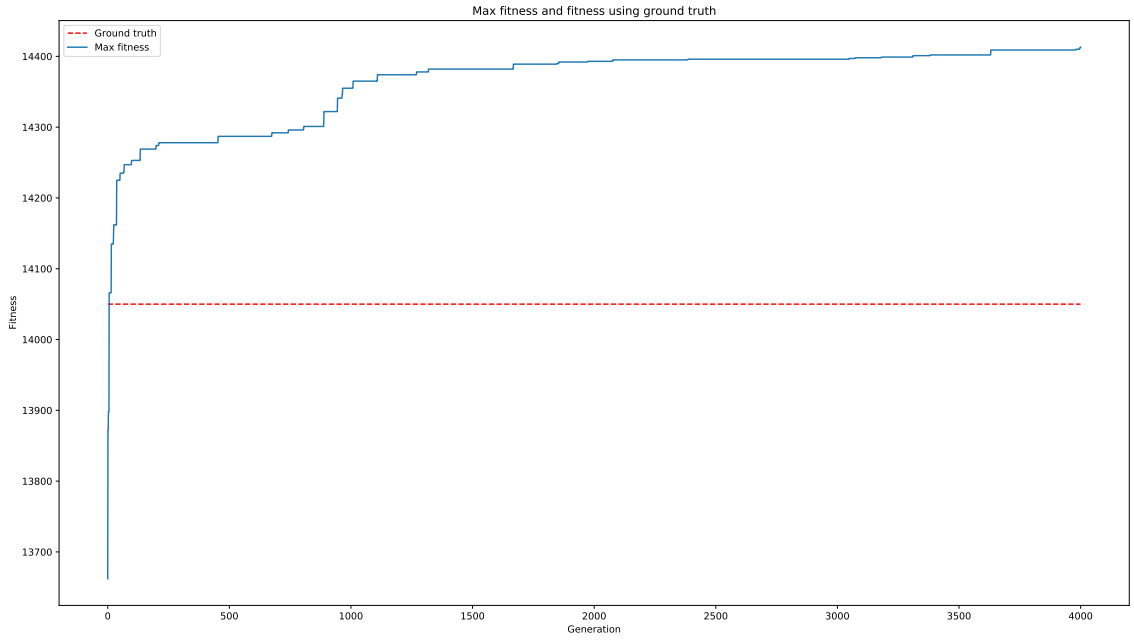


Figure 4.4: Example of the GA finding higher fitness than ground truth. The red line is ground truth and the blue is maximum fitness found by the GA. The vertical axis is fitness and the horizontal axis is number of generations.

5

Discussion

In this chapter the results collected are discussed based upon the research questions stated in Section 1.4. The methods used to achieve the results presented are also discussed and their performance evaluated.

5.1 Structure from motion generated point clouds

One of the main difficulties when designing the LiDAR-camera calibration algorithm was how to represent the camera data in 3D. As stated in the first research question, this thesis has investigated how an SfM pipeline could contribute to constructing a 3D point cloud from camera data. As presented in Section 4.1, Figure 4.1, Figure 4.2b and Figure 4.3, the algorithm was successful to the degree that it was possible to distinguish that the SfM point cloud was a reconstruction of the image data. However, the algorithm worked better for some of the cases.

The noise or duplication phenomenon characterizing the pitch, yaw and combined case appears to be a direct consequence of SfM not being able to align points when the camera images were converted to 3D point clouds. The phenomenon can also be seen in the roll case, but not to the same extent as in the pitch, yaw and combined cases. Theoretically, SfM should not be affected by the camera movement during the simulation. However, the camera pose estimation used in SfM or the triangulation might have been flawed, which would explain the duplication of points observed in some of the cases.

The accuracy and refinement of the constructed point cloud also depend on the images used. One interesting aspect is if the results would have been better if the camera captured images more frequently and in higher resolution, thus generating both more and better images and in turn generating point clouds with less noise. Additionally, an image of higher resolution would increase the number of features that can be detected, hence increasing the chance of denser point clouds.

The second part of the first research question refers to comparability between the two different types of point clouds. First and foremost, the two types of point clouds were based on different concepts. The LiDAR point cloud was based on raw data and was, therefore, more reliable, while the camera point cloud was constructed in

several steps and was, therefore, more sensitive to errors. Since the point clouds were generated by different methods, it was not straightforward if a generated SfM point cloud were of satisfactory standards in relation to the LiDAR point cloud. Nonetheless, a fewer number of points were a pervading trait of the SfM point cloud. As explained in Section 2.7, the number of points used in SfM was based on the quantity of found SIFT features. In earlier works, mainly [10], the point clouds used were only produced by LiDARs, leading to that the same object was represented by similar clusters of LiDAR points in all point clouds intercepting the object. Due to SfM point clouds being generated from features, the same object did not necessarily consist of a similar 3D point cluster as in the LiDAR data. It was not a guarantee that the object would be visible if it did not contain enough features. Contradictory, some objects or features appearing in the SfM point cloud did not appear in the LiDAR point cloud.

Finally, implementing Bundle Adjustment (BA) might help improve the results of the SfM point clouds by reducing noise. In this thesis, BA was not used, but it was tested. However, the test proved to introduce more noise rather than compensate for it.

5.2 Objective function

The objective function did not work as intended shown by the result in Section 4.4. We knew from [10] that the objective function was not perfect. However, for LiDAR-LiDAR calibration, the objective function was good enough to get a satisfactory calibration. One explanation for why the objective function performed worse for LiDAR-camera calibration is that the point clouds are fundamentally different. In LiDAR-LiDAR calibration, the point clouds are built the same way and should in theory be able to overlap perfectly. However, since LiDAR point clouds are more evenly distributed while SfM point clouds are generated from clusters in feature-heavy areas, an overlap as precise as in the LiDAR-LiDAR case cannot necessarily be expected. This difference in performance could indicate that the objective function works better with similar types of point clouds or denser point clouds.

Since the fitness is dependent on a voxel grid filter, it can be biased towards voxels containing many points, no matter if the points are in the correct place or not. Another way to view it is that the algorithm will favor empty voxels since an empty voxel will decrease the number of points after downsampling, thus increasing the score. For LiDAR-LiDAR calibration, the highest number of empty voxels should be when the point clouds overlap perfectly, thus making ground truth have the highest fitness. However, since the LiDAR point cloud and SfM point cloud are different, ground truth may not be the pose that maximizes the number of empty voxels and thus the fitness. The result of using this type of filtering as a fitness function was that the algorithm did not converge toward ground truth, which explains the insufficient quality of the results.

To better be able to determine if a LiDAR-camera calibration is feasible with the

method we have proposed, a new objective function needs to be formulated. The new objective function would preferably match regions of points rather than specific points since the point clouds are built differently. One idea was to base a new fitness function on plane surfaces and use RANSAC with plane fitting. However, the new approach would be heavily reliant on planar features such as buildings or the ground, meaning that it would not work in a non-planar environment, such as a forest.

5.3 Overall performance

The results of the GA were not as accurate as anticipated. There are nonetheless some interesting aspects to discuss. The results presented in Section 4.3 and the spread of the boxplots in Appendix A clearly showed that a smaller search space seemed to be profitable, which could be explained by the increased probability to sample the sensor poses inside the successful bounds. The larger search space allowed the GA to explore further away from the ground truth, hence decreasing the probability to find successful parameters and thus needing more generations to find optimal values.

Regarding the errors in parameter estimations, one interesting aspect was the difference in error between the estimated rotation and translation. The angular parameters could be successfully estimated for all five test cases if the small search space were used, see Table 4.3. Moreover, the error distribution over 50 iterations was inside the successful bounds with a great margin for the base and roll cases, which can be seen in the boxplots in Appendix A and in Table 4.3. The algorithm can thus be used if the aim is to calibrate the angular parameters of a LiDAR-camera pair. One explanation for the algorithm being able to successfully estimate rotation is that rotational offsets give rise to larger distributions of points in the voxel grid filter. Thus, there will be fewer empty voxels, hence decreasing the fitness. Even small angular offsets can cause large distances between points that should overlap. Positional offsets, however, may not be of as much importance, which coincides with the results of the positional estimation not being as adequate as the rotational. If points in the SfM point cloud are still close to coinciding with other parts of the same object, it could still generate a high fitness, since many points may still overlap.

Since the angular parameters could be successfully estimated in all five cases with the small search space, the algorithm can be considered robust to rotational disturbances applied to the vehicle when estimating the sensors' roll, pitch and yaw angles. Disturbances in the roll angle of the vehicle gave the best results, having no disturbance at all excluded. The algorithm is thus regarded to be the most robust to vehicle disturbances in roll.

6

Conclusion

This thesis has investigated a concept for automatic LiDAR-camera co-calibration. One LiDAR and one camera were used. The objective was to find the extrinsic parameters of both sensors. The project consisted of two main parts - the first was to convert camera images to 3D. The second part was to align 3D point clouds from the camera and LiDAR and optimize for the best result, thus retrieving the sought-after sensor poses.

Camera images were converted from 2D to 3D using a structure from motion pipeline. Five different disturbance scenarios were tested. Overall, the quality of the generated point clouds varied in resolution and alignment. The base case, where no rotational disturbance was introduced, produced the best point cloud in the sense of refinement and accuracy. As the camera and LiDAR point clouds were not generated by the same method, different objects in the point clouds were not represented by the same number of points. Some objects were only visible in the LiDAR point cloud, others only in the SfM point cloud, leading to difficulties in the aspect of comparability.

A genetic algorithm was used to optimize the objective function and thus calibrate the sensors. However, the result did not coincide with what we considered a satisfactory calibration result. The primary reason for the somewhat poor results was the flawed objective function. The objective function could achieve better fitness than ground truth, making the algorithm search for a solution that was not ground truth, which proved problematic. However, the calibration results of the angular parameters showed robustness to rotational disturbances in almost all cases, indicating that the method still works for calibrating roll, pitch and yaw. Together with the use of the small search space, the calibration method can be used to thoroughly calibrate the roll, pitch and yaw parameters for a LiDAR-camera pair.

6.1 Future work

The proposed method is not perfect, and to be able to function properly some future work is needed. Most importantly, the objective function and possibly the fitness function should be revised and improved. Furthermore, densification of point clouds should be investigated, to see if denser SfM point clouds will increase the number of

successful calibrations. Another recommendation is to more thoroughly investigate if Bundle Adjustment can improve point cloud alignment and noise reduction.

Ultimately, additional sensors should be introduced, along with possibilities to constrain the sensors to the vehicle frame already in the optimization, not only in relation to each other. Also, a bigger variety of simulation parameters and extended testing of different scenarios should be included in further development of the method.

Bibliography

[1] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, Jun. 2006, ISSN: 1070-9932. DOI: 10.1109/MRA.2006.1638022.

[2] X. Liu, C. Yuan, and F. Zhang, "Fast and Accurate Extrinsic Calibration for Multiple LiDARs and Cameras," Sep. 2021.

[3] Z. Bai, G. Jiang, and A. Xu, "LiDAR-Camera Calibration Using Line Correspondences," *Sensors*, vol. 20, no. 21, p. 6319, Nov. 2020, ISSN: 1424-8220. DOI: 10.3390/s20216319.

[4] L. Zhou, Z. Li, and M. Kaess, "Automatic Extrinsic Calibration of a Camera and a 3D LiDAR Using Line and Plane Correspondences," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Oct. 2018, pp. 5562–5569, ISBN: 978-1-5386-8094-0. DOI: 10.1109/IROS.2018.8593660.

[5] A. Dhall, K. Chelani, V. Radhakrishnan, and K. M. Krishna, "LiDAR-Camera Calibration using 3D-3D Point correspondences," May 2017.

[6] S. A. Rodriguez F., V. Fremont, and P. Bonnifait, "Extrinsic calibration between a multi-layer lidar and a camera," in *2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, IEEE, Aug. 2008, pp. 214–219, ISBN: 978-1-4244-2143-5. DOI: 10.1109/MFI.2008.4648067.

[7] B. Nagy and C. Benedek, "On-the-Fly Camera and Lidar Calibration," *Remote Sensing*, vol. 12, no. 7, p. 1137, Apr. 2020, ISSN: 2072-4292. DOI: 10.3390/rs12071137.

[8] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski, "Building Rome in a day," in *2009 IEEE 12th International Conference on Computer Vision*, IEEE, Sep. 2009, pp. 72–79, ISBN: 978-1-4244-4420-5. DOI: 10.1109/ICCV.2009.5459148.

[9] L. Brown and G. Lindberg, "Extrinsic calibration of multiple 2D Lidars using a Genetic Algorithm," M.S. thesis, Chalmers University of Technology, Gotheburg, 2019.

[10] T. Langfjord Nordgård and O. Ravndal Skjølingstad, "Calibration of 3D lidars A fully automatic and robust method for calibrating multiple 3D lidars using only point cloud data," M.S. thesis, Chalmers University of Technology, Gothenburg, 2020.

[11] M. Khader and S. Cherian, "An Introduction to Automotive LIDAR," Dallas, Tech. Rep., 2020. [Online]. Available: https://www.ti.com/lit/pdf/slyy150.

[12] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," Tech. Rep., 2004.

[13] M. A. Fischler and R. C. Bolles, "Random sample consensus," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981, ISSN: 0001-0782. DOI: 10.1145/358669.358692.

[14] R. Szeliski, *Computer Vision*. Cham: Springer International Publishing, 2022, ISBN: 978-3-030-34371-2. DOI: 10.1007/978-3-030-34372-9.

[15] R. Klette, *Concise Computer Vision*. London: Springer London, 2014, ISBN: 978-1-4471-6319-0. DOI: 10.1007/978-1-4471-6320-6.

[16] H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Nature*, vol. 293, no. 5828, pp. 133–135, Sep. 1981, ISSN: 0028-0836. DOI: 10.1038/293133a0.

[17] R. I. Hartley and P. Sturm, "Triangulation," *Computer Vision and Image Understanding*, vol. 68, no. 2, pp. 146–157, Nov. 1997, ISSN: 10773142. DOI: 10.1006/cviu.1997.0547.

[18] R. I. Hartley, "Chirality," *International Journal of Computer Vision*, vol. 26, no. 1, pp. 41–61, 1998, ISSN: 09205691. DOI: 10.1023/A:1007984508483.

[19] M. Wahde, *Biologically Inspired Optimization Methods - an introduction*. Southampton, UK: WIT Press, 2008.

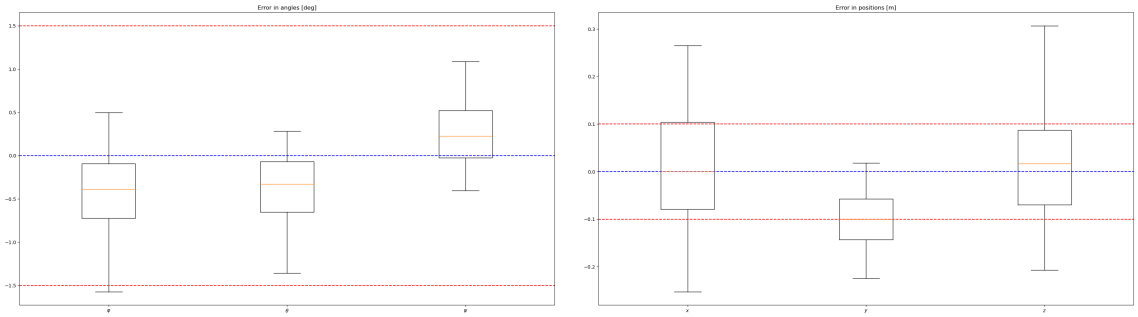
[20] CARLA Team, *CARLA*, 2022. [Online]. Available: https://carla.org/.

A

Boxplots

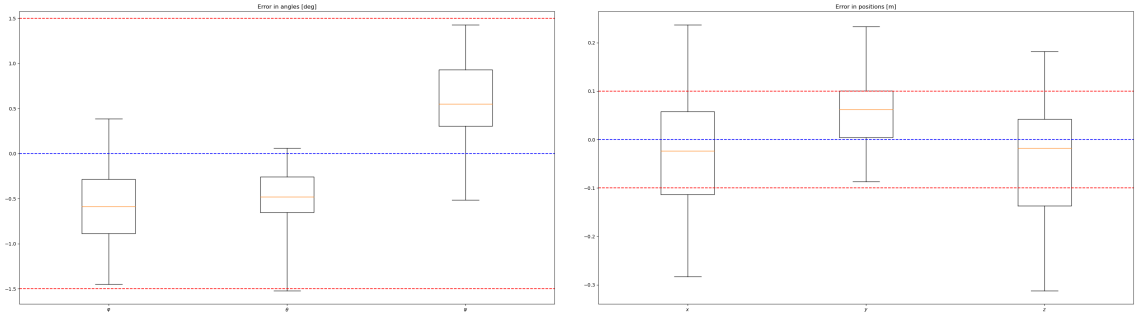
The error of each simulated case are presented below in boxplots. In each boxplot the area between the red lines are the successful bounds as in offset from the true value. The blue line at zero is the desired error.

A.1 Small solution space



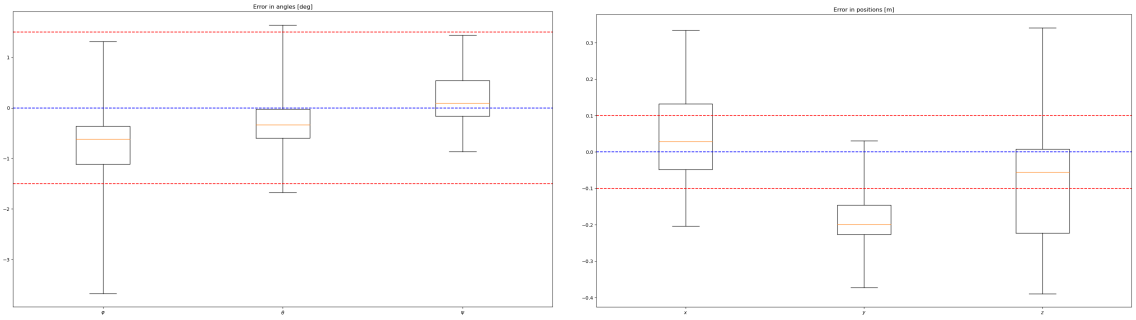
(a) Angular errors. **(b)** Positional errors.

Figure A.1: Base case with small solution space.



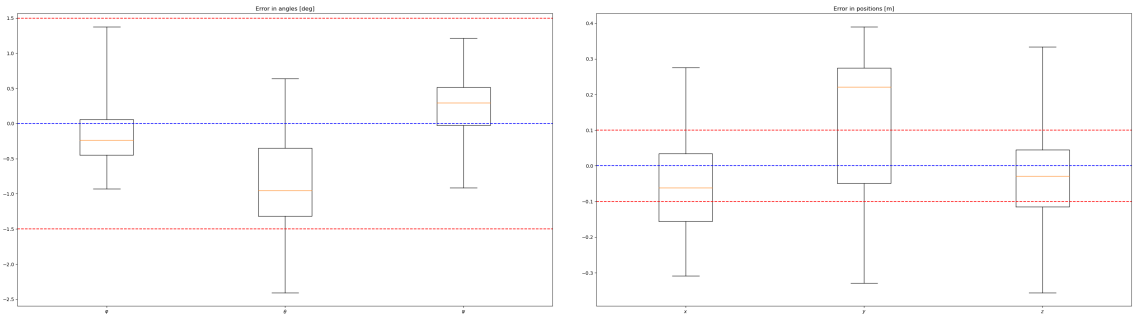
(a) Angular errors. **(b)** Positional errors.

Figure A.2: Roll case with small solution space.



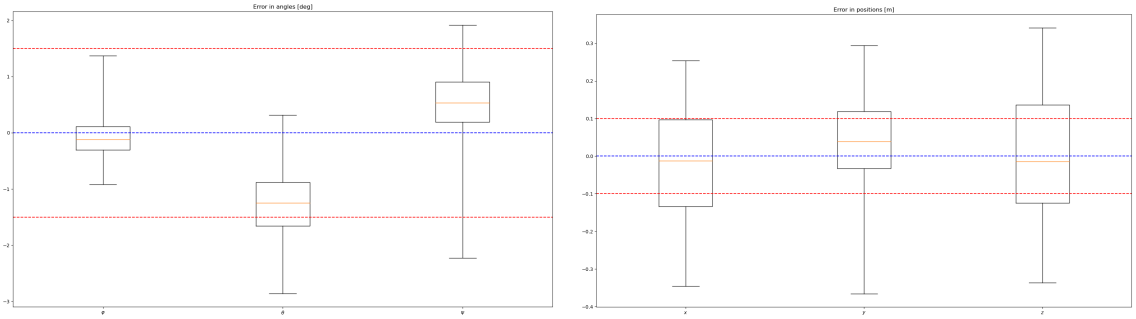
(a) Angular errors. **(b)** Positional errors.

Figure A.3: Pitch case with small solution space.



(a) Angular errors. **(b)** Positional errors.

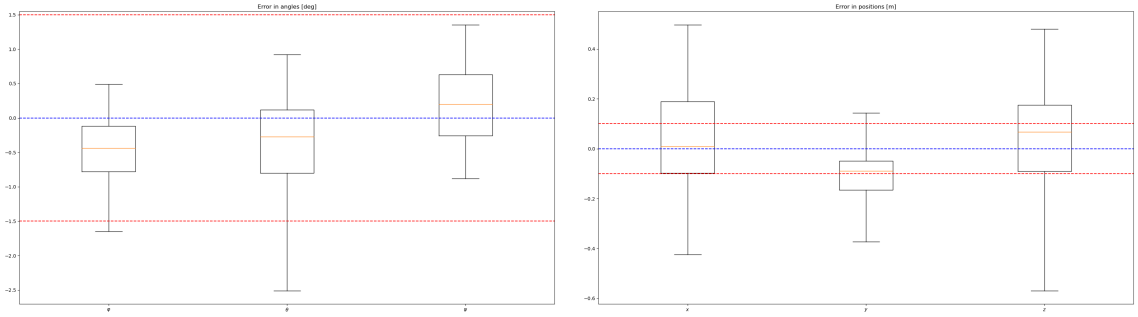
Figure A.4: Yaw case with small solution space.



(a) Angular errors. **(b)** Positional errors.

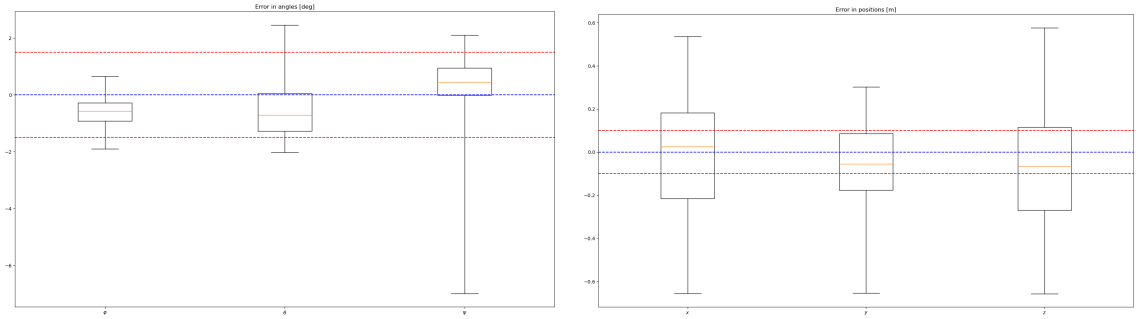
Figure A.5: Combined case with small solution space.

A.2 Large solution space



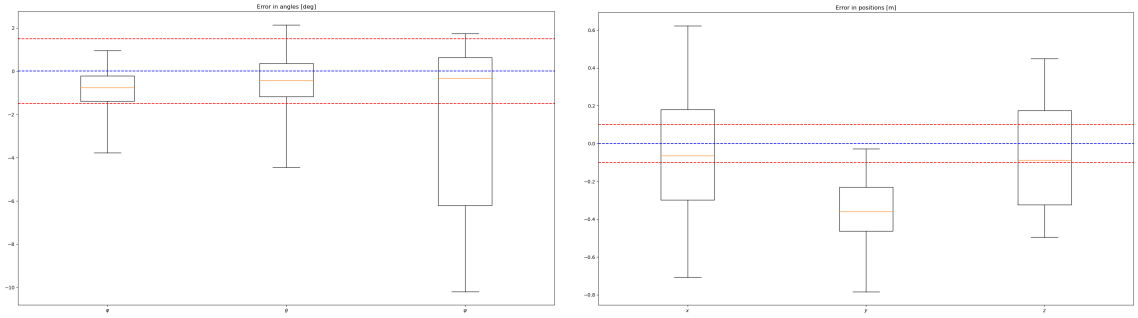
(a) Angular errors. **(b)** Positional errors.

Figure A.6: Base case with large solution space.



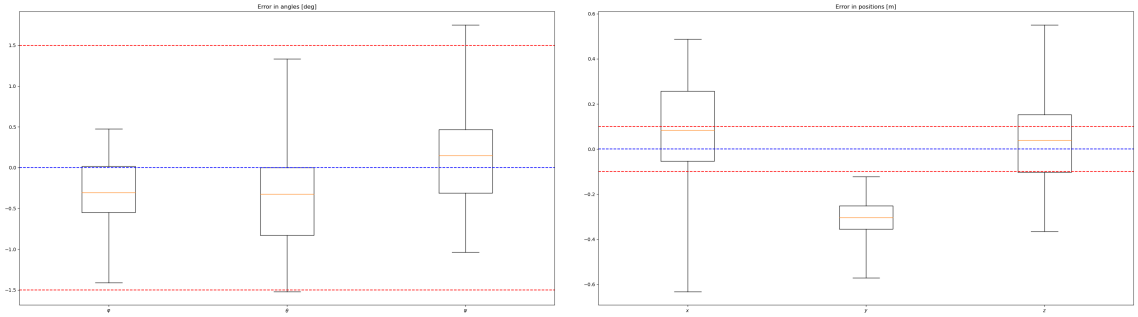
(a) Angular errors. **(b)** Positional errors.

Figure A.7: Roll case with large solution space.



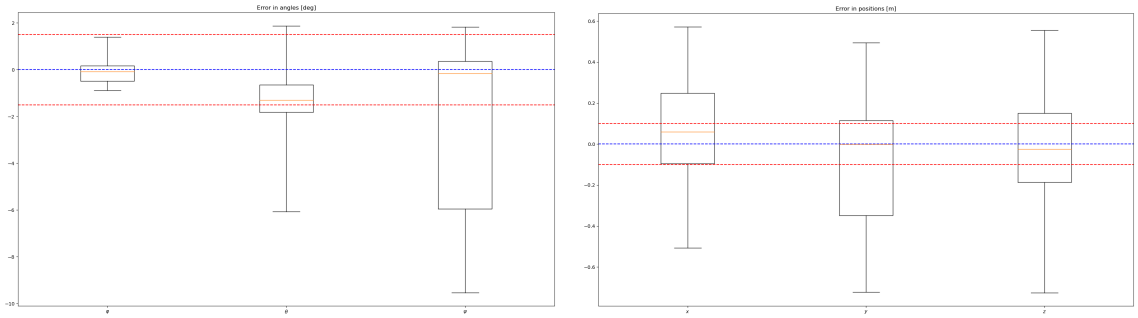
(a) Angular errors. **(b)** Positional errors.

Figure A.8: Pitch case with large solution space.



(a) Angular errors. **(b)** Positional errors.

Figure A.9: Yaw case with large solution space.



(a) Angular errors. **(b)** Positional errors.

Figure A.10: Combined case with large solution space.

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY