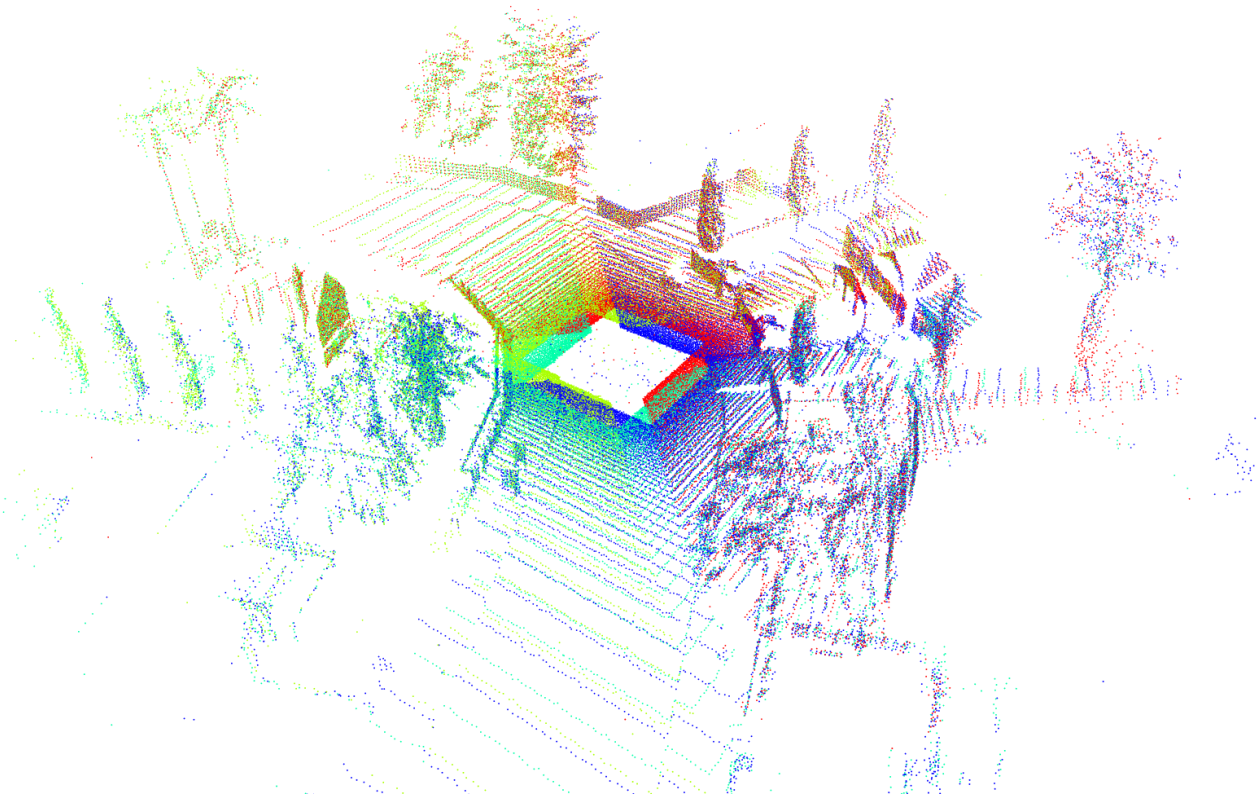
CHALMERS
UNIVERSITY OF TECHNOLOGY



Calibration of 3D lidars

A fully automatic and robust method for calibrating multiple 3D lidars using only point cloud data

Master's thesis in Systems, Control and Mechatronics

TORGEIR LANGFJORD NORDGÅRD
OLAV RAVNDAL SKJØLINGSTAD

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020
www.chalmers.se

Master's thesis 2020:17

Calibration of 3D lidars

A fully automatic and robust method for calibrating multiple 3D lidars using only point cloud data

TORGEIR LANGFJORD NORDGÅRD
OLAV RAVNDAL SKJØLINGSTAD

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2020

Calibration of 3D lidars
A fully automatic and robust method for calibrating
multiple 3D lidars using only point cloud data
TORGEIR LANGFJORD NORDGÅRD
OLAV RAVNDAL SKJØLINGSTAD

Advisor: Lars Brown, CPAC Systems AB
Examiner: Peter Forsberg, Department of Mechanics and Maritime Sciences

Master's Thesis 2020:17
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: Visualization of a point cloud produced by four lidars calibrated using the proposed algorithm. The measurements from the lidars are represented by separate colors.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Calibration of 3D lidars
A fully automatic and robust method for calibrating multiple 3D lidars using only point cloud data

TORGEIR LANGFJORD NORDGÅRD
OLAV RAVNDAL SKJØLINGSTAD
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

In order to use lidars for perception in autonomous vehicles, they must be properly calibrated. Commonly used techniques for automatic calibration, such as iterative closest point, often requires an accurate guess of the calibration parameters, which is challenging to obtain. Additionally, these techniques are often dependent on feature-extraction or designated calibration environments, which are highly application-specific. We propose an alternative calibration algorithm for an arbitrary number of lidars based on particle swarm optimization. Using our method, accurate calibration parameters can be produced from extremely rough initial guesses, without the aforementioned application-specific limitations. When tested on synthetic data, the algorithm is shown to be superior to conventional methods. Additionally, a method for allowing calibration during vehicle movement is explored and proposed.

Keywords: lidar calibration, point clouds, stochastic optimization, voxel grid filter

Acknowledgements

We would like to thank both our advisor Lars Brown and our examiner Peter Forsberg for all the help and interesting discussions they provided during the semester. We would also like to thank the people at CPAC Systems AB for welcoming us into their offices, and helping us when needed.
Finally, we would like to thank SARS-CoV-2 for helping us avoid unnecessary distractions during the writing of this thesis.

Torgeir Langfjord Nordgård & Olav Ravndal Skjølingstad
Gothenburg, June 2020

Thesis advisor: Lars Brown
Thesis examiner: Peter Forsberg

Contents

Acronyms

FL front left. 3, 10, 24, 32, 33
FOV field of view. 3, 10, 38, 39
FR front right. 3, 10, 24, 33

GA genetic algorithm. 6–8, 15, 18, 19, 24, 27–32, 36, 38, 39

ICP iterative closest point. 4–8, 24, 27, 31, 32, 36

PSO particle swarm optimization. 7, 8, 19, 20, 24, 27–33, 36, 37, 39

RANSAC random sample consensus. 5
RL rear left. 3, 10, 24, 33
RMS root mean square. 29, 31, 32
RR rear right. 3, 10, 24, 33

SIFT scale-invariant feature transform. 6, 7

1

Introduction

For the past few years, the development of autonomous vehicles has been booming. The industry is currently exploring usage of autonomous operation, due to the many advantages of automating tasks that has previously been done by humans. This is especially true for environments that are inhospitable and dangerous.

For autonomous vehicles to operate, awareness of the environment is needed. For this, a multitude of sensors are used. Usually, some combination of ranging sensors, imaging sensors, inertia sensors and GNSS receivers are fused to obtain a robust and accurate perception of reality. The knowledge of the orientations and the positions of these sensors in relation to each other and the vehicle influences how well the sensor data can be fused. Therefore, knowledge of the sensors' extrinsic parameters are paramount for the development of usable autonomous platforms.

CPAC Systems, a developer of localization and control systems, was one of the main developers in the Brønnøy project in Norway. The Brønnøy project utilizes multiple autonomous Volvo trucks for transporting limestone from an open pit mine to a nearby port. The trucks uses four 2D lidars as a part of the navigation system. An illustration of their pose on the vehicle can be seen in Figure 1.1.

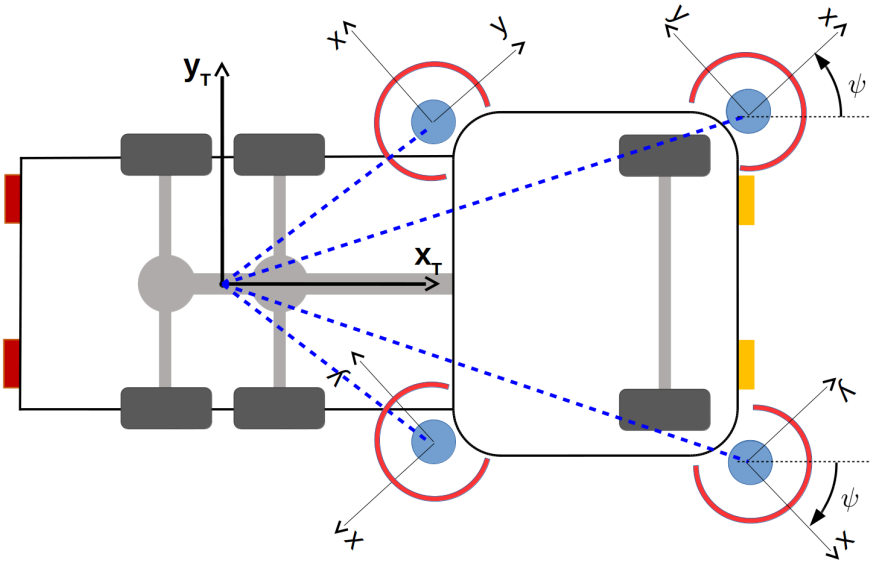


Figure 1.1: Overview of four lidars and their position on the vehicle. Reprinted from [1] and modified with the authors' permission.

1.1 Lidar variants

A lidar is an active sensor that emits light in the form of a laser and measures the time it takes for that light to be reflected back to the device. These measurements can then be used to create a point cloud of the environment. If a lidar has lasers and detectors at one height capturing a single plane, a 2D point cloud can be built. If a lidar has multiple lasers and detectors capturing planes at different heights, a 3D point cloud can be built. There are multiple ways to construct a lidar sensor, but for the purposes of this thesis two main types are relevant; rotating and solid-state.

In rotating lidars, the laser and the detector rotate several revolutions per second to cover a wide field of view. By measuring reflection times distances can be calculated, which together with the angle of the laser at the time of measurement can be used to build a point cloud of the environment. Because the rotation of a lidar is not instantaneous, neither is the data capture. As a consequence, any motion of a rotating lidar during the sweep of the light beam will distort the collected data. So, unless the vehicle is standing still, the motion of the vehicle must be taken into account when calculating the point cloud.

Solid-state lidars, on the other hand, contains no moving parts. Instead of rotating the lasers and detectors to achieve horizontal resolution, the lidar is built by an array of detectors both horizontally and vertically. This approach allows all measurements to be collected simultaneously, which solves the issue of the vehicle moving during capture, prevalent in rotating lidars. When solid-state lidars reach high production volumes, they will be cheaper and more durable than their rotating counterparts [2]. Therefore, an industry-wide shift to solid-state lidars is inevitable, and algorithms for calibrating these lidars will be needed.

1.2 Calibration

Lidars produce a point cloud with respect to their own coordinate system. By estimating the lidars' translation and orientation in relation to each other, the individual point clouds can be merged into a single point cloud as if obtained from a single lidar. The lidars' position and orientation in the vehicle's coordinate system will be referred to as *calibration parameters*. The process of estimating the calibration parameters is known as calibration.

Since the environment in the mines at Brønnøy is quite rough, the lidars are shaken badly and hit at times, altering their original positions. Thus, the lidars should be calibrated regularly before driving sessions and ideally during driving as well. Manual calibration is difficult and time-consuming, so an automatic calibration algorithm is required for both practical reasons and for safety. Furthermore, if the calibration is to be done while the vehicle is moving, the process would have to be automatic.

As additional sensors will add more points of failure, calibration using only point clouds produced by the lidars is desirable. To fulfill this desire, overlap in the

collected point clouds is needed, which in turn requires overlap in the lidars' field of view (FOV). Common features between the point clouds can then be used to infer the poses of the lidars. In point clouds produced by 2D lidars, common features only exist if they depict the same plane, which only happens if the lidars have the same height, roll and pitch. As a consequence, any calibration algorithm using 2D point clouds is limited to calibration of x, y and yaw.

Calibration using 3D point clouds does not have such limitations, as the additional dimension makes the determination of complete poses possible. The richer information available in 3D point clouds also makes them desirable for use in perception. As mentioned earlier, a shift to solid-state lidars is inevitable, so only 3D solid-state lidars are considered in this thesis.

1.3 Aim

The aim of this thesis is to propose and implement a fully automatic calibration algorithm. The calibration algorithm shall determine the poses of four 3D lidars given a rough initial estimation of the calibration parameters, without the use of designated calibration environments. Additionally, only the lidars' point clouds shall be used for calibration. The poses of the lidars' coordinate system relative to the vehicle's coordinate system are illustrated in Figure 1.2.

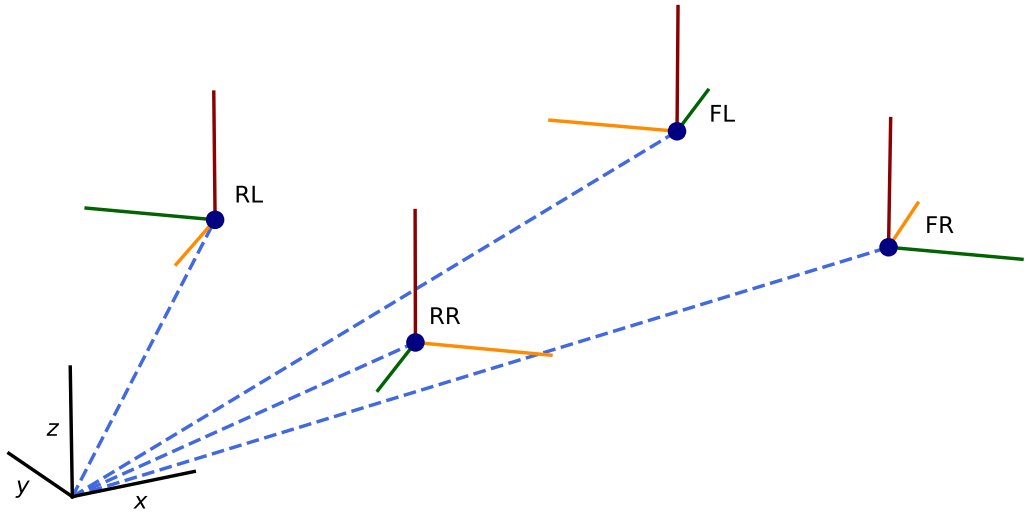


Figure 1.2: Overview of the four lidars and their translations (dashed blue lines) and orientations (green, yellow, and red lines) relative to the vehicle's coordinate system (black lines). The lidars are referred to as front left (FL), front right (FR), rear right (RR) and rear left (RL), which reflects their positions on the vehicle.

A secondary aim is to implement an adaptive version of the automatic calibration algorithm. Adaptive refers to detecting changes in lidar poses during vehicle movement, followed by recalibrating the lidars on the fly. The lidars should thereby stay calibrated at all times, increasing the quality of the vehicles' perception system.

1.4 Related work

The underlying challenges with extrinsic calibration of lidars can be divided into two parts. The first step is to find robust correspondences in the lidar data. With this at hand, the correspondences' overlap can be maximized by changing the pose of the lidars using an optimization method. Ideally, the resulting lidar poses produce a merged point cloud, where all common features between the lidars perfectly overlap.

These challenges are quite common within the fields of perception and positioning [3, 4, 5], so a multitude of solutions and approaches exist [6, Ch. 5]. Those considered most relevant for this thesis are the iterative closest point (ICP) algorithm and biologically inspired optimization methods. For the sake of completeness, feature-based calibration methods will also be explored. In this section these approaches will be presented in more detail, starting with ICP.

1.4.1 Iterative closest point

The iterative closest point (ICP) algorithm offers one of the simplest solutions to calibration of lidars, as it does not require any feature extraction algorithm. As expressed in [7], the main idea behind ICP is to align one point cloud, P, to another point cloud, Q. This process can be viewed as finding the transformation $T^{Q \leftarrow P}$ that best maps the points in P to the corresponding points in Q, and can be expressed more formally as solving

$$T^{Q \leftarrow P} = \arg\min_{T} \text{error}(T(P), Q) \tag{1.1}$$

where $\text{error}(T(P), Q)$ represents the error function. This error function is obtained by first extracting a set of matching points between the two points clouds, defined as

$$M = \{(p, q) : p \in P, q \in Q\} \tag{1.2}$$

In most cases, this set is found by doing a nearest neighbor search between the point clouds. To increase the robustness of the algorithm, a common approach is to associate a weight to each match. This set of weights is formally defined as

$$W = \{w(p, q) : \forall (p, q) \in M\} \tag{1.3}$$

Subsequently, the error function can be defined as the sum of the weighted distance between the matched points as

$$\text{error}(P, Q) = \sum_{p,q \in M} w(p, q) d(p, q) \tag{1.4}$$

where $d(p, q)$ is the distance between point p and q. Unfortunately, using this error function in equation (1.1) would only yield a correct transformation if used with an ideal matching function. To overcome this limitation, ICP iteratively forms

the set M before minimizing the error function. The result is a sequence of N transformations, the product of which produces

$$\hat{T}^{Q \leftarrow P} = \left(\prod_{i=N}^{1} T_i \right) T_{init} \tag{1.5}$$

where $\hat{T}^{Q \leftarrow P}$ is an estimation of the transformation between P and Q, and T_{init} is an initial guess of this transformation.

After the ICP algorithm is finished, the relative transformation must be geometrically constrained to the vehicle frame. The most common approach is to manually determine the pose of the lidar that produced Q, which is difficult to do accurately. Determining the orientation is especially challenging as one must measure the angles of the lidars' laser beams. Any errors in these manual measurements will propagate to the calibration of the lidar that obtained P.

As ICP only calibrates one pair of lidars at a time, calibrating more than two lidars together requires running ICP on all of them in a pairwise fashion. Consequently, any errors made by the algorithm when calibrating one lidar pair will propagate to the next pair. Another issue with ICP is the sensitivity to the initial guess, T_{init}. The accuracy of T_{init} greatly affects the outcome of the matching function, which in turn affects the final transformation.

1.4.2 Feature-based calibration

Feature-based calibration is the process of extracting interesting regions in the data from one sensor, referred to as *features*, and matching these with features from other sensors yielding *correspondences*. A calibrated system is obtained by minimizing the distance between these correspondences, which can be done using, for instance, a modified version of ICP. Hence, feature-based calibration can be viewed as simply changing the definition of the matching function in equation (1.2), which is investigated further in [8].

When dealing with lidars, a common approach is to introduce features manually by adding known geometric shapes in a calibration environment. In [9, 10, 11] various combinations of cameras and lidars are successfully calibrated by creating scenes with checkerboards and boxes. As the shape of the objects are known, shared features such as corners and planes can be extracted from the point cloud and the image. The 3D-features are projected into the image, from which a relative pose can be estimated by maximizing their overlap with the image features. In these publications, the optimization process is done using random sample consensus (RANSAC), which is a paradigm for robust model fitting [12]. In [9, 10] the solution from RANSAC is refined using modified versions of gradient descent to further improve the accuracy.

In [13] a set of 3D lidars are calibrated by including a moving sphere in the environment. By estimating the center of the sphere along its successive positions a trajectory of the ball can be extracted from each lidar. These correspondences are

matched between the lidars using an approach based on singular value decomposition [14], yielding their relative poses. However, estimating the sphere's center requires a certain amount of measurements of the sphere itself. Furthermore, all lidars must detect the sphere at all times, which greatly complicates the setup and reduces the flexibility of this approach.

To account for these drawbacks, descriptor-based feature extraction techniques have been developed. These techniques aim at finding unique, invariant features on naturally occurring objects in a scene, more formally referred to as *descriptors*. Descriptor-based feature extraction has long been used on 2D images with methods ranging from the hand-crafted SIFT-descriptor [15], to descriptors learned by neural networks [16, 17].

In recent years 3D counterparts of these 2D descriptors have been developed. Such 3D descriptors are extensively compared and evaluated in [18, 19]. In [18] different 3D descriptors are tested by merging point clouds using ICP. Their results indicate that the accuracy of the merged point cloud is dependent on the choice of descriptor. An unsuitable descriptor cannot extract enough unique features from the input data for ICP to work. In [19] the descriptors' quality and repeatability are reviewed. Here, the authors conclude that these factors are largely dependent on the nature of the input data, and that the best choice of descriptor is application dependent.

1.4.3 Biologically inspired optimization methods

An alternative to using ICP as the optimization method is to utilize approaches found within the family of biologically inspired optimization methods. As described in [20, Ch. 1], these methods are stochastic optimization algorithms inspired by biological concepts such as evolution and swarming behavior. A shared requirement is the need to describe the quality of a solution by associating it with a numerical value. This value is either maximized or minimized.

One evolutionary algorithm is the genetic algorithm (GA). This method uses the concept of fitness as a numerical description of the quality of a solution, so an optimal solution should result in the highest fitness. In [21] this idea is explored in the context of calibrating the yaw angles of four 3D lidars. The fitness of the calibration parameters is calculated by projecting the 3D points onto the x-y plane. This plane is divided into grids, and the fitness is proportional to the number of projected points within each cell of the grid. By using a GA to evolve the calibration parameters, the yaw angles are successfully found.

The most relevant previous work for this thesis is [1]. It uses data collected from the system presented in the introduction, and has an approach based on a GA. In this work, an automatic calibration method for four 2D lidars is developed for x, y and yaw. Here the fitness of the calibration parameters is given by downsampling the merged point cloud and calculating the number of removed points. The more points removed by downsampling the higher the fitness of the calibration. Additionally, the approach has the advantage of not calibrating lidars pairwise, it calibrates all the lidars at once. However, the solution form the GA only includes the relative transfor-

mation between the lidars. To obtain the calibration parameters, the solution must be geometrically constrained to the vehicle's coordinate system. Here, the lidars are geometrically constrained using assumptions about symmetry and knowledge of the distance from the vehicle's origin to the center point between the rear lidars. When the assumptions hold, most of the constraints will be zero, which greatly simplifies the manual measurements needed to geometrically constrain the lidars.

The authors of [1] also propose that future work in the area should explore the use of particle swarm optimization (PSO), which is another biologically inspired optimization algorithm. This resonates well with [20, Ch. 1], which states that some stochastic optimization algorithms outperforms others on specific classes of problems.

1.4.4 Conclusions on related work

Determining the extrinsic parameters of sensors is a well-researched topic. As discussed in the previous sections, the most common approaches for lidars are based on combinations of feature extraction and deterministic optimization methods, such as different variations of ICP. However, the use of feature extraction based on calibration objects requires that certain information must be present in, or even introduced to, the data. Therefore, these approaches are often limited to designated calibration environments, which is not desirable. Furthermore, the use of descriptor-based feature extractors, such as 3D-SIFT on point clouds, are application dependent and not believed to provide enough descriptors to produce a robust calibration loop.

The simple approach found in ICP without any feature-extraction is appealing. Unfortunately, ICP does suffers from an accumulating error when used with more than two lidars, as they are calibrated in pairs. It is also sensitive to the initial guess as well as error introduced during the manual calibration of the reference. Constraining the system via the reference is another drawback, as the rough environment at Brønnøy might cause its pose to drift, subsequently affecting the entire calibration. For these reasons, ICP is believed to be an unsuitable approach to the problem at hand.

Another calibration method that uses point clouds without any feature extraction is presented in [1]. Here, a genetic algorithm is utilized to successfully calibrate four 2D lidars. The calibration is performed on all four lidars simultaneously, thereby solving the issues with pairwise calibration, making it more robust than ICP. It also introduces a symmetry-based constraining method, which is considered more robust and easier to determine than that used with ICP. A similar concept for evaluating the quality of the calibration parameters could possibly be used for calibration of 3D lidars, and should be further researched.

To the authors knowledge the use of particle swarm optimization in calibration applications is fairly unexplored. The advantages of using a GA also applies to PSO, as the concept of downsampling a merged point cloud to determine calibration quality can be reused. This makes it an interesting subject to look further into.

1.5 Research questions and limitations

Based on the aim and related work, this thesis will focus on answering the following research questions:

1. How can an objective function be defined for calibrating 3D lidars? How does the choice of objective function influence the calibration algorithm's ability to consistently converge to accurate calibration parameters?
2. How do the GA and PSO compare with respect to robustness, i.e. initial estimation, accuracy, consistency and sensor noise? How do they compare to ICP?
3. Which, if any, modifications and additions must be made to the proposed calibration algorithm to make it adaptive? How can an algorithm be defined to detect changes in lidar poses? What are the advantages and challenges with using such an algorithm?

To simplify the work, the algorithms will be tested on data produced in a simulator using four high resolution solid-state lidars. When calibration is performed using point cloud data exclusively, the lidar poses can only be determined relative to each other. Therefore, to determine the calibration parameters relative to the vehicle, some relations between the lidar arrangement and the vehicle's coordinate system will be assumed known.

2

Methods

This chapter aims at providing detailed descriptions of data collection, the objective functions and algorithms used for calibration, how relative poses are geometrically constrained to the vehicle, and a proposed method for adaptive calibration.

2.1 Data collection

The CARLA simulator is used for data collection in this project. CARLA is an open-source driving simulator built on top of the graphics engine Unreal Engine 4 and is purpose-made for autonomous vehicle research [22]. It includes support for collecting data from simulated sensors, such as lidars, depth cameras and IMUs. A screenshot from the simulation environment can be seen in Figure 2.1. By using a simulator we set the ground truth ourselves, and can thus evaluate the correctness of the calibration parameters objectively. Using a simulator is also much faster and more convenient than collecting data from embedded hardware, allowing us to spend more time at developing and evaluating the algorithms.

Figure 2.1: The CARLA environment. This scene is the basis for other figures in this section.

The built-in lidar sensor in CARLA is based on a rotating lidar. Therefore, a

custom sensor is constructed to model a solid-state lidar. The built-in depth camera sensor is used as a basis for the custom lidar model. In CARLA, the depth camera sensor extracts an image from the environment. Each pixel encodes the distance, λ, from the image plane to captured objects. The depth information is stored as bytes in the pixels' color channels and must be decoded, which is explained in the CARLA documentation. To create a point cloud from the depth image computer vision techniques are utilized. Firstly, the image coordinates must be converted to homogeneous coordinates. Secondly, each homogeneous coordinate, $\mathbf{x}$, must be normalized, which is done using the depth camera sensor's calibration matrix

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

where

$$f = \frac{w}{2\tan(\pi \cdot \text{FOV}/360)}, \quad p_x = \frac{w}{2}, \quad p_y = \frac{h}{2} \tag{2.2}$$

Here, w and h are the width and height of the depth image in pixels, and FOV is the field of view. Finally, each 3D point, $\mathbf{X}$, in the point cloud can be computed using the pinhole camera model as

$$\mathbf{X} = K^{-1}\mathbf{x}\lambda \tag{2.3}$$

Each solid-state lidar is comprised of multiple depth cameras with a FOV of at most 90°. These depth cameras are placed in the same point and rotated relative to each other. The reason for using multiple depth cameras instead of a single depth camera with a large FOV is the distortion introduced by the latter. The amount of depth cameras and their rotations are determined by the desired FOV of the solid-state lidar. For example, a solid-state lidar with a FOV of 270° uses three depth cameras, each with a FOV of 90°, rotated 90° relative to each other. The resolution of the lidars are determined by the resolution of the depth cameras, and the range is determined by removing points that are further away than the desired range.

In this project, the lidars are based on the current sensors used in the Brønnøy project. They will have a FOV of 270° and be positioned on the vehicle according to the values in Table 2.1. These parameters results in an overlap of 180° between each lidar. Additionally, the lidars will have a range of 50 meters and an angular resolution of 0.5°. Figure 2.2 shows the result of collecting data using the proposed lidar model.

Table 2.1: Values of true calibration parameters for each lidar.

Lidar	**Translation [m]**			**Rotation [deg]**		
	x	y	z	φ	θ	ψ
FL	2.0	1.5	2.8	0	0	45
FR	2.0	-1.5	2.8	0	0	-45
RR	-2.0	-1.5	2.8	0	0	-135
RL	-2.0	1.5	2.8	0	0	135

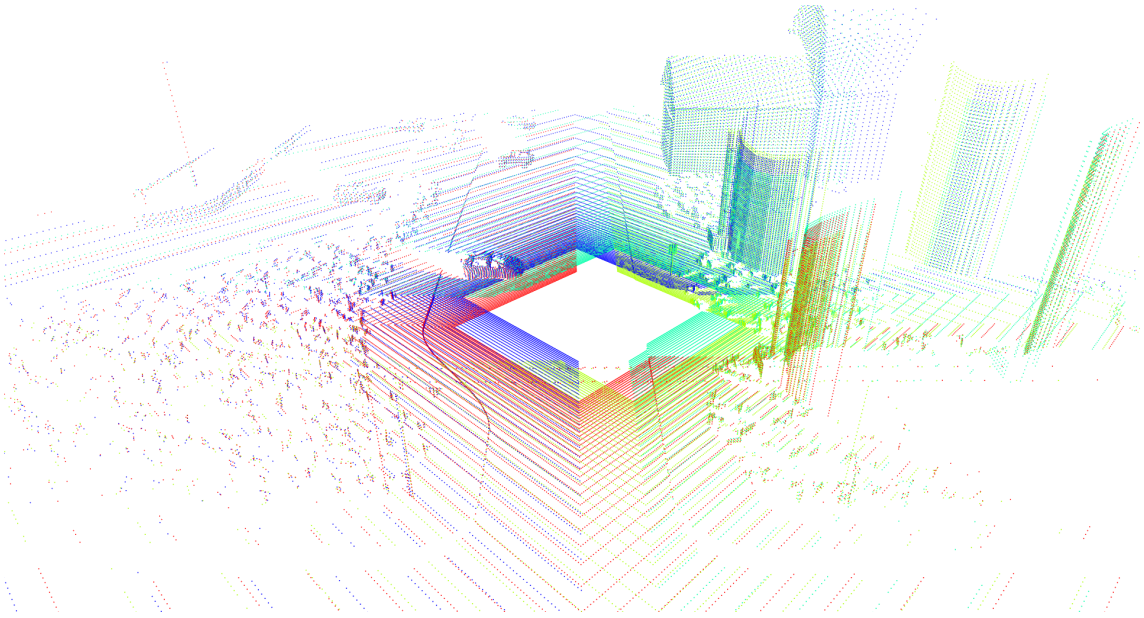


Figure 2.2: A point cloud extracted from the scene depicted in Figure 2.1 using the solid-state lidar model. Each lidar's point cloud is shown with a separate color.

2.1.1 Data modification

The calibration algorithm should eventually be used in the real world, not just a simulated environment. Therefore, the synthetic data is modified to emulate data collected from a real lidar. In the real world, noise is introduced in every part of the data collection chain, from quantization due to the discrete nature of sensors, to electric noise which is inherent in all electronic components.

The noise is modeled as

$$\begin{aligned} \hat{x} &= x + \mathcal{N}(0, 0.1) \\ \hat{y} &= y + \mathcal{N}(0, 0.1) \\ \hat{z} &= z + \mathcal{N}(0, 0.1) \end{aligned} \tag{2.4}$$

where $\hat{x}$, $\hat{y}$ and $\hat{z}$ are the noisy "measurements", x, y and z are the ground truth coordinates, and $\mathcal{N}(0, 0.1)$ is white Gaussian noise with a standard deviation of 0.1.

To further enhance the realism of the data, outliers are added as well. Since measuring the time of flight of light emitted by lidars has extreme requirements of precision, errors in distance are deemed the most probable source of outliers. Therefore, the outliers are generated as follows

$$\begin{aligned} d &= \sqrt{\hat{x}^2 + \hat{y}^2 + \hat{z}^2} \\ r &\sim \mathcal{N}(0, 0.1d) \end{aligned} \tag{2.5}$$

which are used to shift the distance of the noisy measurement computed in (2.4) as

$$\begin{aligned} \hat{x} &= \hat{x} \cdot r \\ \hat{y} &= \hat{y} \cdot r \\ \hat{z} &= \hat{z} \cdot r \end{aligned} \tag{2.6}$$

Note that r is sampled once and multiplied with all three components, as opposed to equation (2.4) where the noise is sampled individually for x, y and z. Using this model we increased the amount of outliers until it looked reasonable, which was found to be 1% of all points. An illustration of a point cloud that is modified using the described noise and outlier model can be seen in Figure 2.3.

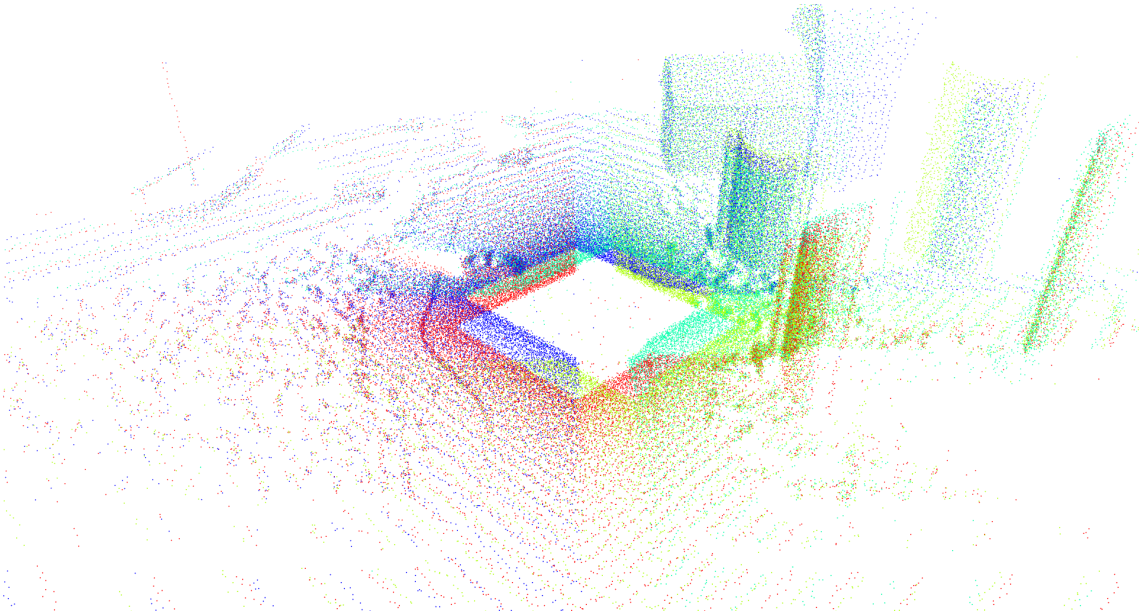


Figure 2.3: The point cloud from Figure 2.2 with added noise.

2.2 Objective function

Since stochastic optimization methods will be used, an objective function must be defined. The objective is to calibrate lidars, so the objective function must associate a numerical value to the quality of the calibration parameters. This number will be referred to as the *score*. The inputs to the objective function are the calibration parameters from four lidars, illustrated in Figure 2.4, and the point clouds from the four lidars, which results in the objective function

$$\text{score} = f(\text{calibration parameters}, \text{point clouds}) \tag{2.7}$$

Since the objective function only takes the calibration parameters and point clouds as input, it can only be used to find relative transformations between the lidars.

Assuming that there are overlapping objects in the lidar scans, a good calibration should result in a larger amount of overlapping points than a bad calibration. Therefore, the objective function should calculate the amount of overlap between the four point clouds after the calibration has been applied. This calculation is done by first transforming each lidar's point cloud using their respective calibration parameters. Subsequently, the four transformed point clouds are merged into a single point cloud. Determining the amount of overlap in this merged point cloud is the task of the objective function. More specifically, it should associate a high score to parameters

that produces a point cloud with a large number of overlapping points, and a low score when the opposite is true. The amount of overlap between the point clouds can be determined numerically using a voxel grid filter.

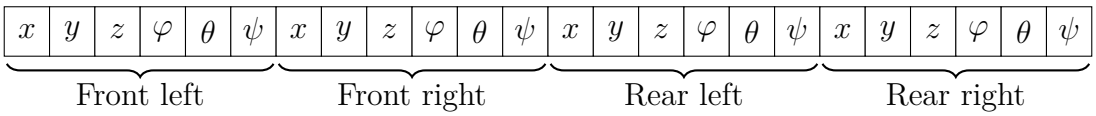


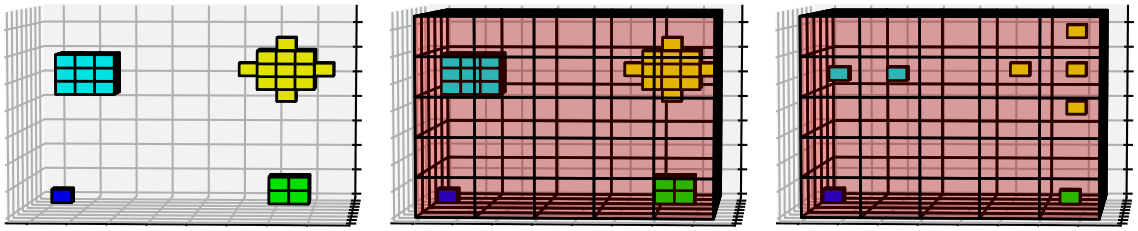
Figure 2.4: A tuple containing the calibration parameters of four lidars.

2.2.1 Voxel grid filter

A voxel grid filter downsamples a point cloud by applying a cubical 3D grid on the data. If any points lie within the bounds of a grid, they will be replaced by a single point at the center of the voxel. Consequently, the voxel grid filter will leave a point cloud with fewer points if overlap is present, which is exploited to produce the score as

$$\text{score} = \text{points before filter} - \text{points after filter} \tag{2.8}$$

An illustration of how the filter works can be seen in Figure 2.5.



(a) Before filtering. **(b)** Applying the filter. **(c)** After filtering.

Figure 2.5: Illustration of applying the voxel grid filter to a simple point cloud.

Determining the size of the voxel grid is important, as a point cloud gets sparser farther away from the lidar. If the resolution of the voxel grid is too fine, objects far away will not be matched correctly as the voxel size will be smaller than the point cloud resolution. Two ways to set the voxel size are explored in this thesis, which gives rise to two objective functions.

2.2.2 Uniform objective function

A simple approach is to use a uniform voxel filter with a voxel size corresponding to the lowest resolution of the point cloud. This method will be referred to as the *uniform objective function*. The voxel size is determined by the angular resolution α and range r of a lidar, according to

$$s = 2\pi r \frac{\alpha}{360} \tag{2.9}$$

An illustration of voxel size computation can be seen in Figure 2.6. The uniform voxel grid filter has the disadvantage of removing the finer details in objects at close range. The result of applying a uniform filter to the point cloud in Figure 2.2 can be seen in Figure 2.7.

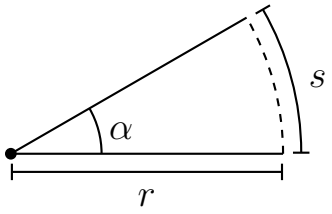


Figure 2.6: The voxel size s is determined by the lidars' angular resolution α and range r.

2.2.3 Distance varying objective function

Another approach is to use a *distance varying objective function*. In this method, the size s of a voxel is varied according to its Euclidean distance d from the lidar, which can be written as

$$s(d) = 2\pi d\frac{\alpha}{360} \tag{2.10}$$

Unlike the uniform objective function the data collected from the lidars is better utilized, as the voxel size is varied according to the resolution of the point cloud. Thus, finer details are kept, which we expect to yield more accurate calibration.

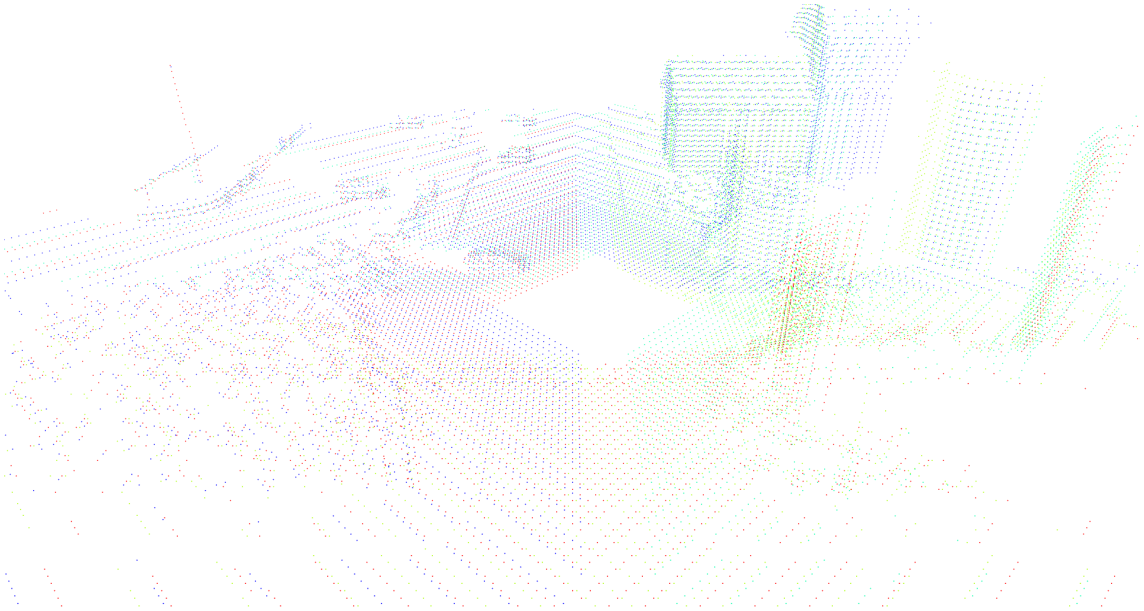


Figure 2.7: The point cloud from Figure 2.2 after applying the uniform voxel grid filter.

2.3 Genetic algorithm

The first stochastic optimization method explored in this thesis is the genetic algorithm (GA). GAs are a subset of evolutionary algorithms, which are optimization methods inspired by natural selection. The work and theory on GAs in this thesis is largely based on [20, Ch. 3].

Fundamentally, GAs generate a population of individuals, each with its own chromosome. The chromosome contains a set of genes, each of which represents a parameter in the search space of an objective function. In our case, the parameters of interest are the translation and rotation of the four lidars in relation to the vehicle. The chromosome is defined as the tuple shown in Figure 2.4, and each entry represents a single gene. More formally, the value of gene g in chromosome C, will be denoted

$$\text{value} = C(g) \tag{2.11}$$

The main pipeline of the genetic algorithm can be summarized as initialization, evaluation, and evolution of the population. The last two steps are repeated until termination. The number of repetitions is usually referred to as generations, but we will instead refer to it as iterations for consistency. An in depth explanation of the different steps in the GA is outlined in the following sections.

2.3.1 Initial population

The first step in a genetic algorithm is to initialize the population. The number of individuals in the algorithm is called the population size, and balances computational complexity and optimization performance. That being said, increasing the population size is often a case of diminishing returns [20, Ch. 6], and the optimal value is different for most problems.

Given a population size, each individual's chromosome must be initialized, which will be done by sampling their genes from the uniform distribution

$$C(g) \sim U(lb(g), ub(g)) \tag{2.12}$$

where $lb(g)$ and $ub(g)$ are the lower and upper bounds of a gene g. These bounds define the solution space of the algorithm and stay constant throughout the execution. The solution space is distinct from the search space, and is defined as a constrained region within the search space.

It is important to choose the lower and upper bounds properly; if they are too narrow the correct parameters could fall outside the solution space, and if they are too wide the algorithm might never converge. We will explore different values for these bounds to evaluate the robustness of the algorithm. As we are using synthetic data, the bounds are chosen such that they contain the ground truth. In the real world, the bounds can be defined around previous calibration parameters, or by manual guesstimation.

2.3.2 Fitness evaluation

Every individual must be evaluated using some measure of the quality of its genes. This measure is called fitness and is calculated using a fitness function. The objective functions defined in Section 2.2 will be used for this purpose. The fitness will in turn be used to guide the evolution of the population.

2.3.3 Evolution

To explore the solution space and exploit the best individuals, the population evolves in each iteration of the algorithm. Evolution is a stochastic process that combines the previously calculated fitness with the evolutionary operators selection, crossover, mutation, and elitism.

Selection

The selection step chooses the individuals that forms the basis of the next iteration. Selection is based on the individuals' fitness and can be done in a multitude of ways. The most common are roulette wheel selection and tournament selection. Roulette wheel selection generates a probability distribution that is proportional to the fitness of the individuals. This distribution is used to select individuals for the next iteration. The probability P_i of selecting an individual i is defined by the equation

$$P_i = \frac{F_i}{\sum_{j=1}^{N} F_j} \tag{2.13}$$

where F is the fitness of an individual. This method is illustrated in 2.8.

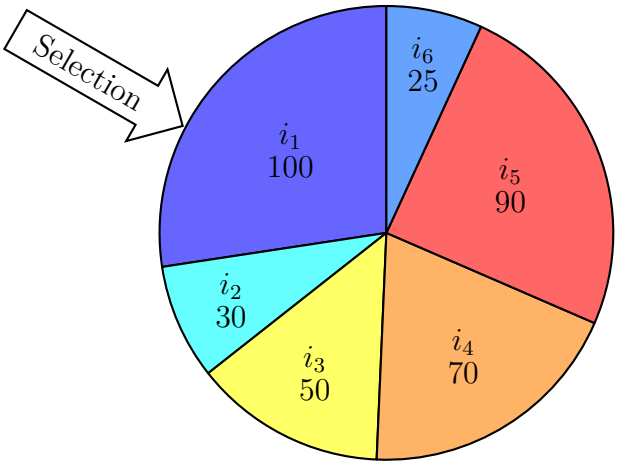


Figure 2.8: Roulette wheel selection. The fitness of each individual is represented by the width of the slices. The method can be thought of as "spinning the wheel".

Another alternative is to use tournament selection, which is illustrated in Figure 2.9. The selection starts by randomly picking a number of individuals defined by the tournament size, n. The second step is to select the individuals in the tournament

by their fitness. The individuals are ordered by fitness, and chosen according to

$$\begin{aligned} P_1 &= p_{\text{tour}} \\ P_2 &= p_{\text{tour}} \cdot (1 - p_{\text{tour}}) \\ P_3 &= p_{\text{tour}} \cdot (1 - p_{\text{tour}})^2 \\ &\vdots \\ P_n &= p_{\text{tour}} \cdot (1 - p_{\text{tour}})^{n-1} \end{aligned} \tag{2.14}$$

where P_i is the probability of choosing individual i and p_{tour} is the probability of choosing the individual with the largest fitness. p_{tour} is typically quite large. The winner of the tournament then goes on as the offspring.

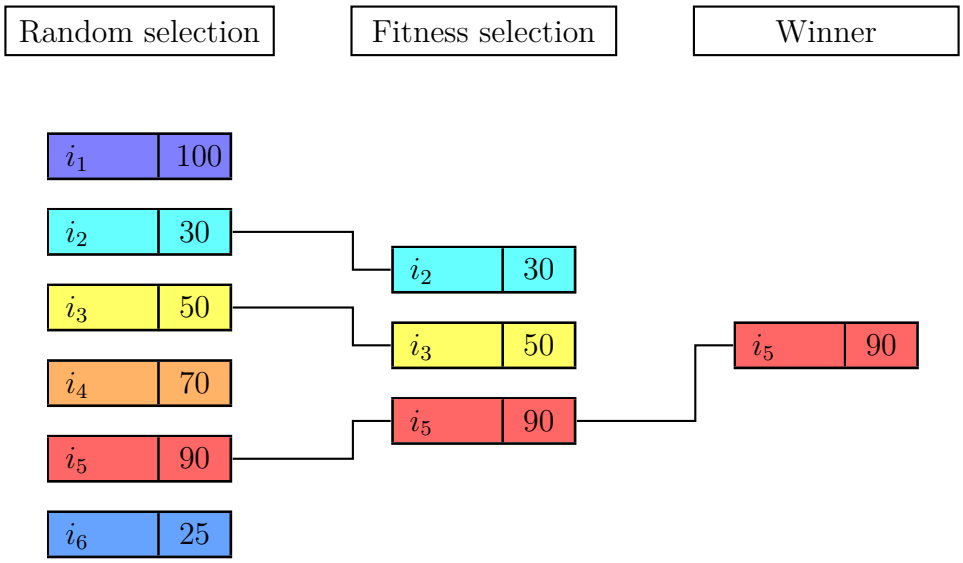


Figure 2.9: Tournament selection with a tournament size 3. Individuals $i_{2,3,5}$ are chosen randomly from the population, and i_5 is the winner of the tournament.

Crossover

The next part of the evolution process is crossover. Crossover mimics the process of reproduction, where the genes of two individuals are mixed and the offspring hopefully inherits favorable traits. These pairs of individuals are chosen from the population according to the probability p_{cross} without replacement. A crossover point is chosen at random, and the two individuals' chromosomes are swapped at that point as illustrated in Figure 2.10. Increasing p_{cross} typically results in faster convergence at the expense of an increased risk of getting stuck in local minima. Therefore, this procedure can be seen as exploitation in optimization terms.

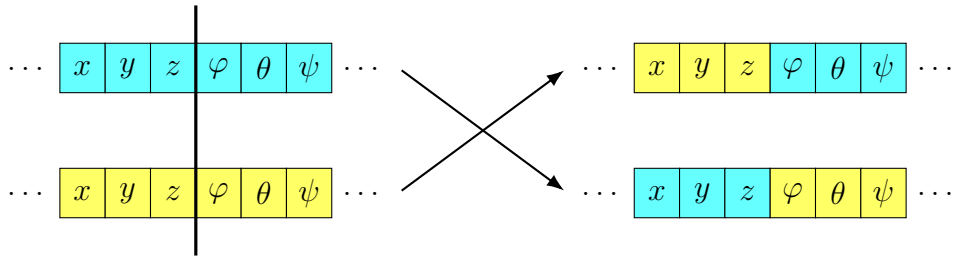


Figure 2.10: Crossover. The two individuals at the left are crossed at the point represented by the black line, resulting in two new individuals at the right.

Mutation

Finally, every gene will be mutated by some probability, p_{mut}. Simply put, mutation is done by changing the value of a random set of genes, which allows the GA to stochastically explore the solution space. An illustration of this concept can be seen in Figure 2.11.

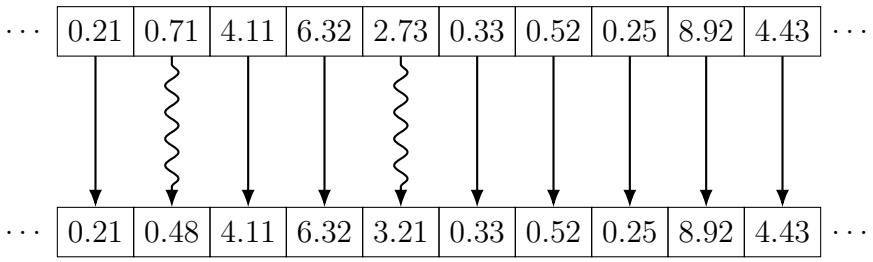


Figure 2.11: Mutation of two genes.

The probability of a gene being mutated is typically chosen as

$$p_{\text{mut}} = \frac{n_{\text{mut}}}{n_{\text{genes}}} \tag{2.15}$$

where n_{mut} is the average number of genes to mutate per chromosome. Mutation is performed by drawing a new gene from a uniform distribution, centered around the value of the previous gene. The width of this distribution is determined by the creep rate, which in turn determines the amount of exploration. The creep rate is typically initialized as quite large, and gradually reduced for each iteration. Due to the different units, different creep rates are used for positions and angles.

To avoid exploring outside of the solution space of interest, mutation is bounded by the same lower and upper bounds, $lb(g)$ and $ub(g)$, as those used to initialize the population in Section 2.3.1. The resulting mutation bounds are defined as

$$\begin{aligned} lb_{\text{mut}}(g) &= \max\left(C(g) - cr(g),\ lb(g)\right) \\ ub_{\text{mut}}(g) &= \min\left(C(g) + cr(g),\ ub(g)\right) \end{aligned} \tag{2.16}$$

where $C(g)$ is the current value and $cr(g)$ is the creep rate of the gene g. These bounds are used in a uniform distribution U, such that the mutated chromosomes are picked according to

$$C_{\text{mut}}(g) \sim U(lb_{\text{mut}}(g), ub_{\text{mut}}(g)) \tag{2.17}$$

Elitism

Even though the probability of the fittest individual getting offspring is high, it is not 100%. Also, the offspring might mutate and be crossed over with another individual, meaning the fittest individual in one iteration might not carry on into the next iteration. Therefore, to ensure that a good individual does not get lost, an exact copy of the fittest individual is made and inserted into the next iteration. This procedure is called elitism and ensures that fitness never decreases.

2.3.4 Termination

The evolved population is evaluated, and the process of evolution-evaluation is repeated until some termination criterion is met. After terminating, the population contains several individuals with differing fitness. The final step is to return the genes of the fittest individual as the solution to the optimization problem.

2.4 Particle swarm optimization

As an alternative to a GA, we will also explore using particle swarm optimization (PSO). The theory in this section is based on [20, Ch. 5]. Some concepts and solutions are similar to the ones used in the GA, so the explanations from the previous section will be reused in this section. Like the GA, PSO is also an evolutionary algorithm. It is inspired by the behavior of bird flocks flying, and schools of fish swimming. In these complex systems, each individual organism has its own freedom and intelligence. However, when individuals are combined into a swarm, the swarm behaves as a single organism, expressing swarm intelligence.

Particle swarm optimization can be expressed as particles swarming around the solution space of an objective function, searching for minima. Each particle's position represents the arguments of the objective function. In our case, the position represent the calibration parameters, as shown in Figure 2.4. Each particle also has a velocity, knowledge of its own best-known position, as well as knowledge of the swarms best-known position. In the following sections, each part of the algorithm will be explained in depth.

2.4.1 Initialization

At the start of the algorithm, the particles are given initial positions, p, according to a uniform distribution as

$$p \sim U(lb, ub) \tag{2.18}$$

where lb and ub are the lower and upper bounds on the search space. These bounds are the same as those used for the GA, as explained in Section 2.3.1. An illustration of a particle swarm can be seen in Figure 2.12.

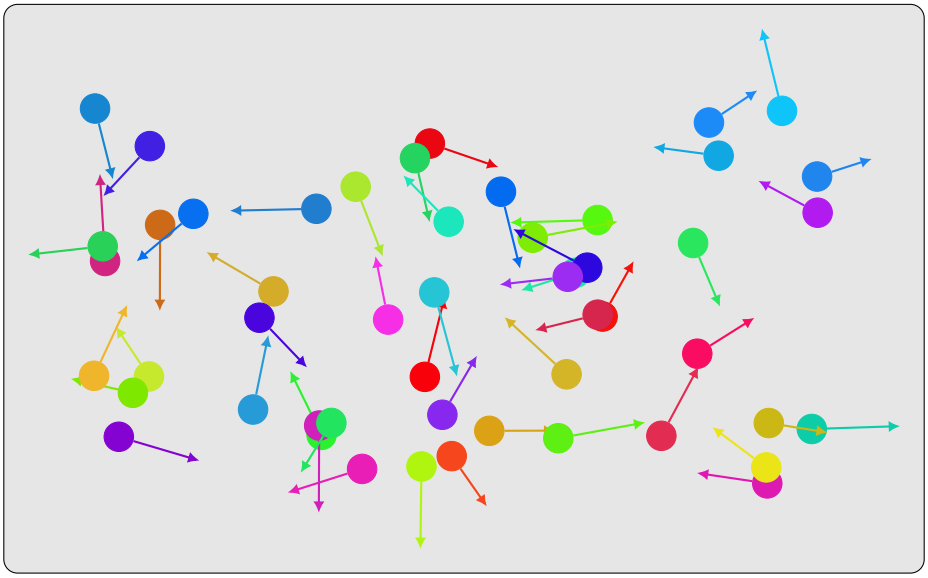


Figure 2.12: A particle swarm. This figure is for illustration purposes only, it does not represent the search space spanned by the calibration parameters.

2.4.2 Evaluation

To determine the quality of a position, the particles must be evaluated using an objective function. The objective functions presented in 2.2 will be used. As most PSO implementations perform minimization, the objective functions are negated.

Each particle has knowledge of its own best-known position p^{pb}, and the swarms best-known position p^{sb}. This information is updated and saved whenever new positions are found that are better than previous ones.

2.4.3 Velocity and position update

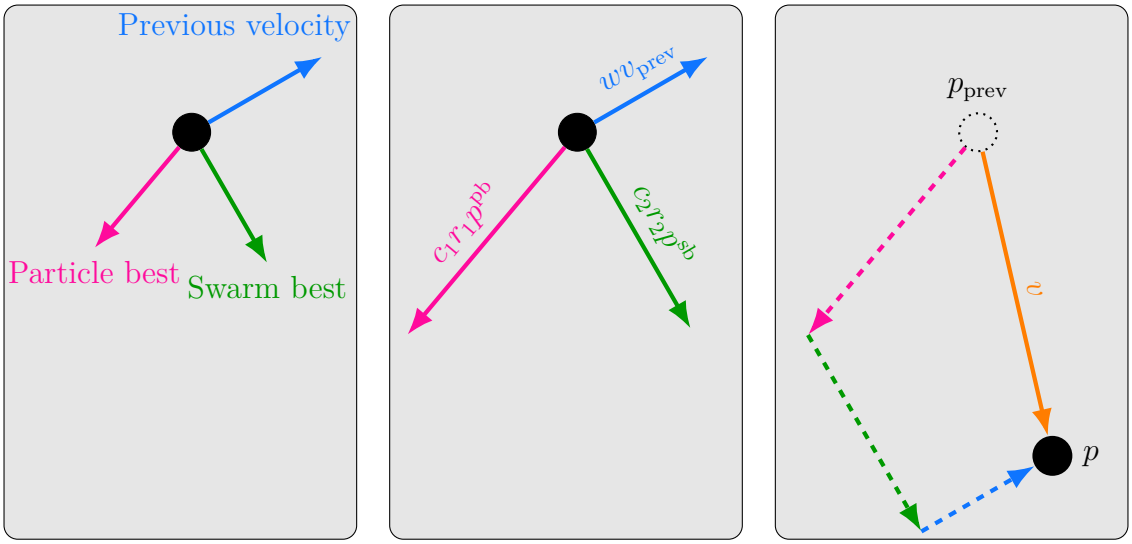
The next step is to update the velocity and position of each particle. The velocity update is done using the previous velocity v_{prev}, as well as the positions p^{pb} and p^{sb}. These three factors are weighed using different tuning parameters, each of which affects how the particles behave. The cognitive component, c_1, determines the particles' reliance on their own performance to find better positions. The social component, c_2, decides the degree to which particles follow the swarm. The inertia weight, w, balances exploration and exploitation, by allowing particles to continue their trajectories as opposed to converging to p^{pb} or p^{sb}. These parameters affect the optimization process heavily and are highly problem specific. Mathematically, the velocity update is defined as

$$v = wv_{\text{prev}} + c_1 r_1 p^{\text{pb}} + c_2 r_2 p^{\text{sb}} \tag{2.19}$$

where r_1 and r_2 are random numbers sampled from the uniform distribution $U \sim (0, 1)$. Using the updated velocity, the position p of each particle is updated as

$$p = p_{\text{prev}} + v \tag{2.20}$$

An illustration of the entire process can be seen in Figure 2.13.



(a) The three directions used for the update. **(b)** Applying the weights. **(c)** Updating the velocity and position.

Figure 2.13: Velocity and position update.

2.4.4 Termination

After the position has been updated, it is evaluated. This process repeats until a termination criterion is met. A common criterion is to terminate after a fixed number of iterations. Finally, the swarm's best-known position, p^{sb}, is returned as the solution to the optimization problem.

2.5 Geometric constraints

The stochastic optimization algorithms only yield relative transformations between the lidars. Consequently, a relationship between the lidars and the vehicle's coordinate system must be obtained manually to geometrically constrain the lidars to the vehicle. Such a relationship can be found by manually determining the pose of a single lidar. However, as discussed in Section 1.4.4, this is not a desirable approach. Therefore, a method based on symmetry around the longitudinal axis is proposed.

An illustration of the proposed method can be seen in Figure 2.14. The longitudinal axis, a_{long}, is defined as the vector that stretches from the center point between the rear lidars to the center point between the front lidars. The latitudinal axis, a_{lat}, is perpendicular to the longitudinal axis. The definition of its absolute orientation is however more complicated. In cases where the lidar positions form a rectangle, as in Figure 2.14, the latitudinal axis is simply the vector between the rear lidars, called v_{rear}. In all other cases, v_{rear} will not be perpendicular to a_{long} and cannot be used as the latitudinal axis. Therefore, the latitudinal axis is defined as the projection of

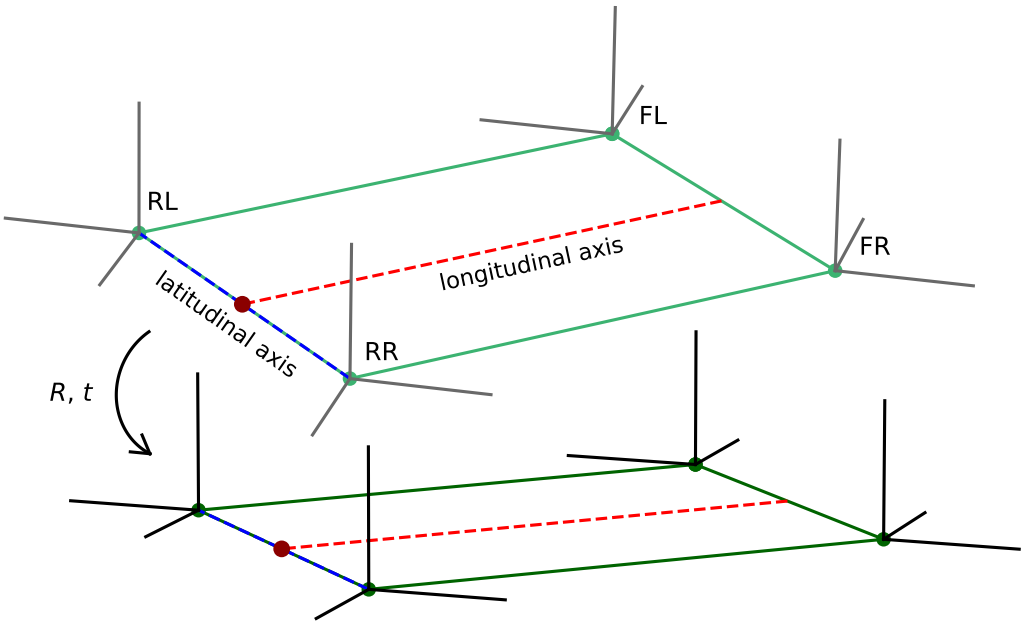


Figure 2.14: The positions and orientations of a relative solution (top) and its constrained counterpart (bottom). R is a rotation matrix and t a translation vector.

v_{rear} onto the plane for which a_{long} is the normal vector, which is expressed as

$$a_{\text{lat}} = v_{\text{rear}} - \frac{v_{\text{rear}} \cdot a_{\text{long}}}{\|a_{\text{long}}\|^2} a_{\text{long}} \tag{2.21}$$

and an illustration of this concept can be seen in Figure 2.15.

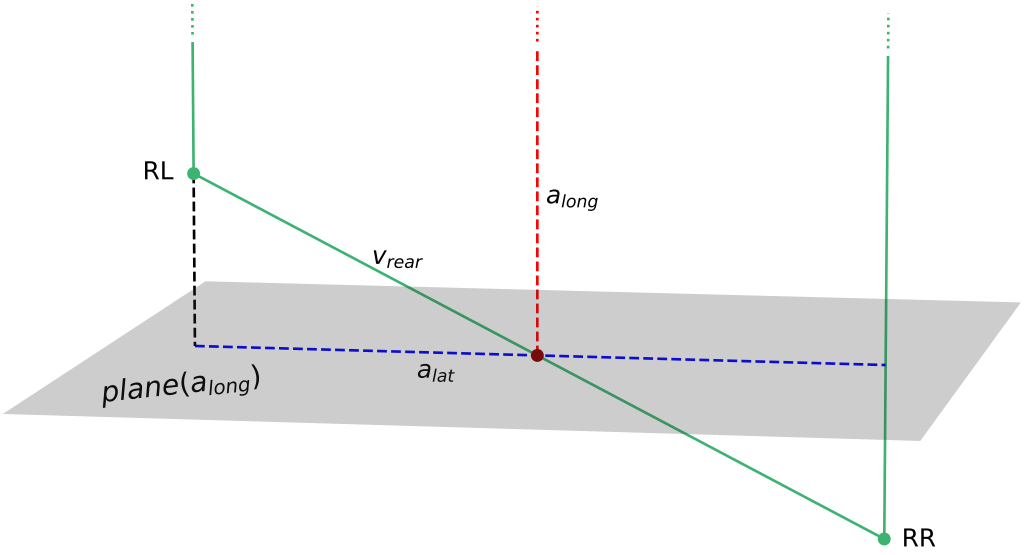


Figure 2.15: Definition of the latitudinal axis, a_{lat}. The longitudinal axis, a_{long}, is a normal vector on a plane. a_{lat} is found by projecting v_{rear} onto this plane.

In the proposed constraining method, the relationship between the lidars and the vehicle's coordinate system is given as a translation, t_c, and a rotation matrix, R_c.

Here, t_c is defined as the center point between the rear lidars. To construct R_c, three angles are needed: The roll φ_c, is the angle between a_{lat} and the vehicle's x-y plane. The pitch, θ_c, is the angle between a_{long} and the vehicle's x-y plane. The yaw, ψ_c, is the angle the between a_{long} and the vehicle's x-z plane. Thus, R_c is defined as

$$R_c = R_z(\psi_c)R_y(\theta_c)R_x(\varphi_c) \tag{2.22}$$

where R_x, R_y and R_z are elemental rotation matrices around the x, y and z axis respectively. Note that this order of multiplication is used for all rotation matrices in this thesis.

To apply the constraints, a corresponding translation vector and rotation matrix must be obtained from the relative solution. The translation, t_{rel}, is the center point between the rear lidars. To get the rotation, R_{rel}, one must first extract the longitudinal and latitudinal axes as described above. These are normalized, yielding $\bar{a}_{\text{long}}$ and $\bar{a}_{\text{lat}}$. A third unit vector, which is perpendicular to both $\bar{a}_{\text{long}}$ and $\bar{a}_{\text{lat}}$ is obtained as

$$\bar{a}_{\text{perp}} = \bar{a}_{\text{long}} \times \bar{a}_{\text{lat}} \tag{2.23}$$

Together, these vectors form the rotation matrix

$$R_{\text{rel}} = \begin{bmatrix} \bar{a}_{\text{long}} & \bar{a}_{\text{lat}} & \bar{a}_{\text{perp}} \end{bmatrix} \tag{2.24}$$

With this at hand, the translation of each lidar, t^{FL}, t^{FR}, t^{RR}, t^{RL} can be constrained by applying the rigid transformation

$$t_c^{XX} = R_c \, R_{\text{rel}}^{\top} \left[t^{XX} - t_{\text{rel}} \right] + t_c \tag{2.25}$$

where XX represent either of the four lidars. To constrain the orientation of the lidars, their rotation matrices must be calculated, yielding R^{FL}, R^{FR}, R^{RR} and R^{RL}. The constrained orientations can then be calculated as

$$R_c^{XX} = R_c \, R_{\text{rel}}^{\top} \, R^{XX} \tag{2.26}$$

Finally, the roll, pitch and yaw angles can be extracted from R_c^{XX} by computing

$$\begin{aligned} \varphi_c^{XX} &= \arctan\left(\frac{R_c^{XX}(3,2)}{R_c^{XX}(3,3)} \right) \\ \theta_c^{XX} &= \arctan\left(\frac{-R_c^{XX}(3,1)}{\sqrt{R_c^{XX}(3,2)^2 + R_c^{XX}(3,3)^2}} \right) \\ \psi_c^{XX} &= \arctan\left(\frac{R_c^{XX}(2,1)}{R_c^{XX}(1,1)} \right) \end{aligned} \tag{2.27}$$

where $R_c^{XX}(i,j)$ is the matrix element at the i-th row and j-th column. These angles along with the constrained translations, t_c^{XX}, represent the final calibration of each lidar.

2.6 Iterative closest point

ICP will serve as a baseline for determining the success of the GA and PSO. The point-to-point variant similar to that described in Section 1.4.1 will be used, as it is not dependent on any feature extraction techniques and is therefore a fair comparison to the proposed calibration methods. To give the ICP similar preconditions as the GA and PSO, it is given an initial guess sampled from

$$T_{\text{init}} \sim U(lb, ub) \tag{2.28}$$

where lb and ub are the same bounds used in the GA and PSO. When calibrating using ICP, the FL lidar is used as the reference that geometrically constrains the lidars to the vehicle's coordinate system. Therefore, its pose is simply given as the ground truth. The remaining lidars are calibrated pairwise in the following order: The first pair is FL-FR, the second pair is FR-RR, and the third pair is FL-RL. These choices result in the FR and RL lidars having ground truth as their reference, while the RR lidar has the FR lidar as its reference.

2.7 Adaptive calibration

As mentioned in the introduction, lidar poses can change while a vehicle is driving. Detecting such a change and recalibrating the lidars is referred to as adaptive calibration. To simulate this scenario, lidar poses will be changed in CARLA while collecting data. We will assume that the lidars are calibrated before introducing changes.

2.7.1 Discrepancy detection

To detect changes in a lidar's pose, its point clouds from two consecutive time steps will be compared. The comparison is done by transforming the point clouds from time k and $k-1$ using the lidar's calibration parameters from time $k-1$. These point clouds are then merged and downsampled using the uniform voxel grid filter presented in Section 2.2.2. The hypothesis is that changes in the pose of a lidar at time k will increase the number of points in the downsampled point cloud, compared to the number of points in the downsampled point cloud at time $k-1$. The difference in points between time steps will be referred to as *discrepancy*.

We assume that the sampling frequency is high, so that the point cloud from time k is quite similar to the one sampled at time $k-1$. Consequently, if the discrepancy is above some threshold it should be due to a change in the lidar's pose, and the lidar should be recalibrated. This threshold is referred to as the *discrepancy tolerance*. If set too small the algorithm becomes too sensitive and will trigger unnecessary calibrations. If it is too large the need for recalibration will go undetected.

The procedure outlined above must be executed for all lidars at every time step.

2.7.2 Online calibration

When recalibration is needed, the search space is reduced to include only the calibration parameters of the affected lidars. This ensures that the process does not affect the calibration of the other lidars. The expectation is that calibrating in a smaller search space should require fewer evaluations in the optimization algorithm without affecting the accuracy. Furthermore, as long as at least one lidar remains calibrated, the constraints presented in Section 2.5 are not needed as the remaining lidars constrains the recalibrated lidars to the vehicle.

The concepts of discrepancy detection and online calibration are summarized as pseudocode in Algorithm 1.

Algorithm 1: Adaptive calibration

Function *adaptive_calibration* **is**
    $\texttt{clouds}_{k-1} \longleftarrow$ initial point clouds
    $\texttt{params} \longleftarrow$ *calibrate*($\texttt{clouds}_{k-1}$)
    $\texttt{clouds}_{k} \longleftarrow$ next point clouds
    for $\texttt{lidar} \in \texttt{lidars}$ **do**
        $\texttt{nPts}_{k-1} \longleftarrow$ *filter*($\texttt{cloud}_{k-1}$, $\texttt{cloud}_{k}$, $\texttt{param}$)
    end
    while *calibration active* **do**
        if *new data available* **then**
            $\texttt{clouds}_{k} \longleftarrow$ new point clouds
            for $\texttt{lidar} \in \texttt{lidars}$ **do**
                $\texttt{nPts}_{k} \longleftarrow$ *filter*($\texttt{cloud}_{k-1}$, $\texttt{cloud}_{k}$, $\texttt{param}$)
                $\texttt{change} \longleftarrow \texttt{nPts}_{k} - \texttt{nPts}_{k-1}$
                if $\texttt{change} > \texttt{threshold}$ **then**
                    $\texttt{calibrate} \longleftarrow \texttt{lidar}$
                end
            end
            if $\exists\ \texttt{lidar} \in \texttt{calibrate}$ **then**
                $\texttt{params} \longleftarrow$ *calibrate*($\forall\ \texttt{lidars} \in \texttt{calibrate}$)
            end
            $\texttt{clouds}_{k-1} \longleftarrow \texttt{clouds}_{k}$
            $\texttt{nPts}_{k-1} \longleftarrow \texttt{nPts}_{k}$
        end
    end
end

Function *filter*($\texttt{cloud}_{k-1}$, $\texttt{cloud}_{k}$, $\texttt{param}$) **is**
    $\texttt{transformed}_{k-1} \longleftarrow$ transform $\texttt{cloud}_{k-1}$ using $\texttt{param}$
    $\texttt{transformed}_{k} \longleftarrow$ transform $\texttt{cloud}_{k}$ using $\texttt{param}$
    $\texttt{concatenated} \longleftarrow$ concatenate $\texttt{transformed}_{k-1}$ and $\texttt{transformed}_{k}$
    $\texttt{filtered} \longleftarrow$ apply voxel grid filter to $\texttt{concatenated}$
    $\texttt{nPts} \longleftarrow$ number of points in $\texttt{filtered}$
end

3

Results

This chapter will present the results of applying the proposed methods to simulated data. It includes a brief summary of the tuning parameters' effect on the performance of the GA and PSO, evaluation of different objective functions, evaluation of robustness to noise, a comparison of the GA, PSO and ICP, and results regarding adaptive calibration.

3.1 Parameter choice - GA

The GA proved difficult to tune due to its large set of parameters, each of which has a significant effect on performance. When evaluating the population size, we saw that the GA did not benefit from increasing the size above a certain threshold. It was be kept rather small to avoid unnecessary evaluations.

Tournament selection with a small tournament size made the algorithm converge more reliably than roulette wheel selection, which indicates that the GA benefits from exploration rather than exploitation. Balancing crossover and mutation was found to be important. Crossover allowed the population to converge to a solution, while mutation was required for exploration. Lastly, using decreasing creep rates, as opposed to constant creep rates, was found to be beneficial. Decreasing creep rates might let the algorithm explore the solution space at the beginning, while allowing for fine tuning at the end.

Table 3.1: Final choice of parameters and evolutionary methods for the GA.

Parameter	**Value**
Population size	12
Selection	Tournament
Tournament size	4
Tournament probability	0.8
Crossover probability	0.2
Mutation rate	$\frac{3}{nGenes}$
Initial creep rate	Max offsets in solution space
Final creep rate	0.02 m & 0.3°
Decreasing creep rate duration	80% of iterations

3.2 Parameter choice - PSO

The parameters c_1, c_2, w, and swarm size were modified until the PSO algorithm converged reliably to an accurate solution. The tuning of these parameters had a significant effect on both the speed and the accuracy of convergence.

The swarm size was found to affect how consistently PSO converged. A larger swarm size gives greater coverage of the solution space at the cost of extra evaluations. If the solution space is not well-covered by particles, the discovery of good minima is inconsistent. However, a larger swarm size does not always mean better performance. Increasing the swarm size was found to yield diminishing returns over a certain threshold. Therefore, attempts were made to find the smallest viable swarm size.

It was important to balance c_1 and c_2 properly. The swarm must be able to converge to a globally superior position, while letting the particles explore the solution space. The inertia weight, w, mostly affected the speed of convergence. Setting w too low resulted in fast convergence to subpar solutions, which was probably due to the particles getting stuck in local minima. Setting w too high resulted in erratic and random behavior.

Table 3.2: Final parameter choice for PSO.

Parameter	**Value**
Swarm size	30
Cognitive component (c_1)	2.0
Social component (c_2)	1.7
Inertia weight (w)	0.7

3.3 Evaluation methods

Every result in the following sections is an average of ten repeated executions, as we determined it was an adequate amount to produce repeatable results. To ensure fair comparisons between the various methods, all tests were done with a constant amount of 60000 objective function evaluations. For instance, a GA with a population size of 12 ran for 5000 iterations and a PSO with a swarm size of 30 ran for 2000 iterations.

Three different sets of point clouds were used, collected from different environments in CARLA. These sets will be referred to as scenes from now on. Scene 1 is the same as the one depicted in Figure 2.1, scene 2 is an urban environment and scene 3 is a rural environment. For each scene, ten point clouds were created with pre-sampled noise, as generating new noise at runtime could affect the comparisons.

We defined calibration accuracy as a percentage of successfully determined calibration parameters, where every parameter was considered individually. For x, y and z to be considered successfully determined, they had to be within 2.5 cm of ground truth, and φ, θ and ψ had to be within 1.0° of ground truth. The average percent-

age of ten calibrations is referred to as the *success rate*, and is used to represent most results. Root mean square (RMS) errors were also computed. To ensure that values of translation and rotation were on a similar scale, meters and radians were used for this computation. As an additional tool to compare the performance of the algorithms, boxplots are included in Appendix A. They depict the error of the calibration results in relation to ground truth for all ten calibrations for every scene and solution space.

The size of the solution space were important when it came to convergence speed and calibration accuracy. Therefore, three different bounds were chosen. They can be seen in Table 3.3 and represent different situations. The small bounds represent recalibrating from previous calibration parameters. The large bounds represent initial calibration with minimal knowledge of calibration parameters. The medium bounds represent a case in between.

Table 3.3: Lower and upper bounds on the search space centered at ground truth, which results in three solution spaces.

	Small	**Medium**	**Large**
Translation	±0.2 m	±0.5 m	±1.0 m
Rotation	±5°	±15°	±45°

3.4 Objective functions

The two proposed objective functions were compared to find the best method with respect to accuracy and consistency. Due to a mistake in the implementation of the distance varying objective function, the distance was calculated from the vehicle's origin instead of the origin of each lidar. This is thought to have a negative effect on the performance of the distance varying objective function.

The GA and PSO were used to test both objective functions. To ensure convergence, they were given the small solution space defined in Section 3.3. Lastly, the point clouds from the three scenes were not corrupted by any noise or outliers, as to not affect the evaluation of the best possible accuracy.

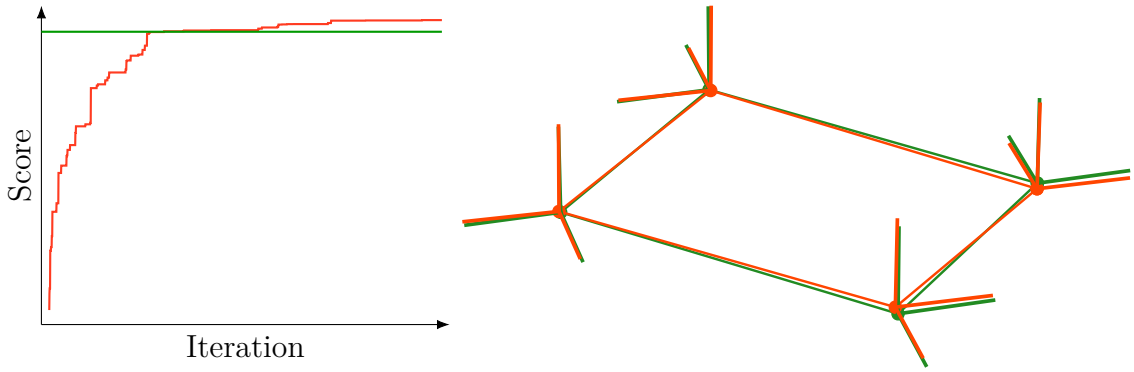
The resulting success rates are summarized in Table 3.4. As can be seen, the uniform objective function performed best on all scenes in both algorithms. It will therefore be used as the objective function for the remainder of the results.

Table 3.4: Comparison of the objective functions' success rates.

	Uniform		Varying	
	GA	PSO	GA	PSO
Scene 1	90.0%	92.1%	59.6%	70.8%
Scene 2	94.6%	98.3%	65.4%	82.5%
Scene 3	86.7%	92.1%	41.7%	45.8%

3.4.1 Flaw in the objective functions

Ideally, the objective functions should associate the highest possible score to the ground truth. This was not the case with the proposed objective functions. More specifically, the calibration algorithms might find incorrect calibration parameters despite having converged. An example of this flaw can be seen in Figure 3.1, which shows the score history and final calibration poses from the PSO compared to the ground truth. Interestingly, higher scores from the uniform objective function seems to correspond to more accurate calibration parameters than higher scores from the distance varying objective function.



(a) Cost plot **(b)** Pose plot

Figure 3.1: An example of the flaw. A calibration result and its score during calibration are shown in red, while the ground truth is shown in green.

Another interesting observation was seen in experiments where point clouds were downsampled prior to running the calibration algorithm. In these cases, the correlation between the score and the ground truth was even worse.

3.5 Robustness to sensor noise

To evaluate the algorithms' robustness to sensor noise, calibration was performed using data with and without applied noise. This procedure was done on all three scenes using the small solution space. The results are summarized in Table 3.5 where it can be seen that both algorithms are affected by the addition of noise.

Table 3.5: Comparison of success rates with and without noise.

	Without noise		**With noise**	
	GA	PSO	GA	PSO
Scene 1	90.0%	92.1%	79.2%	72.9%
Scene 2	94.6%	98.3%	87.1%	87.5%
Scene 3	86.7%	92.1%	47.1%	85.4%

3.6 GA, PSO and ICP comparison

To compare the performance of the algorithms, they were all tested against each other using all three solution spaces defined in Table 3.3. The resulting success rates are summarized in Table 3.6. In addition, the RMS errors are provided in Table 3.7. It is important to note that the GA and PSO were given 2 of the 24 calibration parameters through the constraints, which represent 8.33 % success rate. Similarly, the ICP was given 6 of the 24 calibration parameters, which represent 25 % success rate. With this in mind, the tables show that PSO consistently yields the most accurate calibration parameters. The GA performs similarly to PSO for the small solution space, but struggles in larger solution spaces. Both methods beat ICP, which is especially evident when comparing their RMS errors in Table 3.7 and boxplots in Appendix A. Moreover, ICP seems to struggle regardless of the size of the solution space.

The differences in the performance is abundantly clear when comparing e.g. Figure A.16, A.17 and A.18, which shows boxplots from scene 2 using the large solution space. Here, PSO manages to successfully calibrate a majority of the parameters, while the GA and ICP barely gets any parameters right.

Table 3.6: Comparison of the success rates of the algorithms for each scene. Scenes 1-3 are abbreviated $S_{1\text{-}3}$.

	Small			**Medium**			**Large**		
	GA	PSO	ICP	GA	PSO	ICP	GA	PSO	ICP
$\mathbf{S_1}$	79.2%	72.9%	45.0%	44.6%	70.0%	44.6%	37.1%	74.2%	45.8%
$\mathbf{S_2}$	87.1%	87.5%	43.8%	72.9%	92.1%	50.8%	9.2%	75.8%	53.3%
$\mathbf{S_3}$	47.1%	85.4%	62.5%	39.6%	82.5%	59.6%	10.4%	65.8%	63.8%

Table 3.7: Comparison of the RMS error of the algorithms for each scene. Scenes 1-3 are abbreviated $S_{1\text{-}3}$.

	Small			**Medium**			**Large**		
	GA	PSO	ICP	GA	PSO	ICP	GA	PSO	ICP
$\mathbf{S_1}$	0.020	0.020	0.828	0.078	0.021	0.941	0.229	0.026	0.775
$\mathbf{S_2}$	0.015	0.013	0.723	0.022	0.012	0.789	0.499	0.024	0.797
$\mathbf{S_3}$	0.038	0.016	0.901	0.155	0.019	0.977	0.642	0.034	0.833

3.7 Adaptive calibration

A number of assumptions and expectations were presented in Section 2.7, which must be evaluated to conclude whether the proposed methods for discrepancy detection and online calibration are feasible.

3.7.1 Discrepancy detection

For discrepancy detection to be feasible, it must be sensitive to small changes in the calibration parameters, while avoiding false detections. The sensitivity to changes is determined by the discrepancy tolerance. We chose this tolerance as the largest discrepancy observed when driving around in CARLA without altering the lidars' poses. The speed of the vehicle was set to 30 km/h, and the sampling frequency was set to 30 Hz. The test data was gathered from a series of simulations conducted at the same locations as those used to decide the discrepancy tolerance. In each simulation, one of the pose parameters of the FL lidar was modified ten times with a magnitude according to the values in Table 3.8. After each modification the parameter was reset to its original value. Note that all modifications in translation were positive, since negative translations would move the lidar inside the vehicle. For rotation this was not an issue, so these modifications were alternated between being positive and negative.

Table 3.8: Magnitude of modifications applied to the FL lidar in the different tests.

	Tiny	**Small**	**Medium**	**Large**
Translation	0.01 m	0.05 m	0.10 m	0.20 m
Rotation	$\pm 0.05°$	$\pm 0.10°$	$\pm 0.50°$	$\pm 1.0°$

The resulting sensitivity of the discrepancy detection algorithm is summarized in Table 3.9. It shows that the sensitivity is highly dependent on the parameter being altered. False detections can occur, an example of which can be seen in the x-column for the medium test. In the same test, the algorithm only detected negative modifications to ψ.

Table 3.9: Results on discrepancy detection given as a percentage of detections made by the algorithm out of the ten introduced modifications.

	x	y	z	φ	θ	ψ
Tiny	10%	0%	0%	30%	10%	0%
Small	60%	0%	100%	100%	80%	0%
Medium	110%	0%	100%	100%	100%	50%
Large	100%	100%	100%	100%	100%	100%

3.7.2 Search space reduction

New data had to be generated to examine the effects of calibrating on a reduced search space, so three scenarios were produced with different numbers of modified lidar poses. The data was collected by running the same simulation in CARLA three times, and modifying lidar poses by a given amount. The lidar poses that were modified in each scenario can be seen in Table 3.10.

Table 3.10: The lidar poses that were modified in each scenario.

	FL	**FR**	**RL**	**RR**
Scenario 1	✗			
Scenario 2	✗	✗		
Scenario 3	✗	✗	✗	

PSO was used for calibration, together with the small solution space defined in Table 3.3. Note that the solution space was centered around the previous ground truth, but we ensured that the new ground truth was within the solution space. Calibrating using a reduced search space was compared to calibrating using the full search space by running the PSO for 60000 evaluations on all three scenarios.

We expected that reducing the search space would yield faster and more accurate calibration. This was not the case at all. As can be seen in Table 3.11, reducing the search space resulted in worse accuracy for all three scenarios.

Table 3.11: Comparison of success rates when calibrating using the full search space and the reduced search space.

	Full	**Reduced**
Scenario 1	97.9%	94.2%
Scenario 2	96.3%	77.5%
Scenario 3	95.4%	53.3%

4

Discussion

This chapter will discuss the methods used in the thesis and the results from the previous chapter. It will also explore to what degree the research questions posed in the introduction were answered.

4.1 Objective functions

The first research question is stated as follows:

> How can an objective function be defined for calibrating 3D lidars? How does the choice of objective function influence the calibration algorithm's ability to consistently converge to accurate calibration parameters?

This question is answered by the objective functions presented in the Section 2.2, and the results in Section 3.4. It is evident from the results that a well-defined objective function is paramount for successful stochastic optimization. Even though the distance varying objective function performed a similar operation as the uniform objective function, they produced very different results.

The flaw presented in Section 3.4.1 indicates that the uniform objective function does not work exactly as desired. Similar behavior was observed, but to a higher extent, when we attempted to speed up the calibration process by downsampling the point clouds prior to calibration. Consequently, we believe this issue is correlated to the downsampling that is performed as part of the objective function. To verify if this is the case, calibration should be performed on higher resolution data. Higher resolution means less downsampling, which should in turn mitigate the flaw if the hypothesis is correct. Regardless, the inaccuracies introduced by this issue are very minor and should not be a problem for the intended application.

The distance varying objective function was believed to mitigate this issue by varying the filter size depending on the distance. On the contrary, it performed much worse than the uniform objective function. We do, however, believe that the distance varying objective function would have performed better if it was implemented correctly. Regardless of the implementation, we think there is a problem with the shape of the voxels. The combination of the restrictions imposed by equation (2.10), and the fact that the voxels are cubical is believed to introduce discontinuities between the grids. Instead, downsampling should be performed with some kind of spherical mesh, where the mesh size is varied according to the resolution and distance. An

example that might work is a cubed-sphere grid. These mistakes were unfortunately not discovered during development, as point clouds downsampled using the distance varying objective function looked reasonable.

As an alternative approach, using a neural network as an objective function could be interesting. Such a neural network could take a candidate point cloud as input during evaluation, and output a numerical measure of its "calibratedness". It could be trained on a large variety of uncalibrated lidars by simply altering the calibration parameters for each lidar. We think this is worth looking into, as neural networks often perform well at visual tasks such as classification, which this problem essentially is.

4.2 Comparison

The second research question reads as follows:

> How do the GA and PSO compare with respect to robustness, i.e. initial estimation, accuracy, consistency and sensor noise? How do they compare to ICP?

The answers to these questions are presented in Section 3.5 and 3.6, which together shows that PSO is superior in all respects.

The addition of sensor noise to the point clouds also introduces noise in the objective functions. Noisy objective functions are thought to make convergence more difficult for the GA and PSO, yet the results in Section 3.5 indicate that the algorithms are able to converge nonetheless. However, we only introduced white Gaussian noise and outliers as explained in Section 2.1.1, so a more realistic noise model should be developed for testing robustness.

We are surprised to see how much better PSO performed compared to the GA. However, as stated in [20, Ch. 1], some stochastic optimization algorithms do outperform others on specific classes of problems. While this might be true, the GA was very difficult to tune and might have performed better using another combination of parameters or other evolutionary methods. A notable example is Boltzmann selection, which is a selection scheme that gradually shifts from exploration to exploitation. We also recommend trying other stochastic optimization algorithms, as some might perform even better than PSO.

The ICP algorithm performed poorly on all tests, and was outperformed by the proposed methods. We believe that ICP struggles since the point clouds from the lidars are not identical. As a result, the nearest neighbour search yields incorrect point-correspondences. Subsequently, the distance minimization is performed on incorrect correspondences, and the algorithm diverges. It is possible that more sophisticated versions of ICP would perform better. However, these usually rely on feature extraction techniques and are not considered comparable to the objective functions used in the GA and PSO.

4.3 Adaptive calibration

The third research question is stated as follows:

> Which, if any, modifications and additions must be made to the proposed calibration algorithm to make it adaptive? How can an algorithm be defined to detect changes in lidar poses? What are the advantages and challenges with using such an algorithm?

The question was answered by adding discrepancy detection and reducing the search space as described in Section 2.7, and by the results in Section 3.7. Several challenges and potential improvements were discovered during development.

The discrepancy detection worked quite well for angular changes, but was not very sensitive to translational changes. We believe the reason is that angular changes affects the overlap in the merged point cloud much more significantly than translational changes. However, it is very unlikely that a change in pose only affects a single parameter, so the concept should work. Unfortunately, it makes big demands with regards to the update frequency of the data, and requires an uninterrupted chain of comparisons between each time step. Recalibration must be done in between time steps to not break this chain. The result is an algorithm that requires vast amounts of computing power constantly, which is a limited resource on vehicles. An alternative approach to discrepancy detection could be to use neural networks to determine if calibration is needed, in the same way as discussed in Section 4.1.

A surprising result is that calibrating a subset of the lidars performed worse than calibrating all the lidars. Intuitively, reducing the search space of a problem should make it easier to optimize, which this result seems to contradict. A possible explanation is that optimizing in the entire search space has an infinite amount of solutions, as the lidars are constrained to the vehicle after the optimization process. Removing one or more lidars from the search space effectively constrains the problem beforehand, which results in the optimization problem having a unique solution. As a consequence, the optimization problem becomes much harder, and the PSO's parameters might need to be retuned.

Future work on adaptive calibration should explore terminating the calibration algorithm as soon as it has converged. Usually, this is not possible, as the maximum score of any given scene is unknown. However, in online calibration, knowledge of the previous score can be used. The calibration algorithm can, for instance, terminate at a certain percentage of the score it had before the lidar poses changed. This approach assumes that the score stays fairly constant as long as the lidars are calibrated.

An alternative approach to adaptive calibration could be to calibrate each lidar intertemporally. The idea is to compute a relative transformation between point clouds from two time steps for each lidar, and use it to transform the previous lidar pose. This approach would theoretically handle all four lidars changing pose, as the calibration is done relative to their previous calibration, and not relative to each other.

4.4 Geometric constraints

An important part of the calibration algorithm is the constraints introduced in Section 2.5. The measurements required for this constraining method will introduce errors as they are done manually. However, our approach only needs measurements on the translations of all lidars. Since the translations are assumed to be symmetric around the vehicle's coordinate system, we consider these measurements simpler to obtain than determining the orientation and translation of a single lidar, and should thus introduce less errors.

Ideally, the constraints should be determined automatically, for which additional information is required. A combination that could work is a 3D map of the environment and knowledge of the vehicle's position.

4.5 Methodological issues

During the course of the thesis, some weaknesses in the methods, the proposed solutions, and the results were discovered.

We implemented the GA, voxel grid filters, and data collection ourselves. As a consequence, we cannot exclude the possibility of bugs in the code affecting the results.

A multitude of scenes should be used to evaluate the performance of each algorithm more diversely. Only three scenes were used for collecting the results, as the main purpose was to check if the proposed algorithms worked, and for comparing the different approaches. Furthermore, we set the constraints using the ground truth. In reality, such accuracy cannot be achieved, so the results might be slightly misleading. Additionally, as the adaptive part of the calibration was defined as a secondary aim, the related methods were not tested and tuned as thoroughly as the rest. Therefore, we could have missed something that would emerge from doing more tests.

An unrealistic assumption regarding the sensor model was a FOV of 270°, which is significantly larger than specifications of planned solid-state lidars. Larger FOV corresponds to more overlap between the lidars, which probably results in faster convergence and more accurate calibration. Accordingly, the algorithms' should have been tested on lidar data with less overlap than 180°.

5

Conclusion

The purpose of this thesis has been to develop a fully automatic calibration algorithm for four lidars. The proposed algorithm successfully calibrates all lidars simultaneously using only point cloud data, without the need for any calibration environment or feature-extraction techniques. Additionally, the algorithm should work for an arbitrary number of lidars as long as there exist sufficient overlap in the lidars' field of view.

Two stochastic optimization methods were explored, namely a genetic algorithm and particle swarm optimization. They were compared with respect to initial estimation, accuracy, consistency and sensor noise. In the end, only the PSO algorithm performs well enough to be considered robust and should work for the intended task.

An objective function based on a voxel grid filter was successfully defined, which allows the calibration algorithm to converge reliably to accurate calibration parameters. Feature-extraction and other application-specific functions were not introduced, so the algorithm should work regardless of the environment where the calibration is performed.

The secondary objective of making the algorithm adaptive was partially successful. Temporal differences in individual lidars' point clouds can be used to detect changes in lidar pose, but might be too computationally demanding for use in real applications. Additionally, calibrating only a subset of lidars performs worse than calibrating all lidars at once.

Future work

The proposed objective function is not perfect, as ground truth is not the global optimum. Therefore, alternatives should be looked into. One approach could be to use neural networks to determine the quality of the calibration parameters. Such a network could be used as an objective function and for detecting changes in lidar pose for adaptive calibration.

Currently, the proposed calibration algorithm requires knowledge of the relationship between the lidars and the vehicle's coordinate system to geometrically constrain the lidars. The relationship must be must be determined manually, which requires human intervention and introduces errors. If this information could be determined automatically, the overall accuracy could be improved.

Bibliography

[1] L. Brown and G. Lindberg, "Extrinsic calibration of multiple 2D Lidars using a Genetic Algorithm," Master's thesis, Chalmers University of Technology, 2019. [Online]. Available: https://hdl.handle.net/20.500.12380/256909

[2] M. Khader and S. Cherian, "An Introduction to Automotive LIDAR," Dallas, 2018. [Online]. Available: http://www.ti.com/lit/wp/slyy150/slyy150.pdf?&ts=1589273697450

[3] F. Lu and E. Milios, "Robot pose estimation in unknown environments by matching 2d range scans," *Journal of Intelligent and Robotic Systems*, vol. 18, no. 3, pp. 249–275, Mar 1997. [Online]. Available: https://doi.org/10.1023/A:1007957421070

[4] S. T. Pfister, K. L. Kriechbaum, S. I. Roumeliotis, and J. W. Burdick, "Weighted range sensor matching algorithms for mobile robot displacement estimation," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 2, May 2002, pp. 1667–1674 vol.2.

[5] B. Jensen and R. Siegwart, "Scan alignment with probabilistic distance metric," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2191–2196 vol.3.

[6] M. Magnusson, "The three-dimensional normal-distributions transform : an efficient representation for registration, surface analysis, and loop detection," Ph.D. dissertation, Örebro University, School of Science and Technology, 2009.

[7] F. Pomerleau, F. Colas, and R. Siegwart, "A Review of Point Cloud Registration Algorithms for Mobile Robotics," *Foundations and Trends in Robotics*, vol. 4, no. 1, pp. 1–104, 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01178661

[8] A. Censi, "An ICP variant using a point-to-line metric," in *2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 19–25.

[9] Z. Pusztai, I. Eichhardt, and L. Hajder, "Accurate calibration of multi-LiDAR-multi-camera systems," *Sensors*, vol. 18, no. 7, p. 2139, Jul. 2018. [Online]. Available: https://doi.org/10.3390/s18072139

[10] Y. Park, S. Yun, C. Won, K. Cho, K. Um, and S. Sim, "Calibration between color camera and 3d LIDAR instruments with a polygonal planar

board," *Sensors*, vol. 14, no. 3, pp. 5333–5353, Mar. 2014. [Online]. Available: https://doi.org/10.3390/s140305333

[11] E. Kim and S. Park, "Extrinsic calibration of a camera-lidar multi sensor system using a planar chessboard," in *2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN)*, 2019, pp. 89–91.

[12] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, p. 381–395, Jun. 1981. [Online]. Available: https://doi.org/10.1145/358669.358692

[13] M. Pereira, V. Santos, and P. Dias, "Automatic calibration of multiple lidar sensors using a moving sphere as target," in *Robot 2015: Second Iberian Robotics Conference*, L. P. Reis, A. P. Moreira, P. U. Lima, L. Montano, and V. Muñoz-Martinez, Eds. Cham: Springer International Publishing, 2016, pp. 477–489.

[14] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-d point sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 5, pp. 698–700, 1987.

[15] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004. [Online]. Available: https://doi.org/10.1023/b:visi.0000029664.99615.94

[16] J. L. Schönberger, H. Hardmeier, T. Sattler, and M. Pollefeys, "Comparative evaluation of hand-crafted and learned local features," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6959–6968.

[17] E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, P. Fua, and F. Moreno-Noguer, "Discriminative learning of deep convolutional feature point descriptors," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 118–126.

[18] R. Hänsch, T. Weber, and O. Hellwich, "Comparison of 3d interest point detectors and descriptors for point cloud fusion," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. II-3, pp. 57–64, Aug. 2014. [Online]. Available: https://doi.org/10.5194/isprsannals-ii-3-57-2014

[19] T.-H. Yu, O. J. Woodford, and R. Cipolla, "A performance evaluation of volumetric 3d interest point detectors," *International Journal of Computer Vision*, vol. 102, no. 1-3, pp. 180–197, Sep. 2012. [Online]. Available: https://doi.org/10.1007/s11263-012-0563-2

[20] M. Wahde, *Biologically inspired optimization methods: an introduction.* Southampton, UK Boston, MA: WIT Press, 2008.

[21] J. M. Maroli, Ü. Özgüner, K. Redmill, and A. Kurt, "Automated rotational calibration of multiple 3d lidar units for intelligent vehicles," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 1–6.

[22] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

A

Boxplots

This appendix contains boxplots which illustrates the accuracy and consistency of the three calibrations algorithms, and is meant as a supplement to Chapter 3. The plots illustrate the error between the ground truth and the calibration result of ten executions of each algorithm. As in Chapter 3, calibration is considered successful when parameters are within 2.5 cm or 1.0° of ground truth.

Contents

A.1 Scene 1

PSO, small solution space

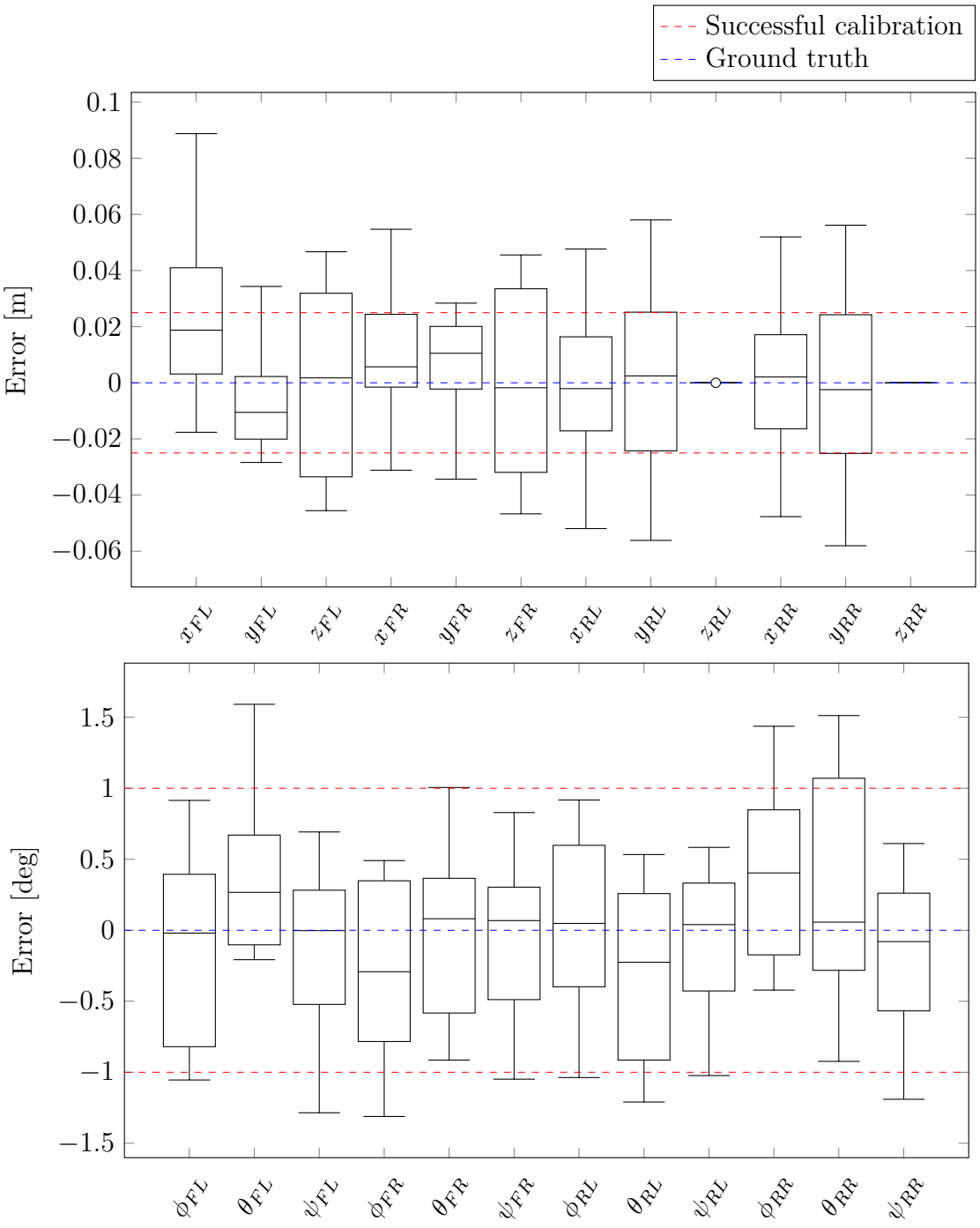


Figure A.1: Box plots of errors in translation and orientation from calibration using PSO with the small solution space on scene 1.

GA, small solution space

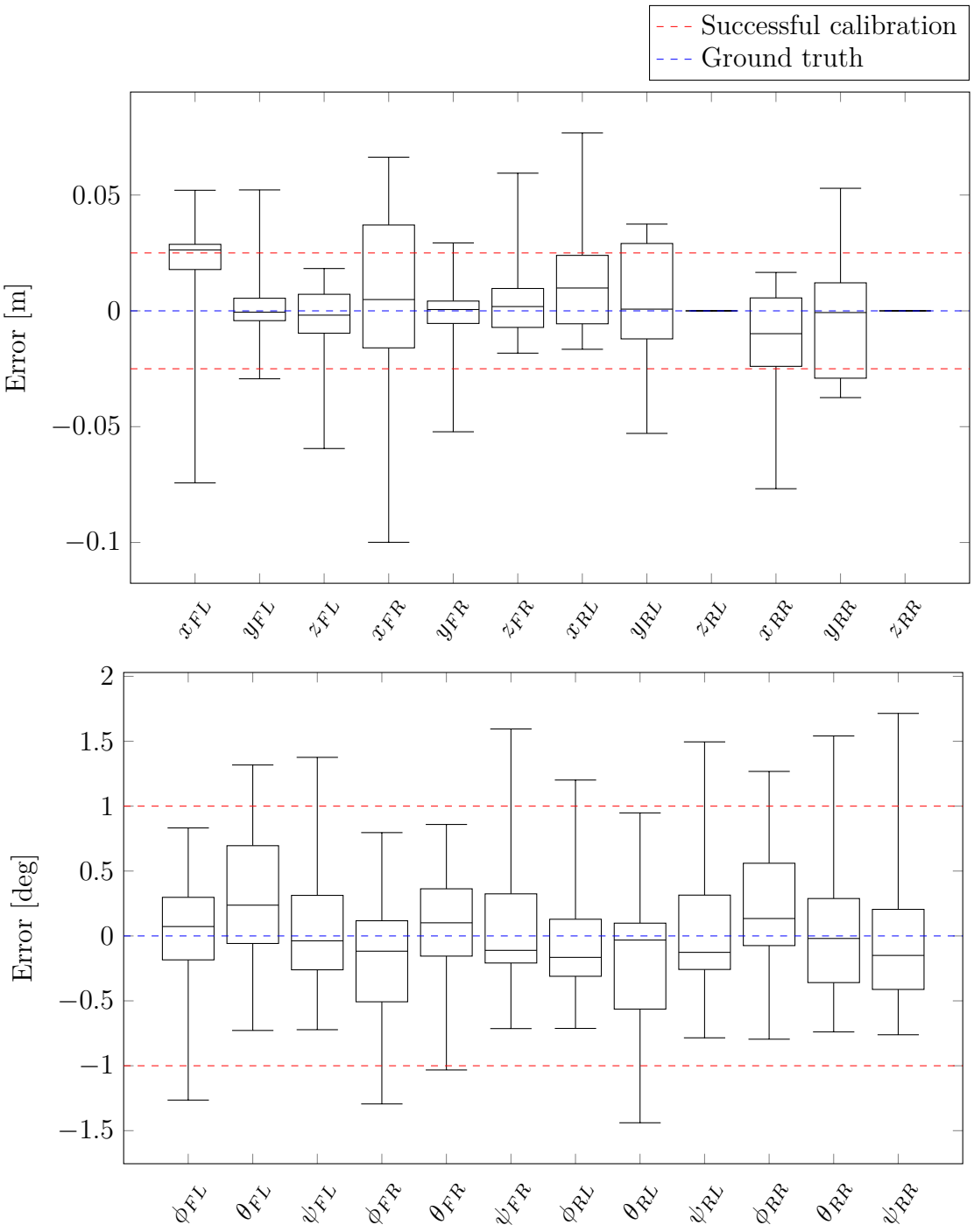


Figure A.2: Box plots of errors in translation and orientation from calibration using the GA with the small solution space on scene 1.

ICP, small solution space

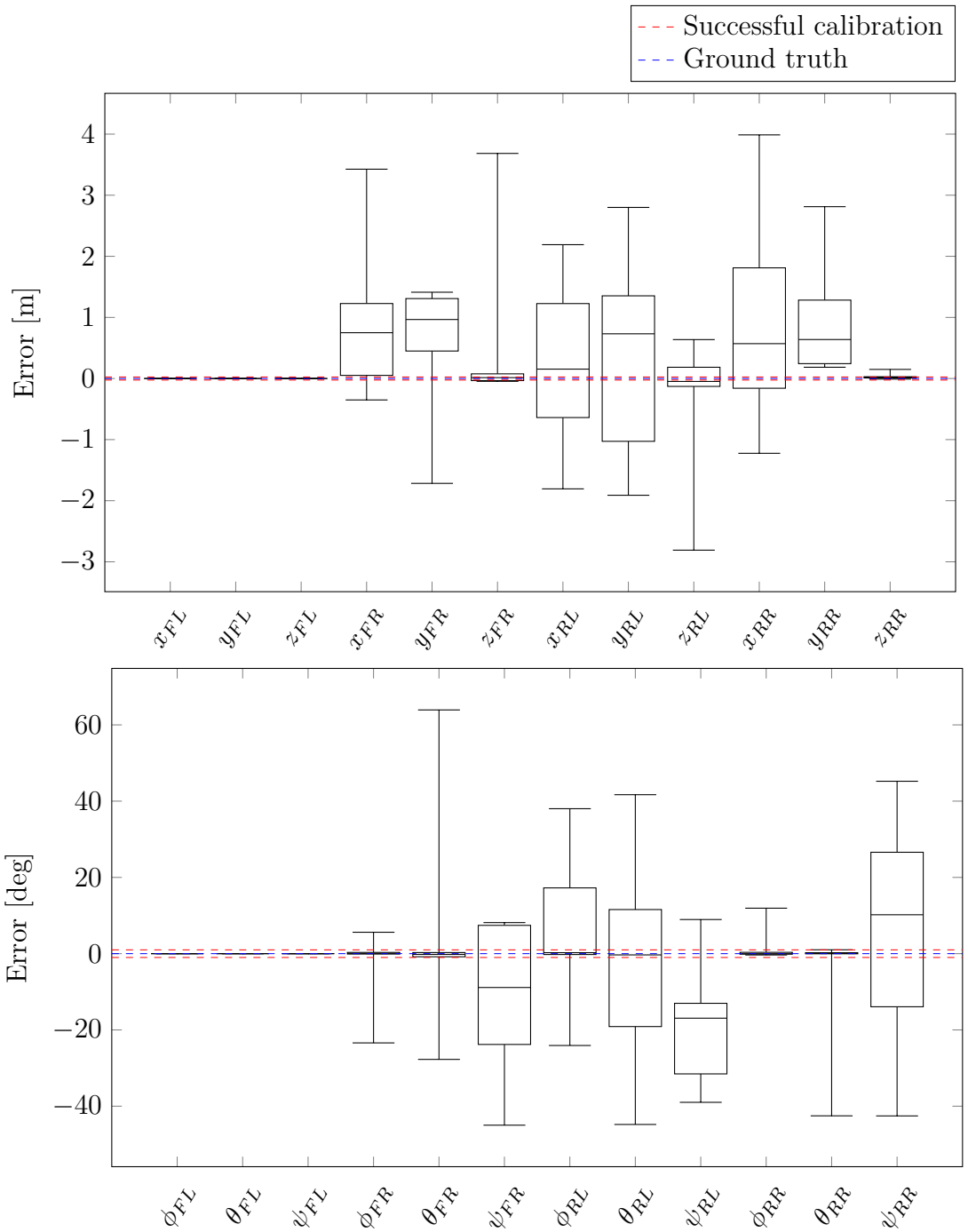


Figure A.3: Box plots of errors in translation and orientation from calibration using ICP with the small solution space on scene 1.

PSO, medium solution space

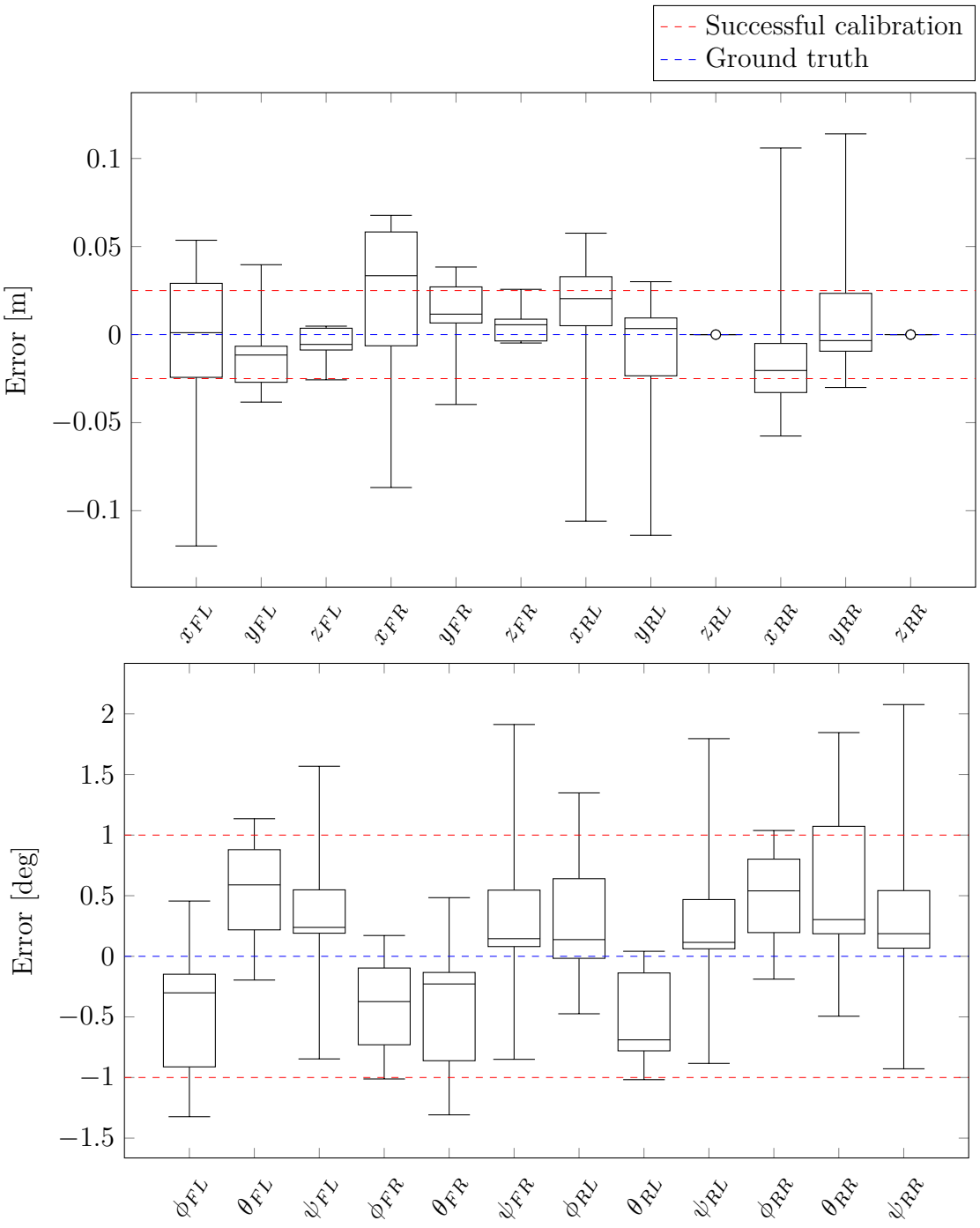


Figure A.4: Box plots of errors in translation and orientation from calibration using PSO with the medium solution space on scene 1.

GA, medium solution space

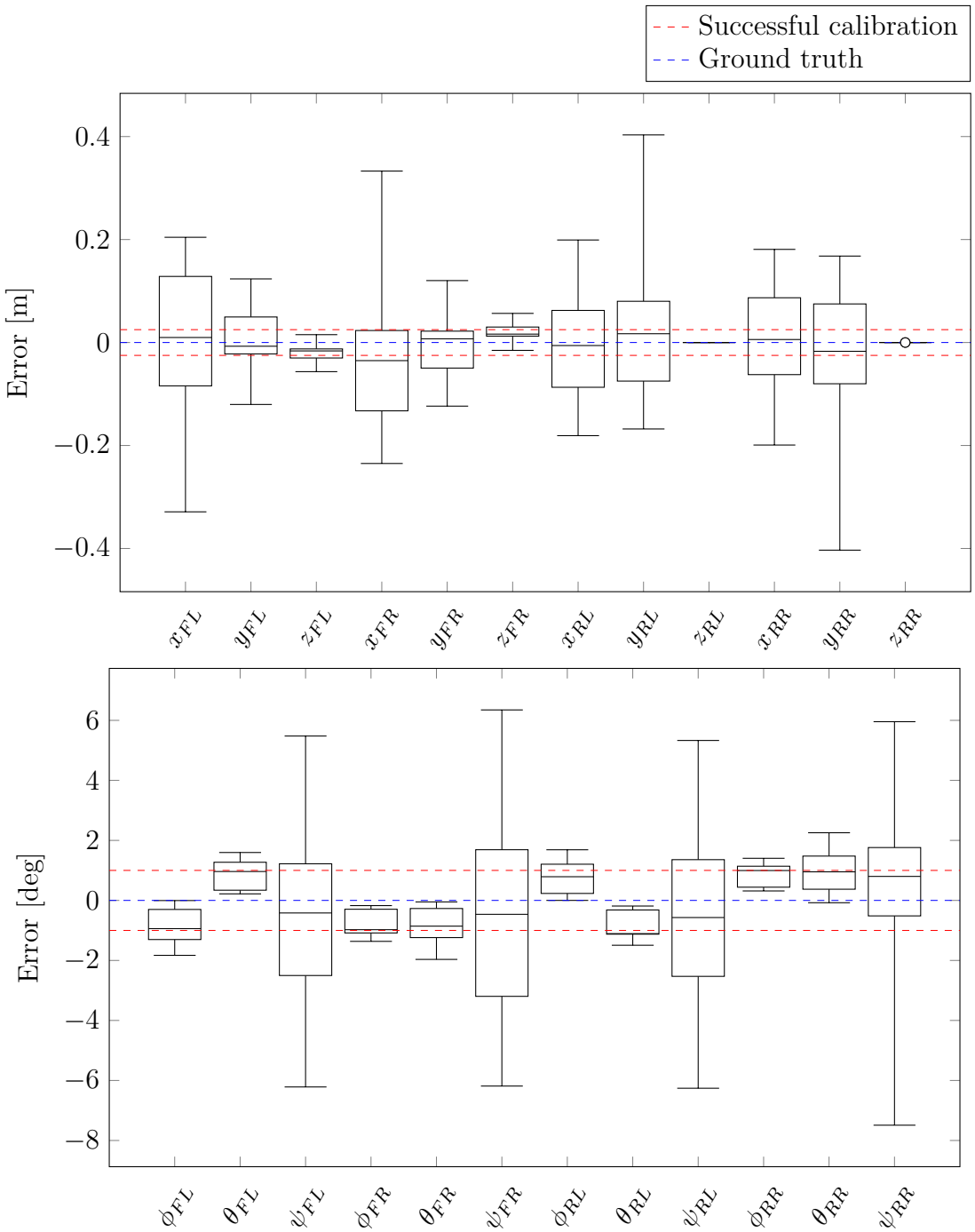


Figure A.5: Box plots of errors in translation and orientation from calibration using the GA with the medium solution space on scene 1.

ICP, medium solution space

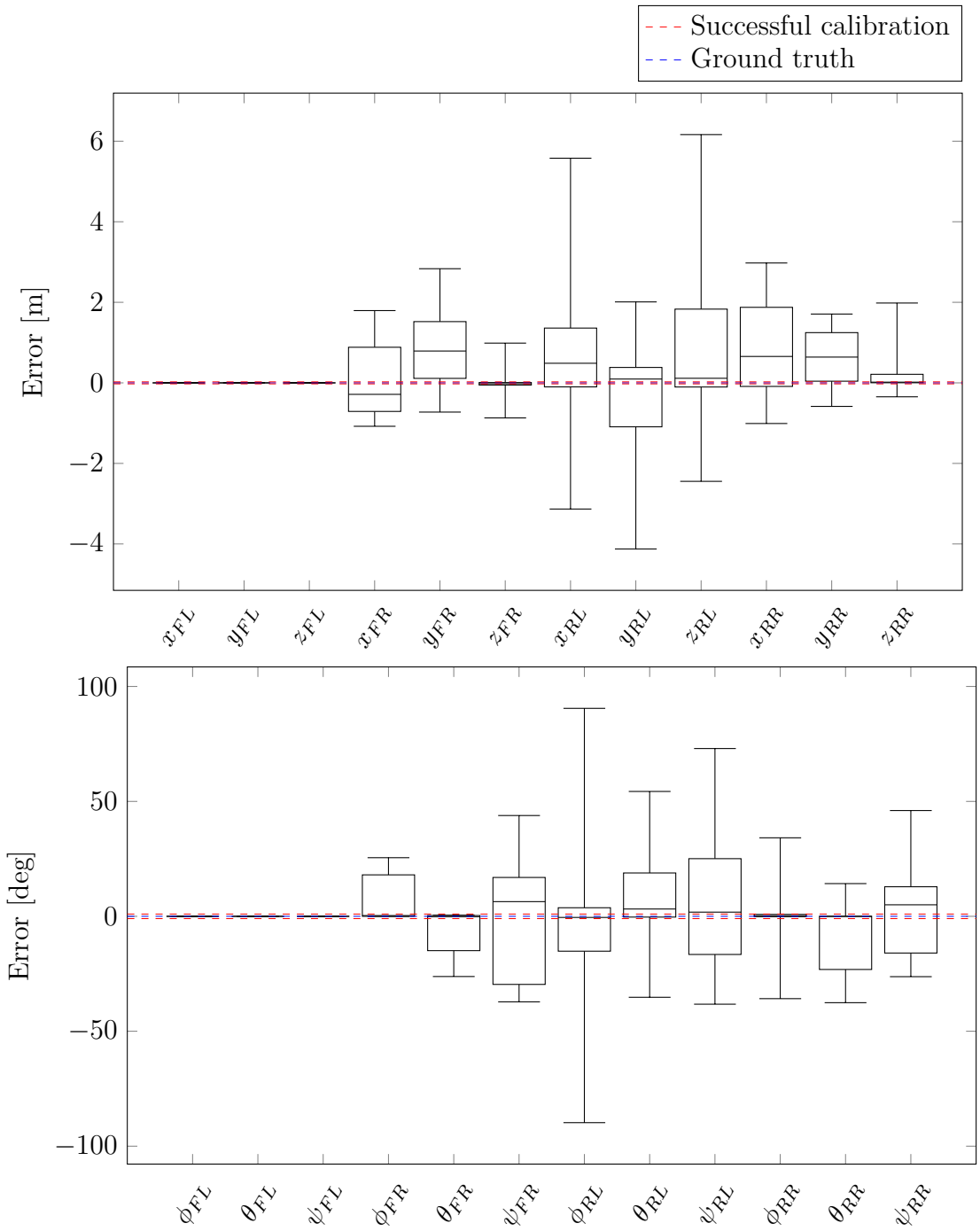


Figure A.6: Box plots of errors in translation and orientation from calibration using ICP with the medium solution space on scene 1.

PSO, large solution space

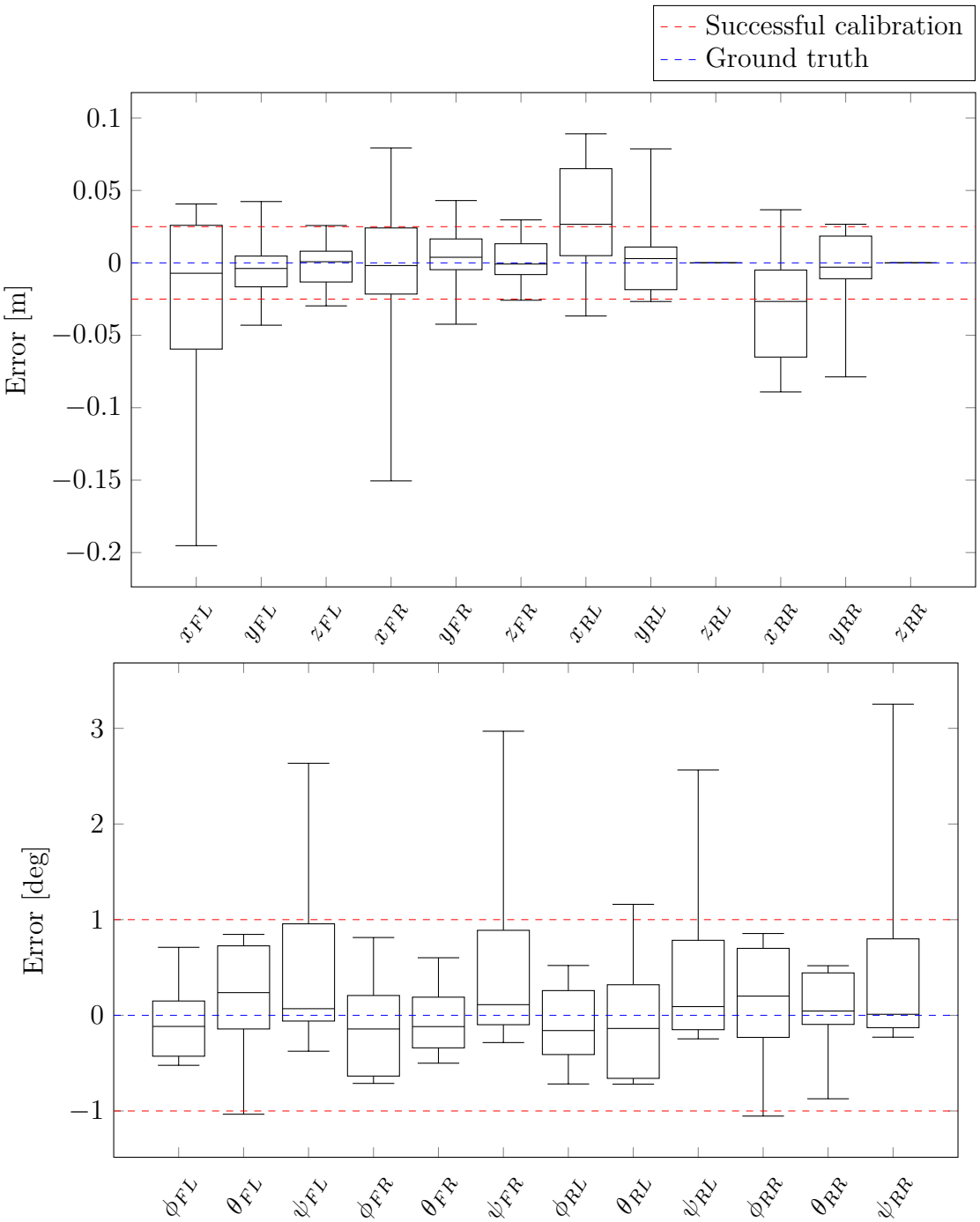


Figure A.7: Box plots of errors in translation and orientation from calibration using PSO with the large solution space on scene 1.

GA, large solution space

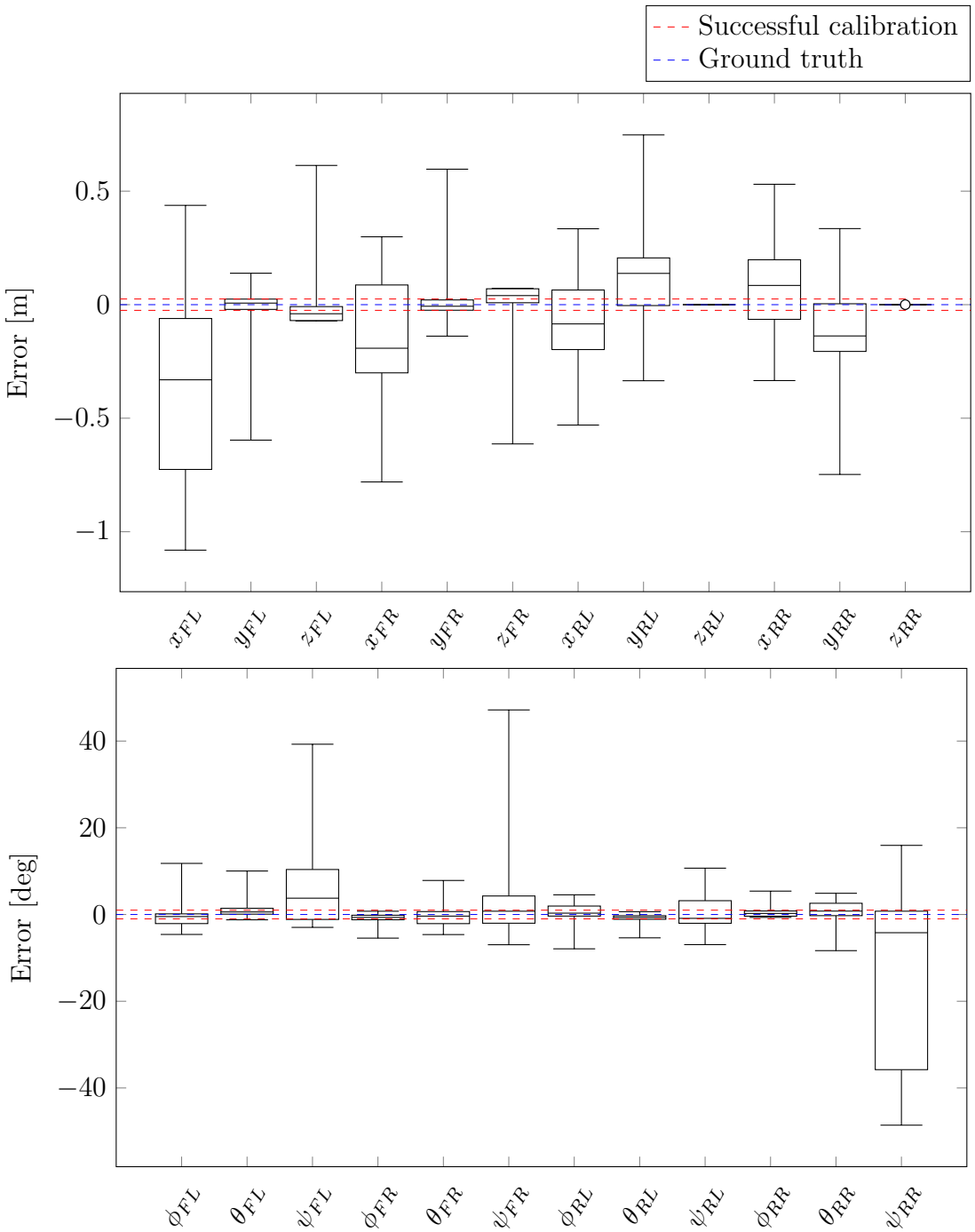


Figure A.8: Box plots of errors in translation and orientation from calibration using the GA with the large solution space on scene 1.

ICP, large solution space

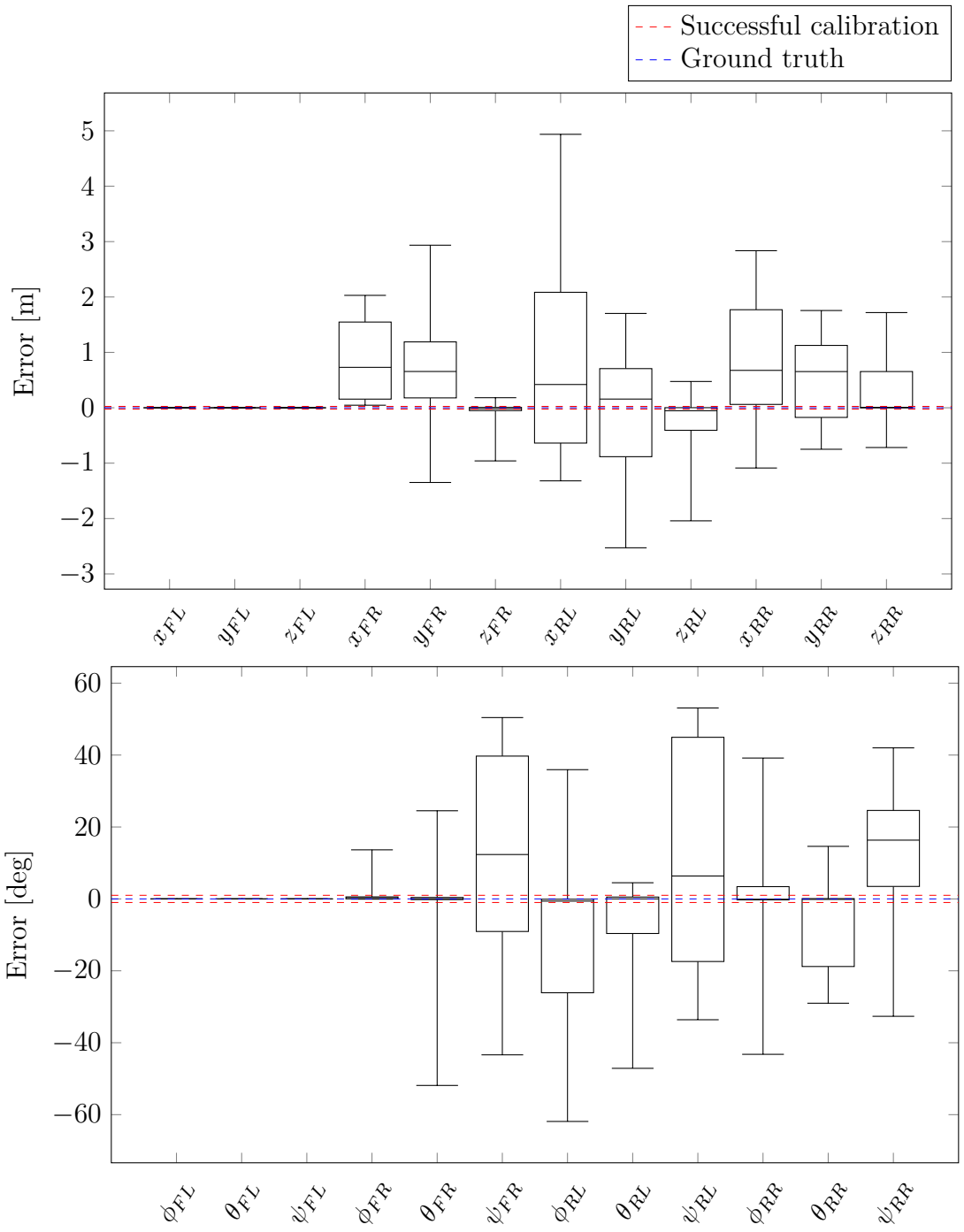


Figure A.9: Box plots of errors in translation and orientation from calibration using ICP with the large solution space on scene 1.

A.2 Scene 2

PSO, small solution space

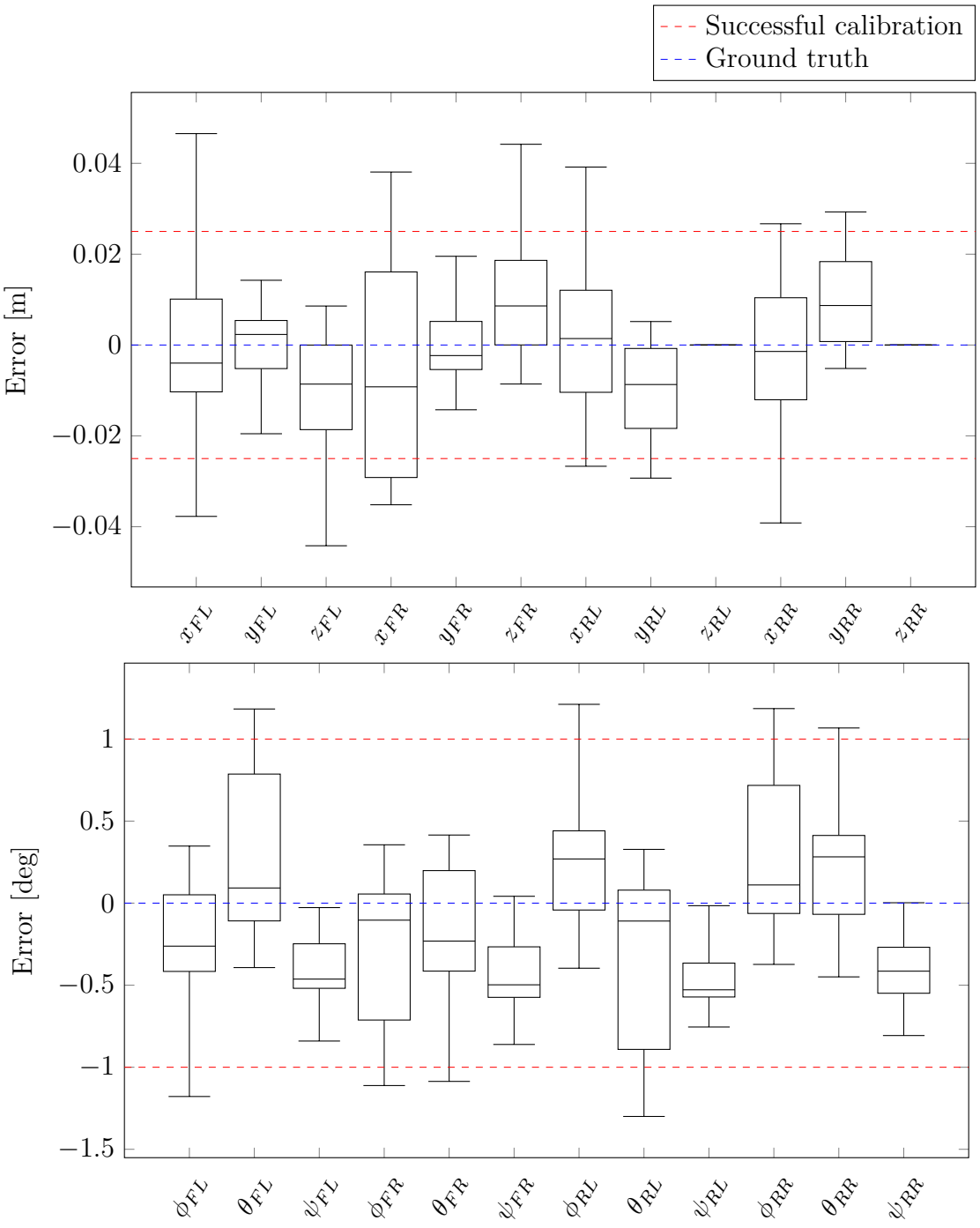


Figure A.10: Box plots of errors in translation and orientation from calibration using PSO with the small solution space on scene 2.

GA, small solution space

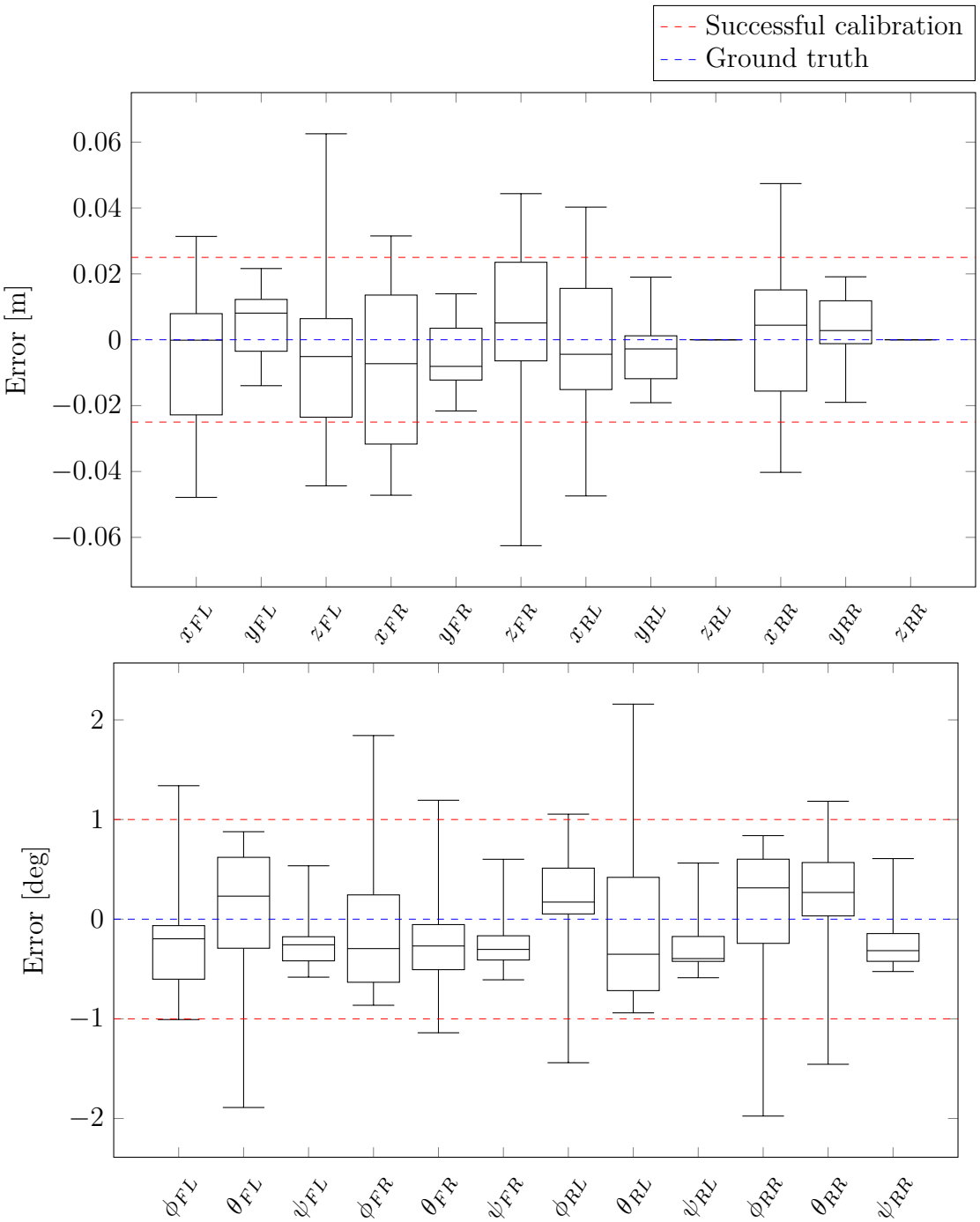


Figure A.11: Box plots of errors in translation and orientation from calibration using the GA with the small solution space on scene 2.

ICP, small solution space

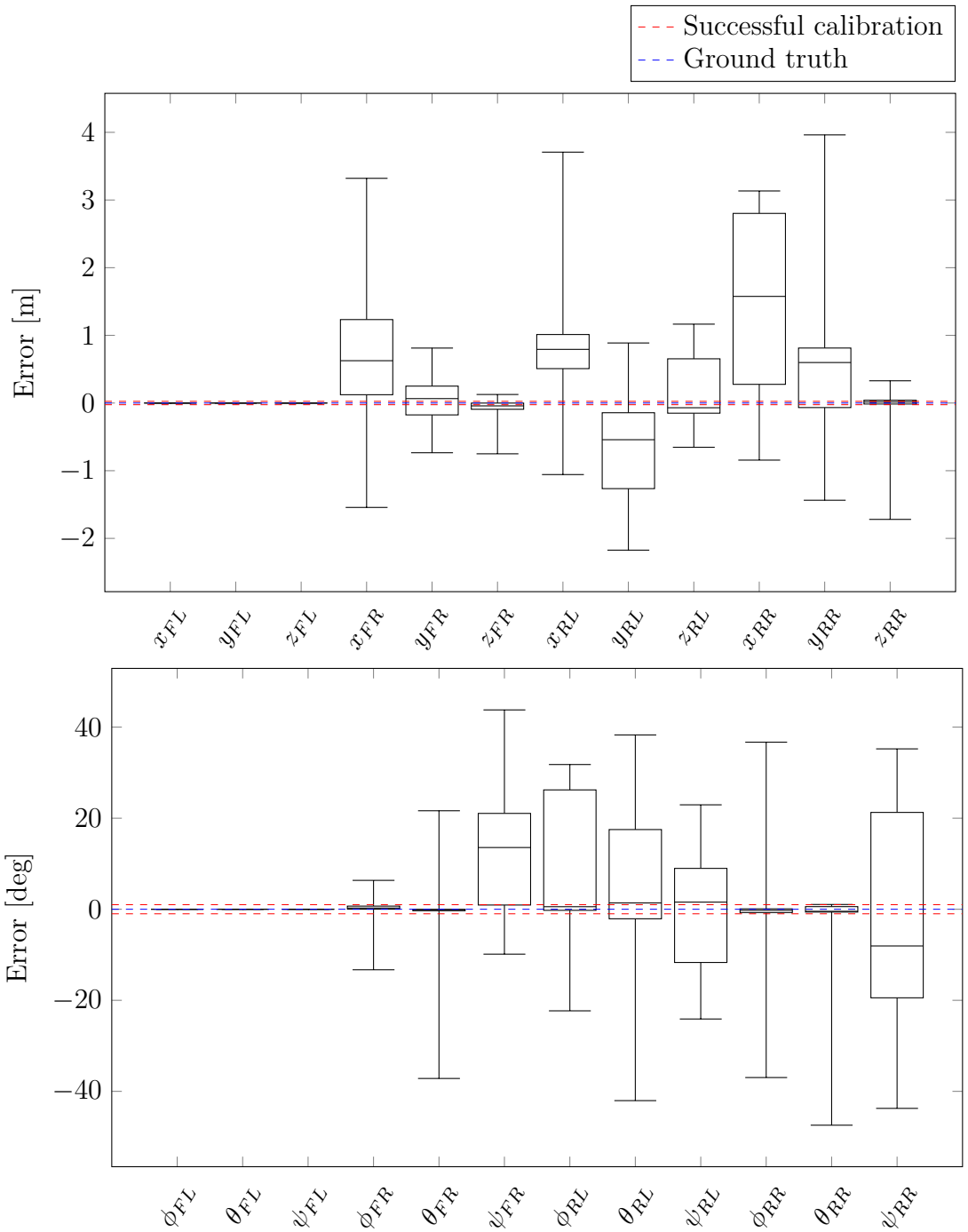


Figure A.12: Box plots of errors in translation and orientation from calibration using ICP with the small solution space on scene 2.

PSO, medium solution space

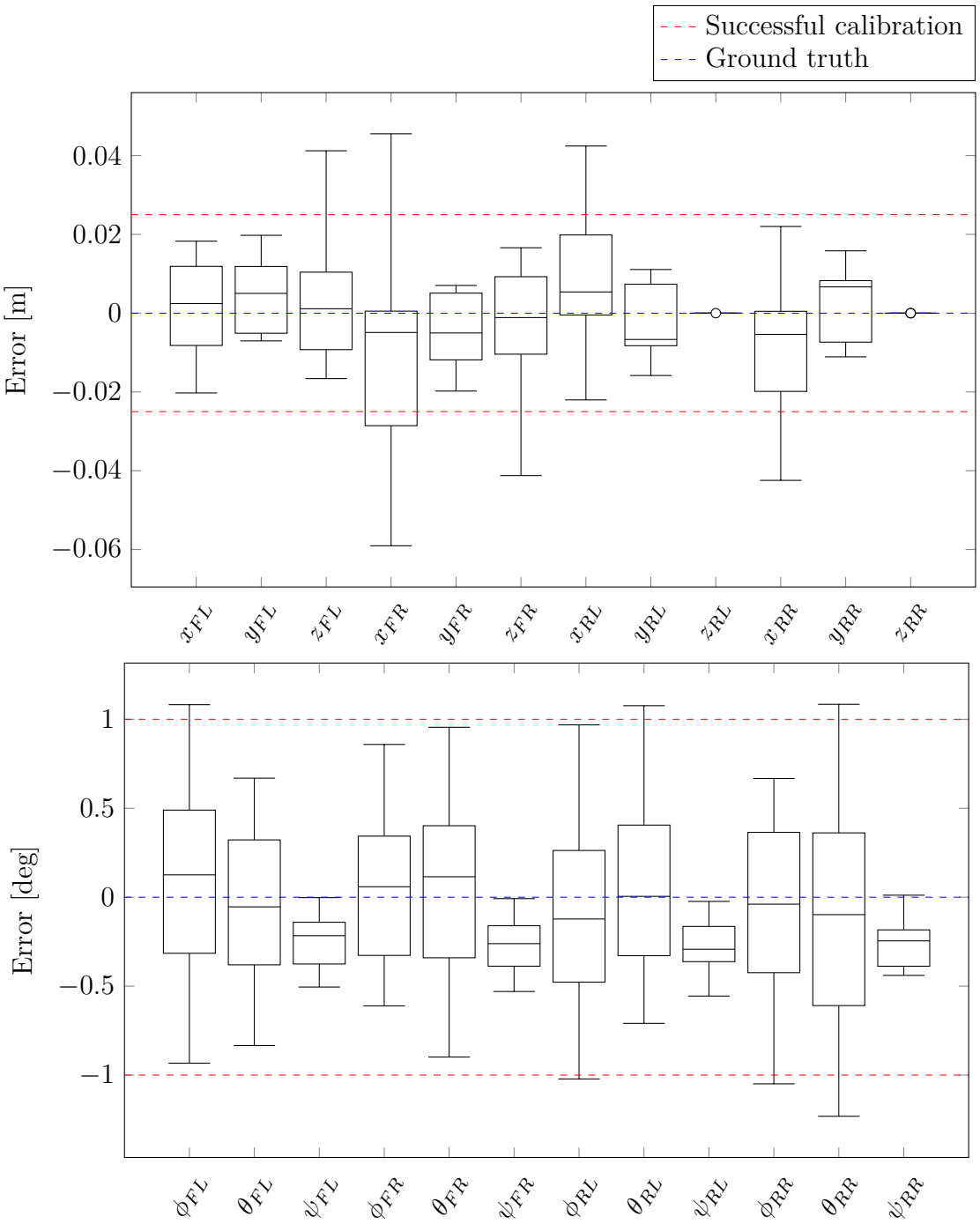


Figure A.13: Box plots of errors in translation and orientation from calibration using PSO with the medium solution space on scene 2.

GA, medium solution space

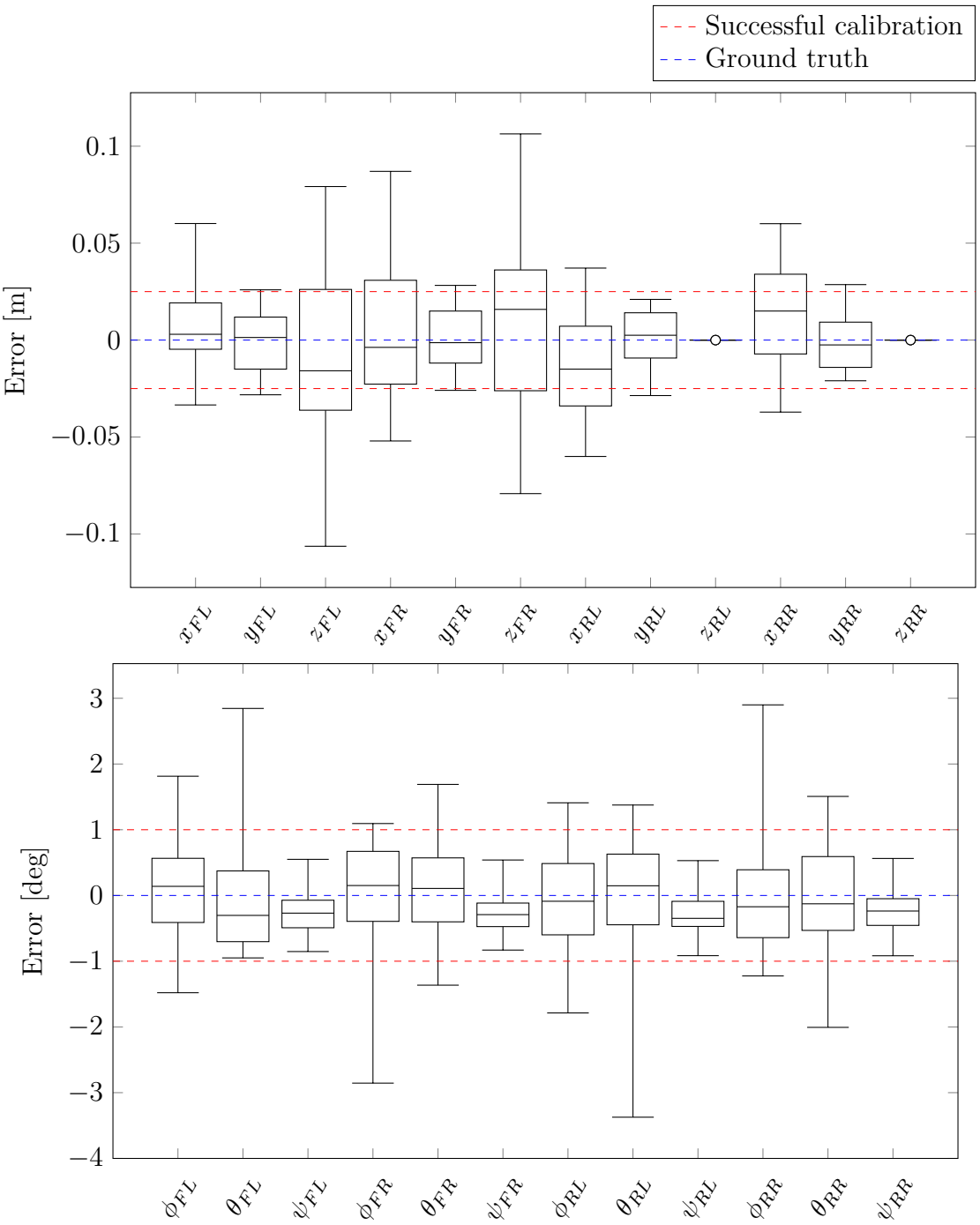


Figure A.14: Box plots of errors in translation and orientation from calibration using the GA with the medium solution space on scene 2.

ICP, medium solution space

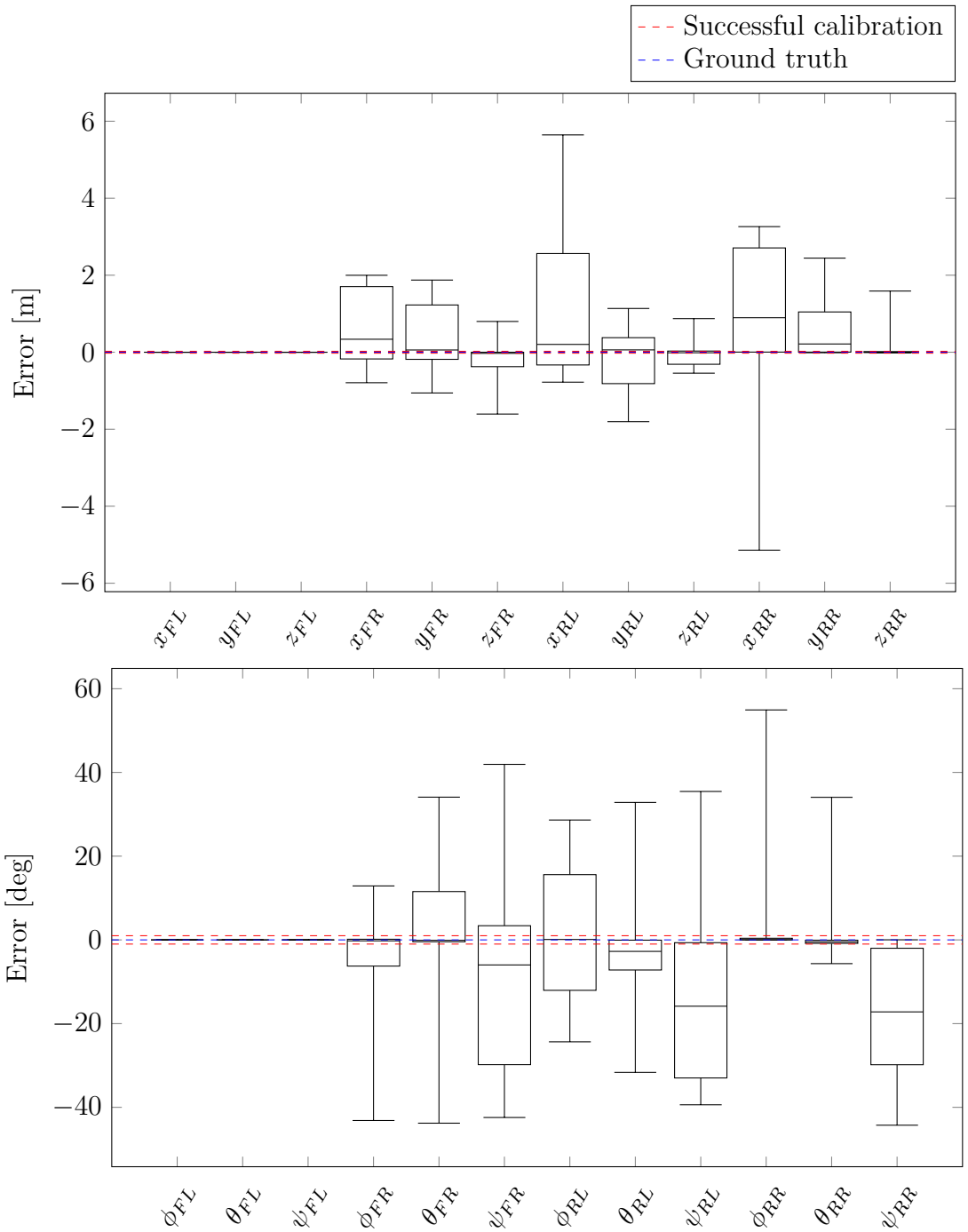


Figure A.15: Box plots of errors in translation and orientation from calibration using ICP with the medium solution space on scene 2.

PSO, large solution space

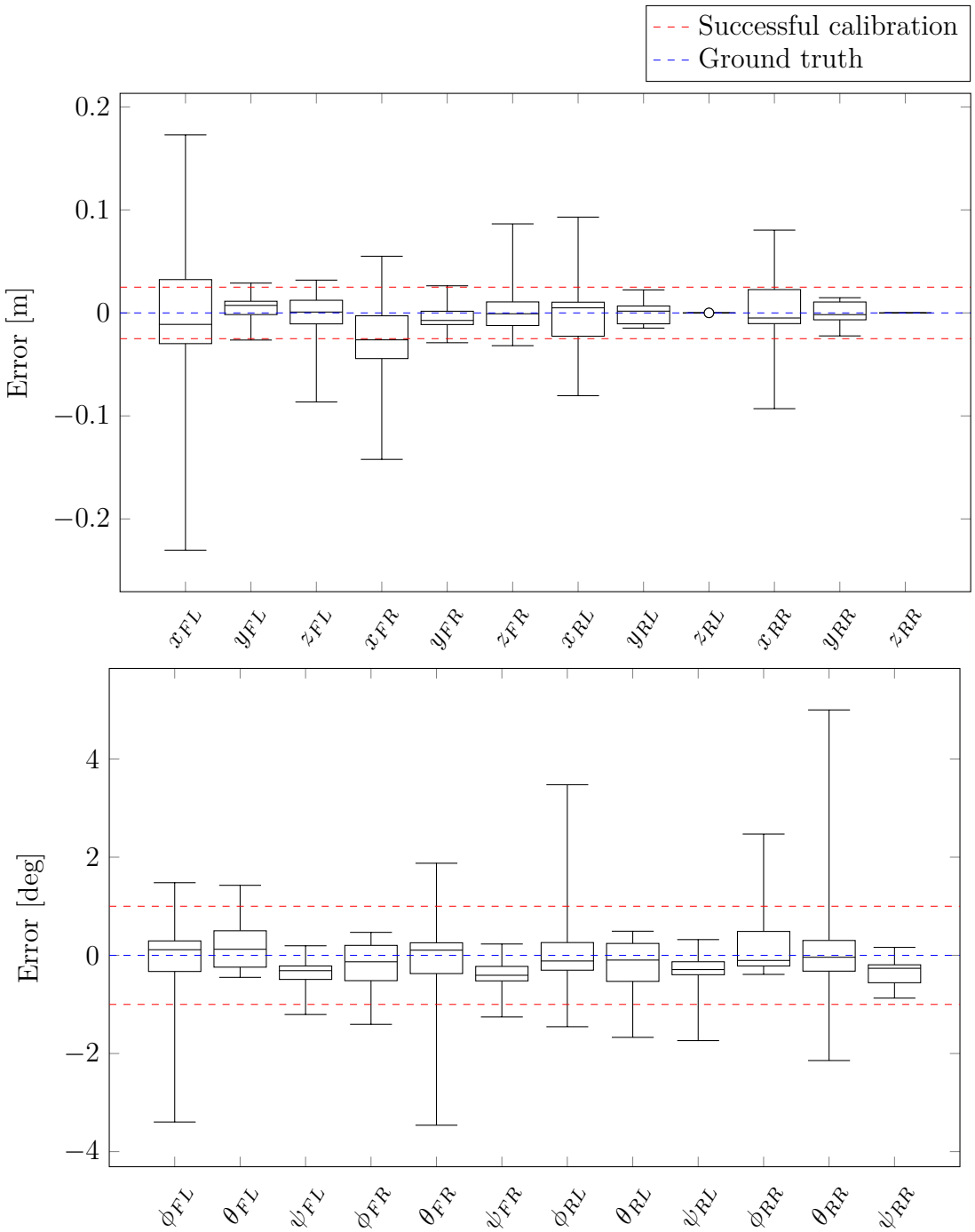


Figure A.16: Box plots of errors in translation and orientation from calibration using PSO with the large solution space on scene 2.

GA, large solution space

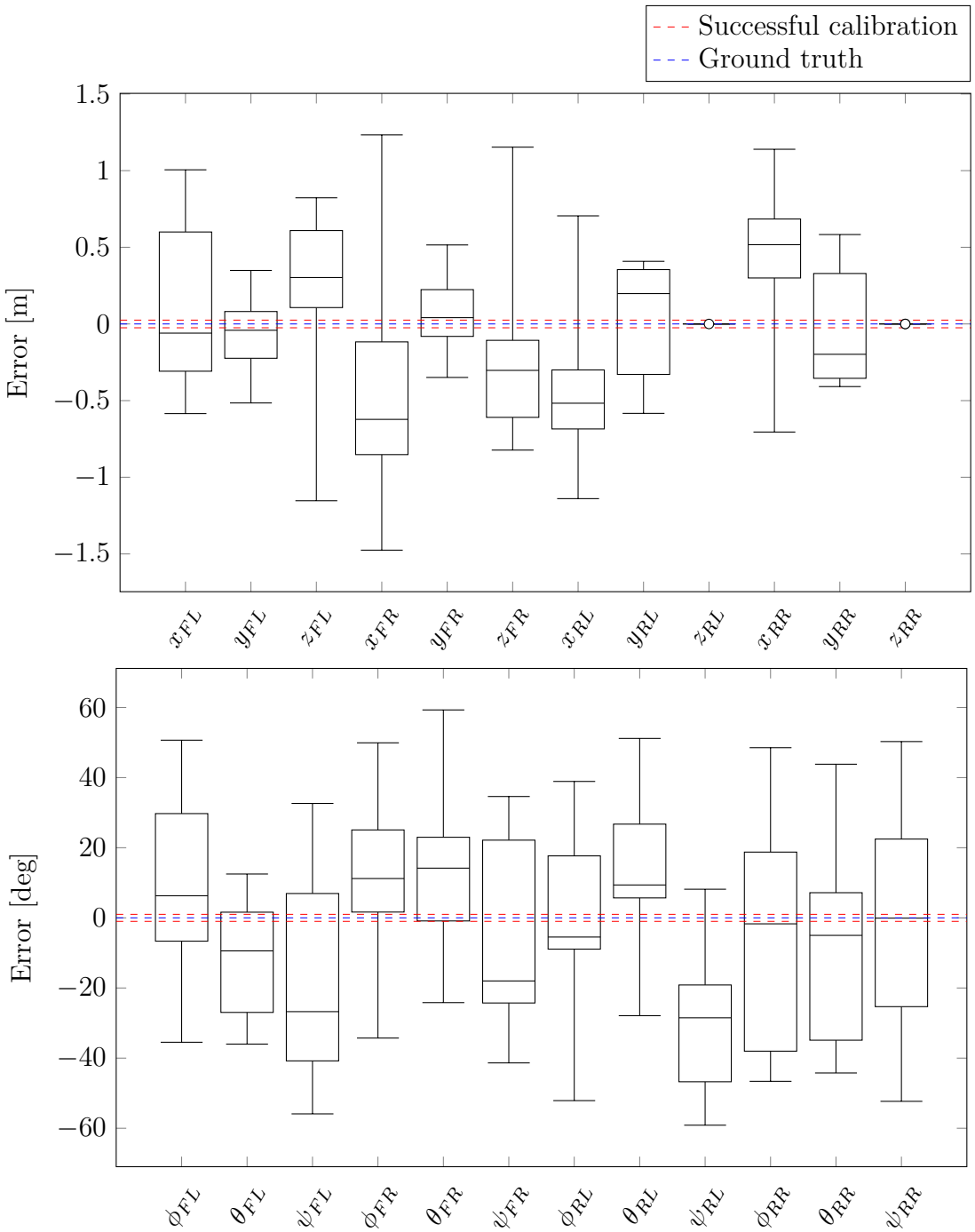


Figure A.17: Box plots of errors in translation and orientation from calibration using the GA with the large solution space on scene 2.

ICP, large solution space

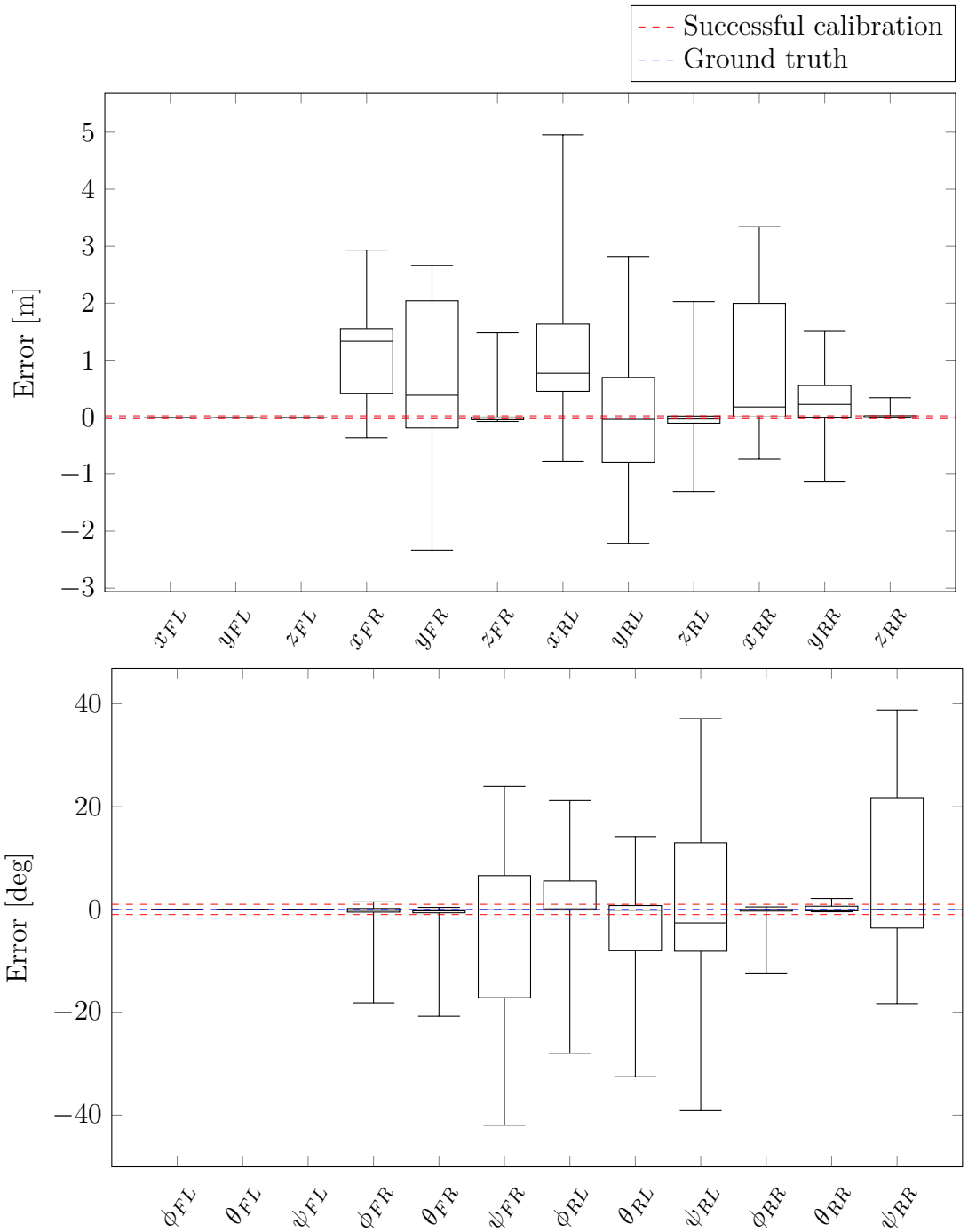


Figure A.18: Box plots of errors in translation and orientation from calibration using ICP with the large solution space on scene 2.

A.3 Scene 3

PSO, small solution space

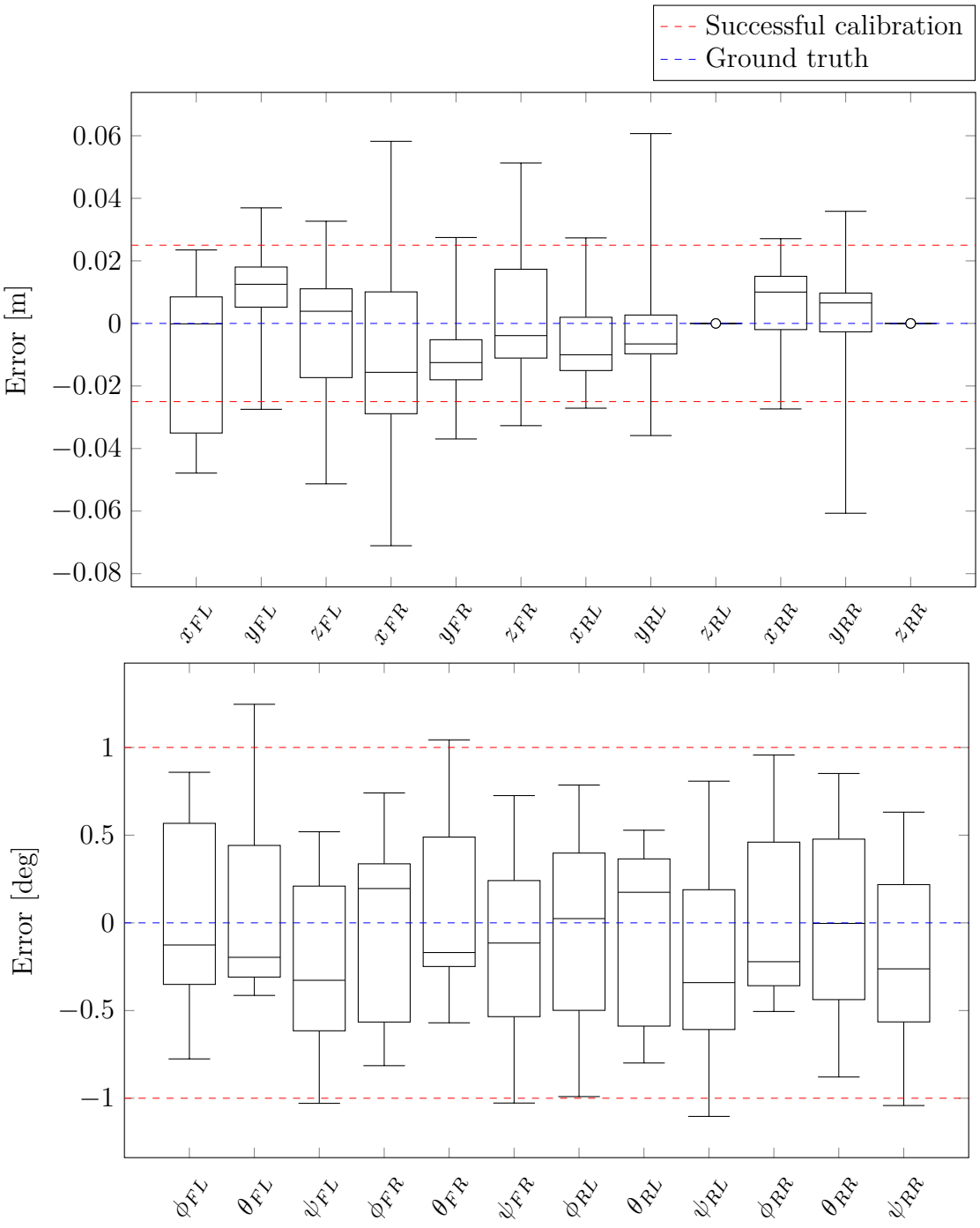


Figure A.19: Box plots of errors in translation and orientation from calibration using PSO with the small solution space on scene 3.

GA, small solution space

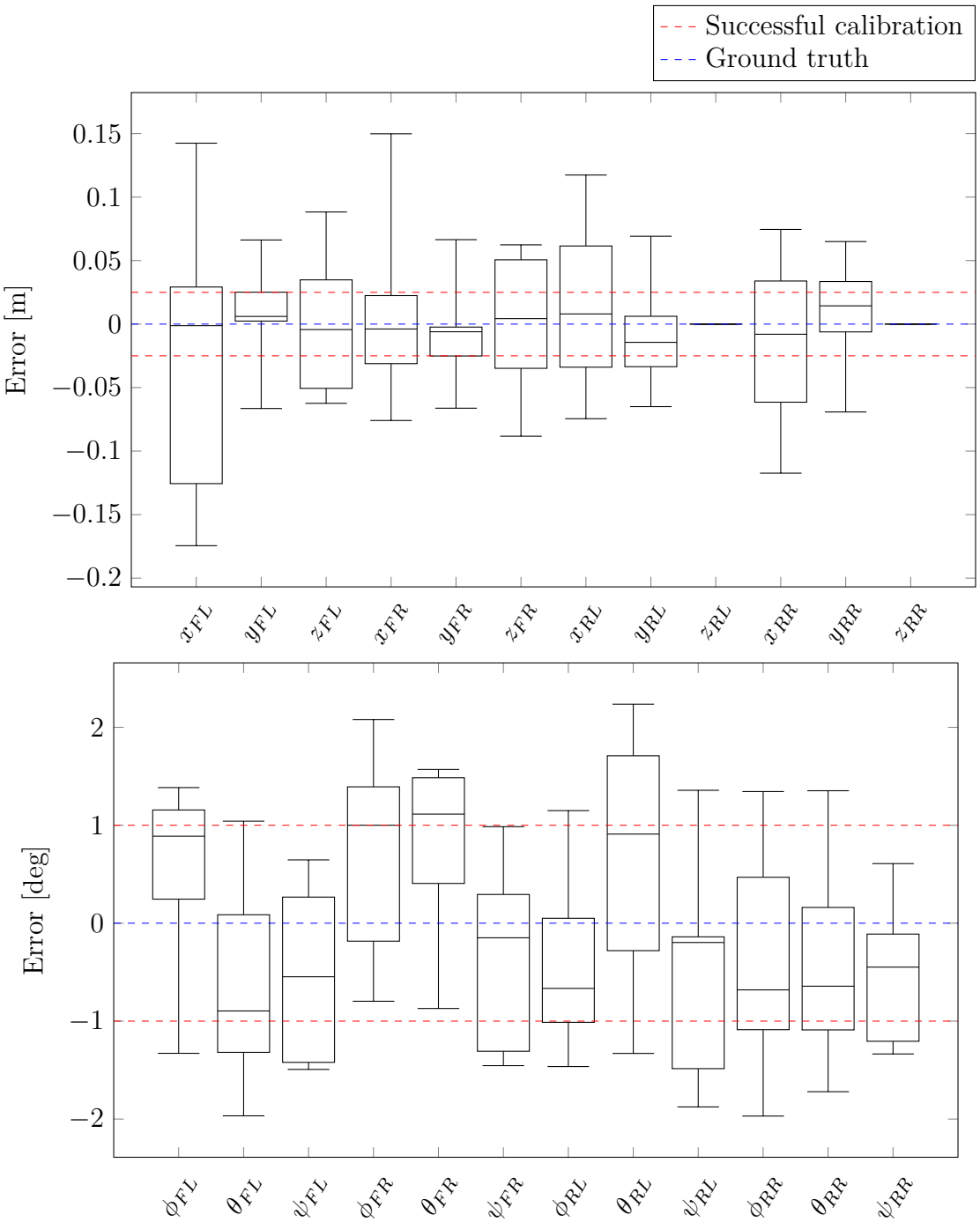


Figure A.20: Box plots of errors in translation and orientation from calibration using the GA with the small solution space on scene 3.

ICP, small solution space

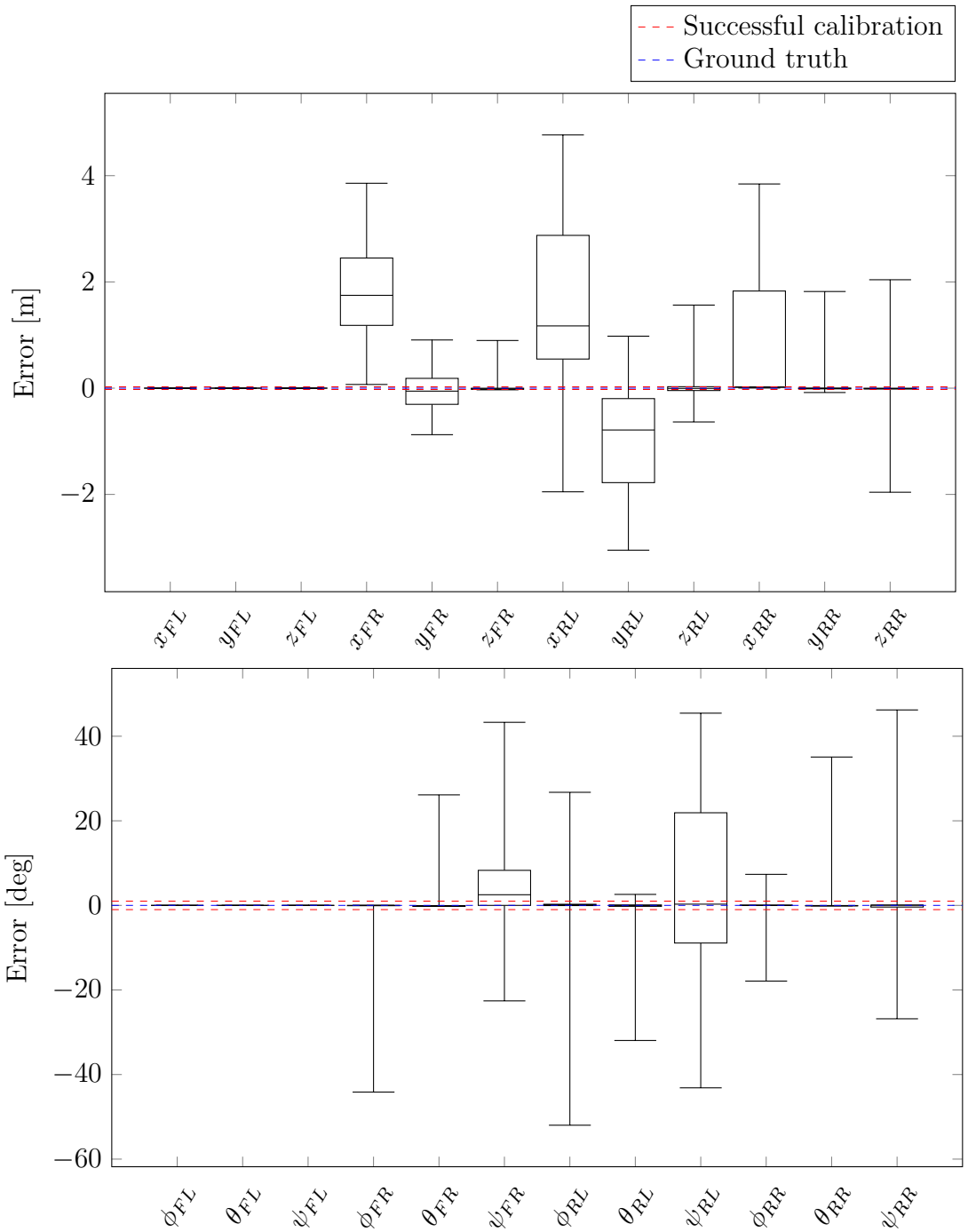


Figure A.21: Box plots of errors in translation and orientation from calibration using ICP with the small solution space on scene 3.

PSO, medium solution space

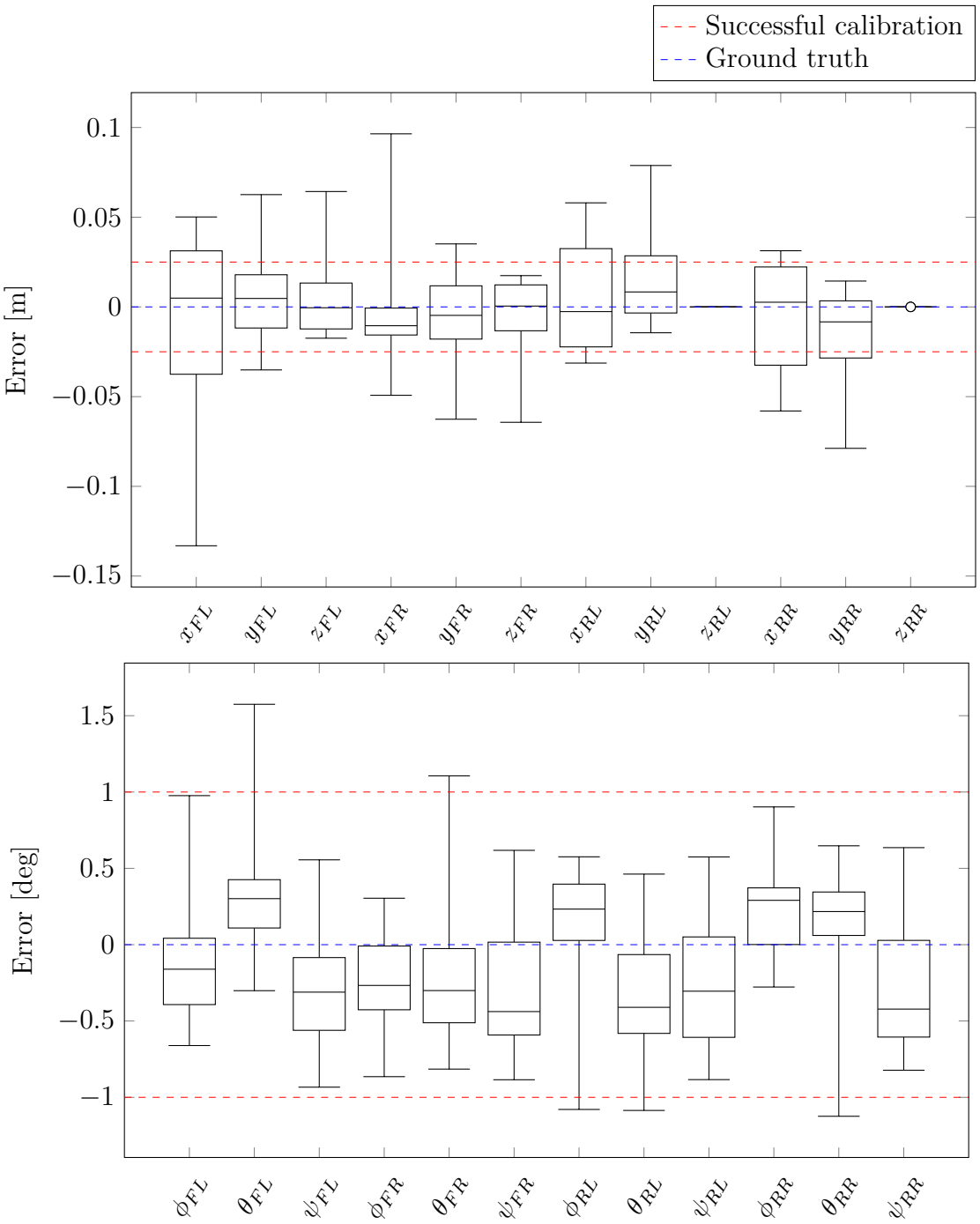


Figure A.22: Box plots of errors in translation and orientation from calibration using PSO with the medium solution space on scene 3.

GA, medium solution space

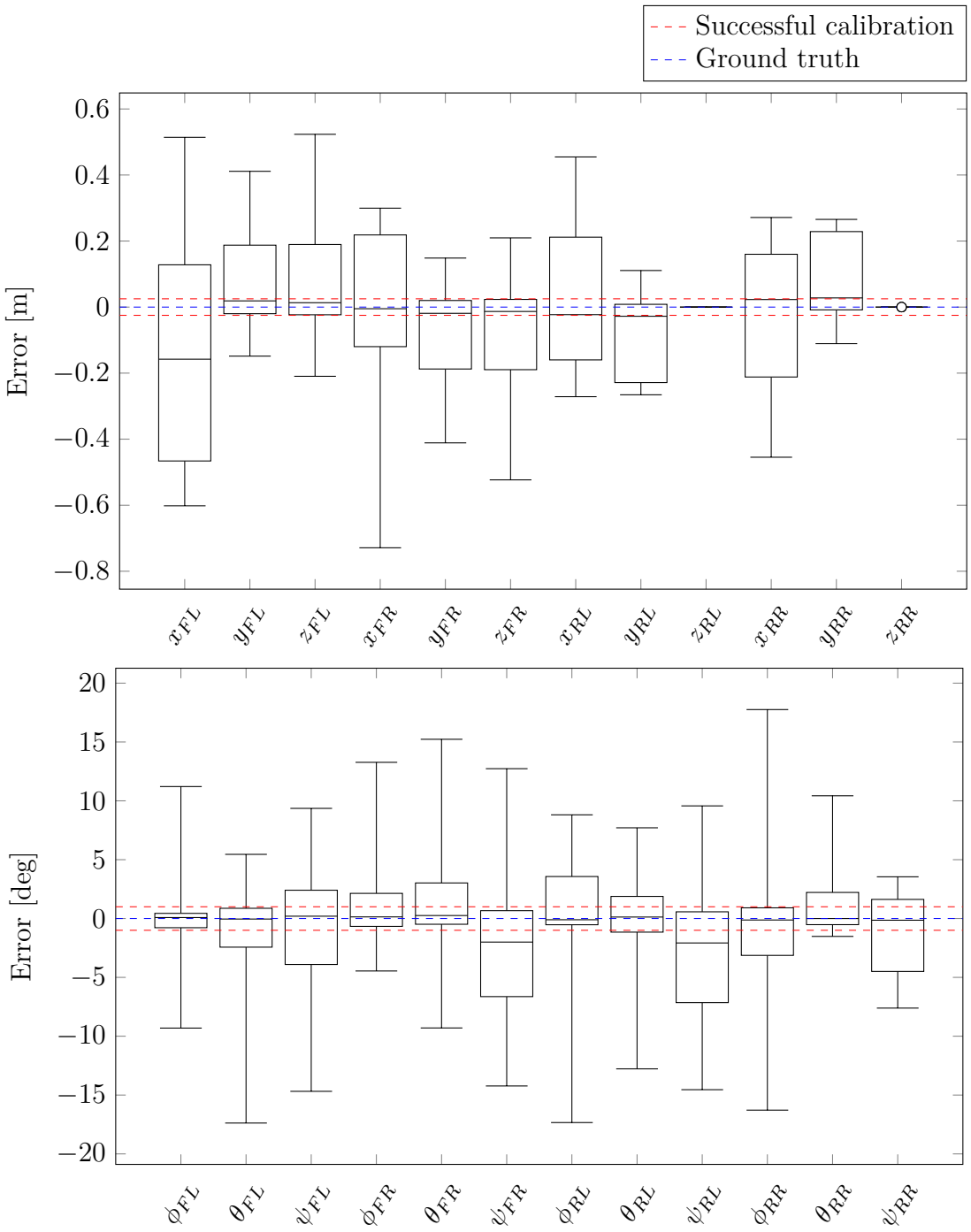


Figure A.23: Box plots of errors in translation and orientation from calibration using the GA with the medium solution space on scene 3.

ICP, medium solution space

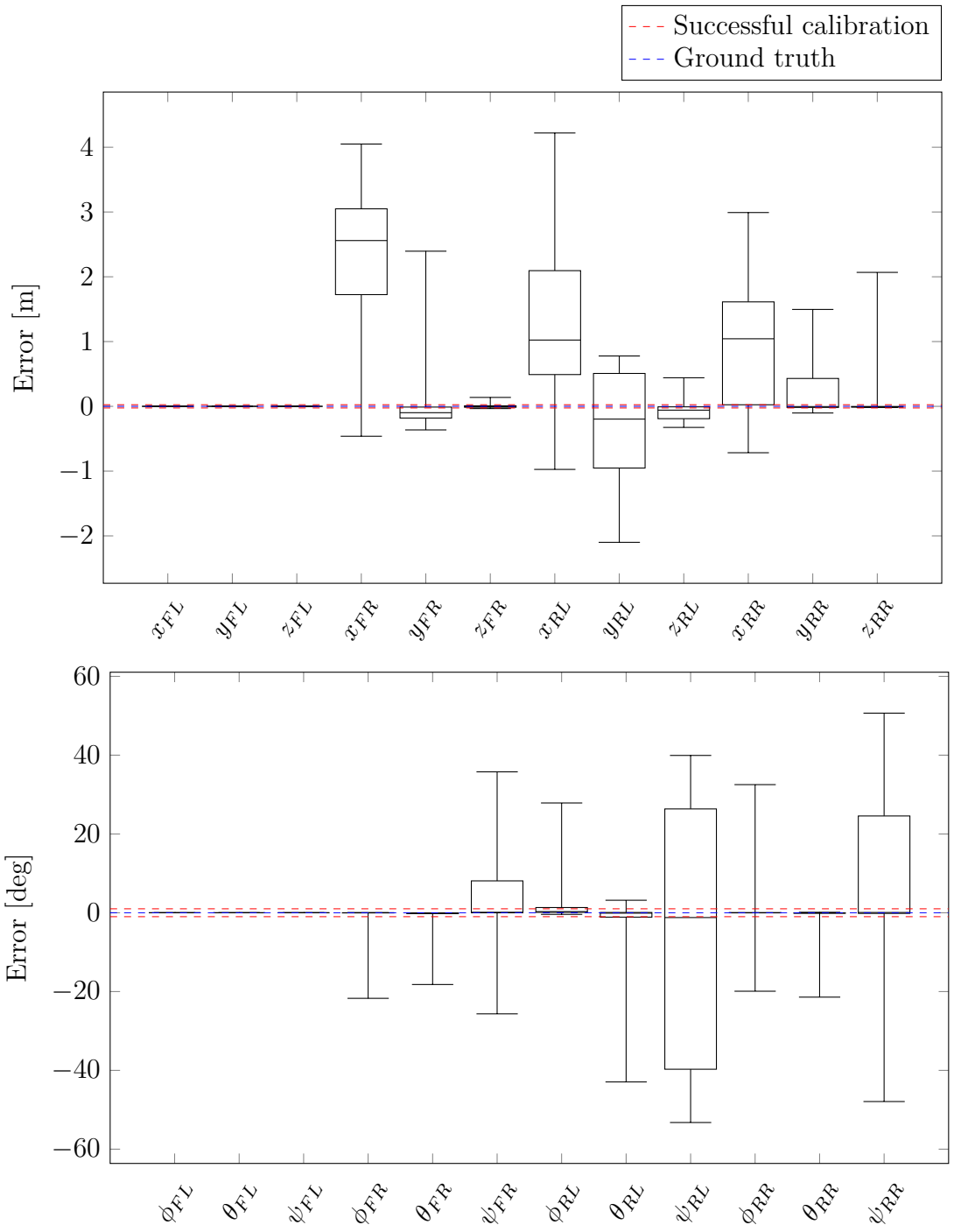


Figure A.24: Box plots of errors in translation and orientation from calibration using ICP with the medium solution space on scene 3.

PSO, large solution space

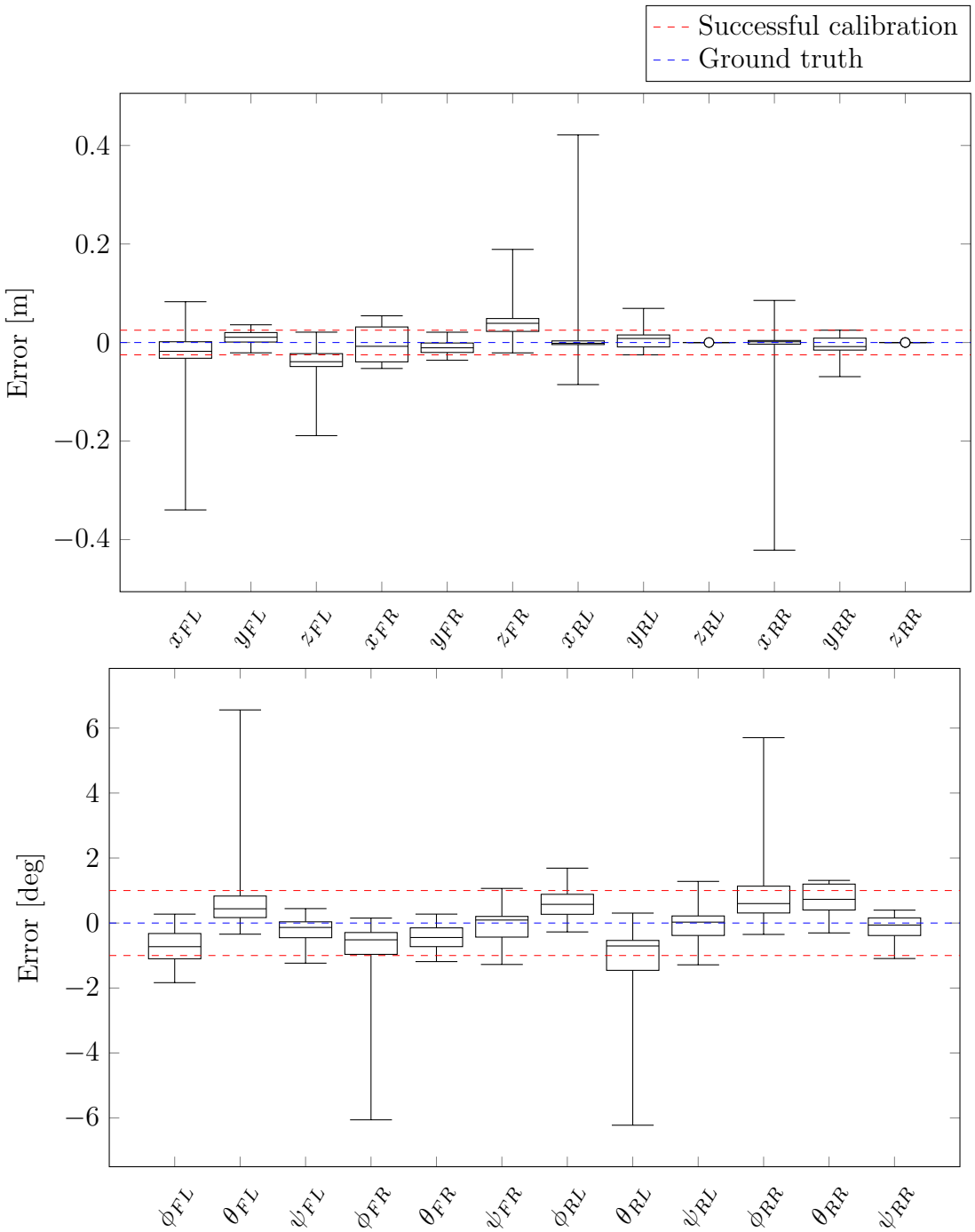


Figure A.25: Box plots of errors in translation and orientation from calibration using PSO with the large solution space on scene 3.

GA, large solution space

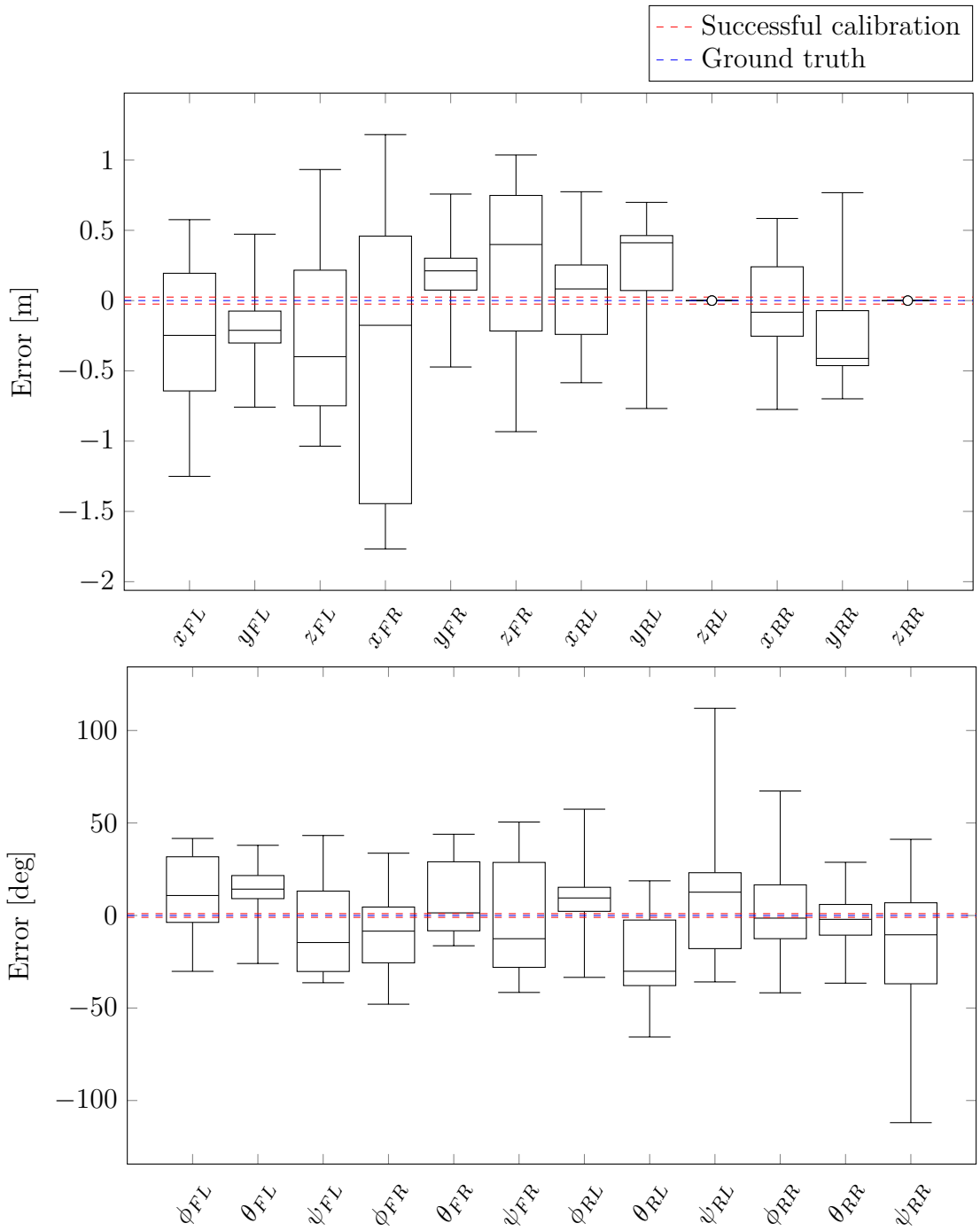


Figure A.26: Box plots of errors in translation and orientation from calibration using the GA with the large solution space on scene 3.

ICP, large solution space

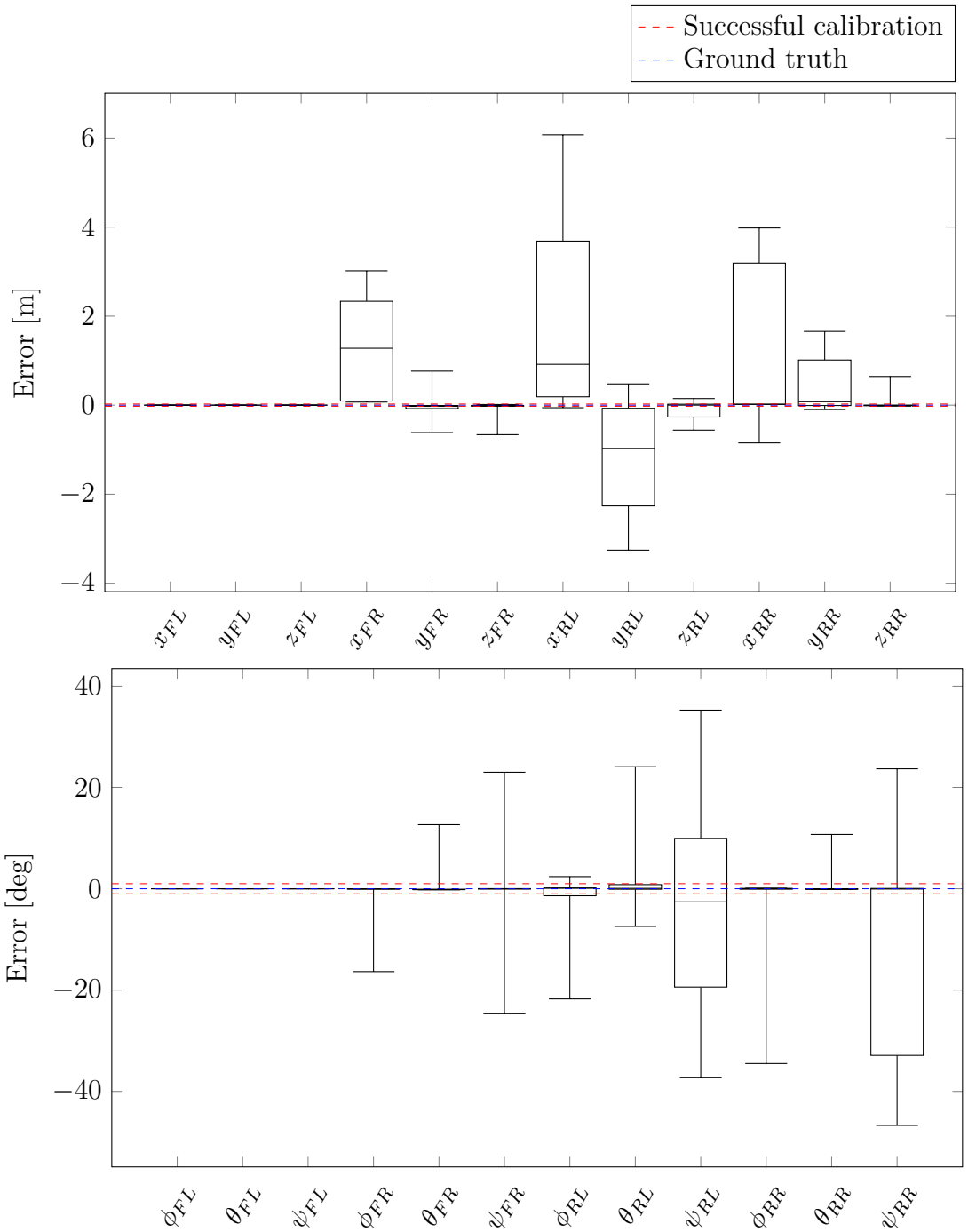


Figure A.27: Box plots of errors in translation and orientation from calibration using ICP with the large solution space on scene 3.

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

CHALMERS
UNIVERSITY OF TECHNOLOGY