University of Technology, Sydney

Faculty of Engineering and Information Technology

3DoF Vehicle Localisation from Multi-View Camera Geometry

by

Brendan Emery

Student Number: 11391265

Project Number: A17-106

Major: Mechanical and Mechatronics Engineering

Supervisor: Jaime Valls Miro

A 12 Credit Point Project submitted in partial fulfilment of the requirement for the Degree of Bachelor of Engineering

2 July, 2018

Abstract

3DoF Vehicle Localisation from Multi-View Camera Geometry

Brendan Emery - Autumn 2018

Early detection of paint defects during vehicle manufacture leads to a reduction in costs, energy consumption and the number of vehicles that fail late-stage quality control tests. Automotive manufacturers including Ford, Volkswagen and BMW, have factories around the world currently using a system to autonomously detect defects in the paintwork of vehicles. This detection is done early on during manufacturing using an array of cameras and structured lighting. These defects are identified in each camera image and projected onto a 3D model of the vehicle so that workers can locate and fix the defects. To project these defects accurately onto the model, an accurate estimate of the position and orientation of the vehicle with respect to the camera array needs to be known.

The current process finds a 2D transformation between each new image and a reference image for each camera. This transformation is applied to each defect before projecting it onto the 3D model. While the pose offset of the vehicle is a 3D transformation in the world frame, the current process can only give a 2D approximation in each camera image.

My work aims to provide an accurate, global estimate of the vehicle's lateral rotation and translation on the conveyor belt using RGB images from the camera array and a CAD model of the vehicle. The system first extracts prominent edges from the RGB images of each camera. A cost function is formulated to minimise the distance between each pixel in these edge images and the projected surface of the CAD model. The transformation estimate from this optimisation procedure is used to reproject the 3D model back into the RGB images at every iteration. The system repeats this process until a final rotation and translation estimate has been found. The system is evaluated on simulated data and has been shown to perform well past the tolerances required by the manufacturing plants.

Acknowledgements

The opportunity to work on my chosen Capstone project has been possible thanks to the collaboration between my Capstone supervisor, Associate Professor Jaime Valls Miro from the Centre for Autonomous Systems, University of Technology Sydney (UTS), and my project collaborator, Dr. Juan Ernesto Solanes Galbis from the Universitat Politécnica de Valéncia, Spain.

I would also like to thank Dr. Liang Zhao who gave me guidance and advice about various areas of the project.

Contents

List of Figures

List of Tables

Acronyms

DoF Degrees of Freedom.

ICP Iterative Closest Point.

IRLS Iteratively Reweighted Least-Squares.

LAE Least Absolute Error.

MAR Median Absolute Residual.

UTS University of Technology Sydney.

Glossary

Blender 3D computer graphics software.

Matlab Matrix based programming language useful for solving engineering and scientific problems.

Pose Position and orientation of an object.

Real camera Camera that is in the real car manufacturing plant. Currently, since we don't have real data, these cameras are also simulated in Blender. The distinction between the real and simulated camera is that the real cameras observe the vehicle with a pose offset, as would be seen in the real manufacturing plant.

Real camera image Image captured from a real camera.

Simulated camera Simulated camera inside Blender which has the same intrinsic and extrinsic parameters as one of the real cameras. These cameras always capture the vehicle without a pose offset.

Simulated camera image Image from a simulated camera.

Chapter 1

Introduction

This report is organised as follows:

Chapter 1 provides an introduction into the problem we are trying to solve and the existing method that is being used to solve it. We motivate why our proposed solution will be an improvement on the existing work.

Chapter 2 reviews the literature relevant to the project.

Chapter 3 covers the solution that we have proposed and explains how it has been implemented.

Chapter 4 presents the results of our system and demonstrates the experiments we carried out to test various parts of our system.

Chapter 5 discusses the results and also interesting aspects of the project that have been discovered through our research and work.

Chapter 6 outlines the recommendations we have for the project and concludes the report.

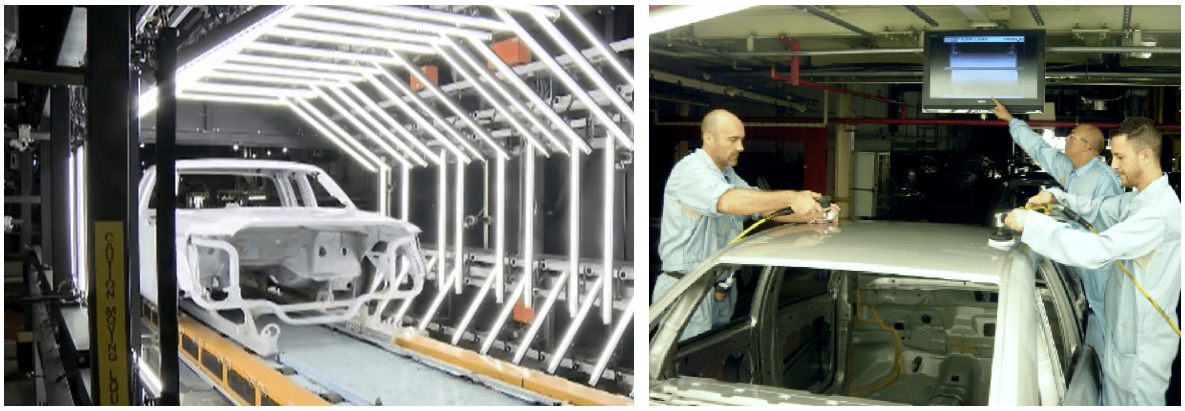


Figure 1.1: Left: Inspection tunnel in Ford Factory at Almussafes, Spain. **Right**: Workers visually fixing defects located by autonomous defect detection system.

1.1 Existing Project Overview

My Capstone is contributing to an industry project originally with Ford Motor Company in Valencia, Spain. The existing work on the project has been to develop and implement an autonomous system to detect paint defects in newly manufactured vehicles. These defects can be as small as 0.2mm, caused by issues such as undesired objects existing on the vehicle surfaces before painting, worker mistakes and temperature variations. The system developed uses a static imaging system consisting of 23 monochromatic cameras to observe uniform light patterns as they move along the length of the vehicle. Figure 1.2 depicts the camera poses in the inspection tunnel. This lighting causes shadows to appear around defects which can be identified by the vision system and highlighted on a virtual model of the vehicle, which workers then use to locate and fix the defects. This system is currently being used in 14 Ford, Volkswagen and Mercedes manufacturing plants around the world.

The conventional method of paint defect detection requires skilled operators to perform visual and tactile inspection of the vehicles in the early stages of production, which results in up to 80% of minor defects being missed. Late detection of defects leads to increases in cost, energy consumption and the number of vehicles that do not pass the late-stage quality control requirements, incurring considerable cost to the manufacturer. The automated process has improved defect detection accuracy which has helped to alleviate these problems.

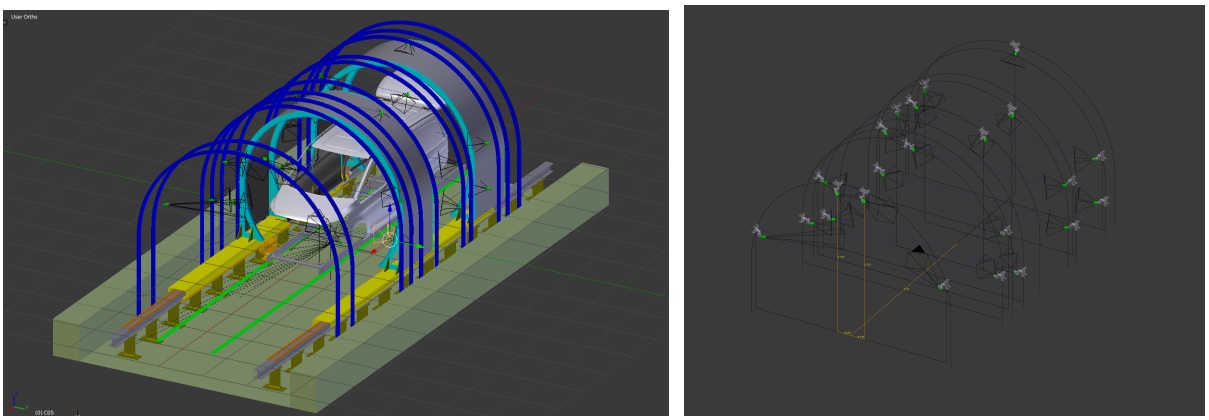


Figure 1.2: Simulated 3D Blender model of: **Left**: The entire inspection tunnel. **Right**: The tunnel frame and cameras.

1.2 Existing System

For each vehicle type that is assessed using the defect detection system, a series of reference images of the vehicle are obtained. The reference images are taken by placing the vehicle on the skid in a known pose, recording images of the vehicle with each camera and then extracting unique features from each of the images. This process is repeated for several vehicles, and the mean of the image features for each camera are taken as the reference images, such that the reference images only contain image features. In an ideal case, the reference images could be taken of the CAD model. However, since the real cameras have not been properly calibrated, we cannot accurately simulate the camera images and therefore should use the real cameras.

Once the reference images have been captured, the scanning process takes place. The process is outlined below and summarised in Figure 1.4.

1. The vehicle to be inspected is placed on a skid which is clamped to the conveyor belt. The vehicle is held in place by its weight rather than any mechanical attachments. The skid is moved into the position at which the reference images were taken and remains there throughout the duration of scanning.
2. Images of the vehicle are captured with each camera at 5MP and 15FPS as the fluorescent lights move along the length of the vehicle.
3. The images from each camera are superimposed to form a single image from each camera's viewpoint. These are the images that will be used by my project's localisation algorithm.
4. These images are passed into the defect detection algorithm, outlined in [1], which highlights the defects in each of the fused images. Figure 1.3 shows the superimposed images with highlighted defects.
5. Features are extracted from each of the images. These features are compared with the reference image for each camera and a 2D transform is calculated and applied to the

Figure 1.3: Superimposed images which consist of all images from each camera from a single scan of the light fixture.

defects in each image. It is important that this step is performed after defect detection, as the transformation of the raw images may obscure the defects.

6. The transformed defects are projected onto the 3D CAD model in Blender, see Figure 1.2.
7. 2D images of the defects are then generated based off the 3D model, which is used by workers to locate and fix the defects. The original defects are projected onto the 3D model so that new 2D images can be generated observing the vehicle from any viewpoint. This allows workers to specify the preferred viewpoints.

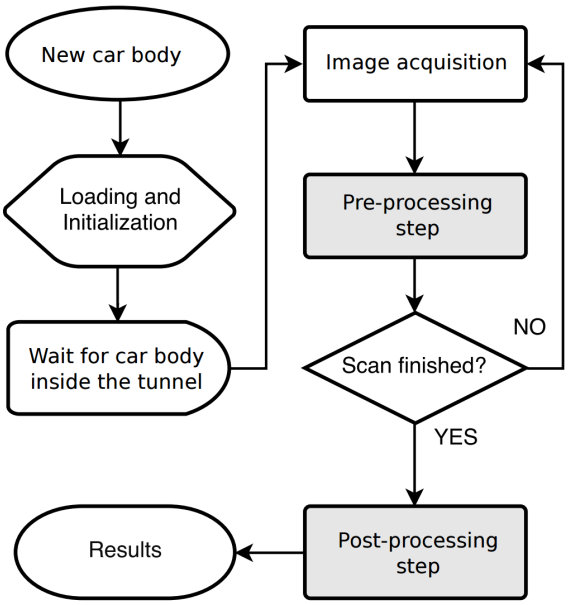


Figure 1.4: System process flowchart. Pre-processing step covers image fusion and post-processing step runs these images through the defect detection algorithm.

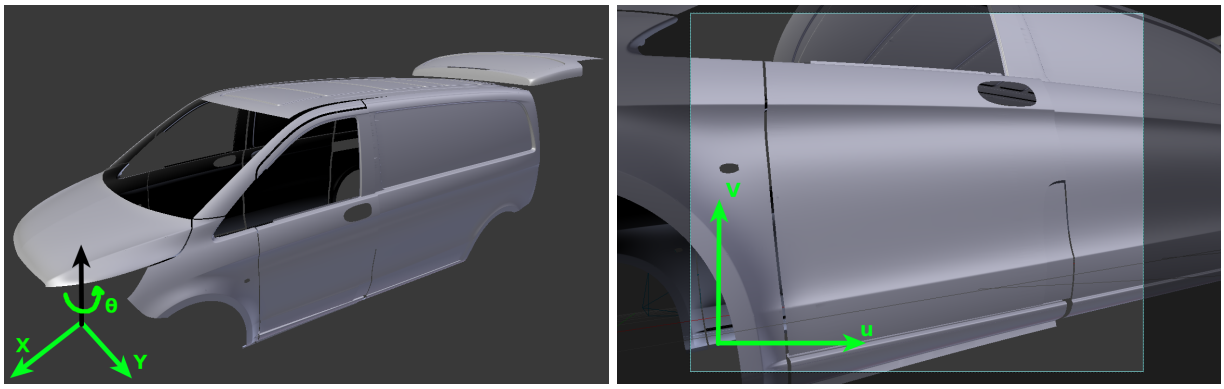


Figure 1.5: Left: Pose transformation in 3D: Translation in X and Y axes. Rotation about Z axis.
Right: Pose transformation in 2D image: Translation along horizontal image axis, u, and vertical image axis, v.

1.3 Capstone Project

Once the defects have been detected in each camera image, they need to be projected onto a 3D model of the vehicle so that workers can locate and fix the defects. To make an accurate projection, an accurate estimate of the position and orientation of the vehicle with respect to the camera array needs to be known. Since the vehicle is sitting flat on a converyor belt, we need to estimate the translation in X and Y, and the rotation of the vehicle about the Z axis. The tolerance of the current pose estimate of the vehicle is ±2cm, which means that we can assume that the offset of the vehicle will be within that range.

The existing system uses the simple vehicle localisation technique described in Sec. 1.2 to reduce the computational complexity of the pose estimation in order to satisfy the short cycle time for vehicle inspection. While this implementation is efficient, it finds a 2D transformation between each new image and the reference image for each camera individually. Since the actual vehicle offset is a 3D transformation in the world frame, the existing process can only give a 2D approximation of the offset in each camera image (See Fig. 1.5). Additionally, it ignores the fact that the vehicle pose in each camera image is correlated, since it calculates a separate transformation for each camera frame.

The aim of my project is to leverage this correlation to calculate a single transformation between the reference vehicle (i.e. the CAD model in simulation) and the actual vehicle being scanned to produce a more accurate and robust pose estimate.

Chapter 2

Literature Review

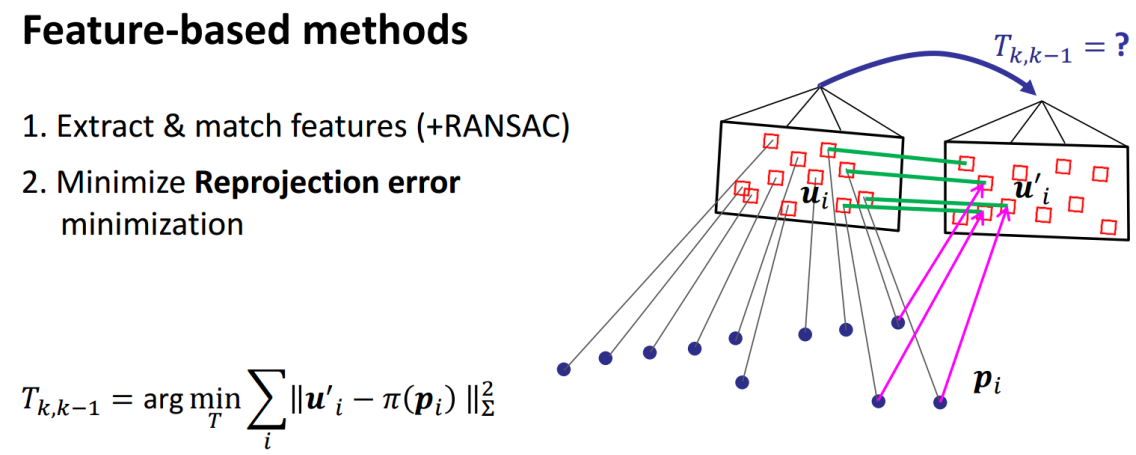


Figure 2.1: Overview of feature-based methods [2].

2.1 3D reconstruction

During Capstone A, I investigated using either feature-based or direct methods to reconstruct the vehicle using point or pixel correspondences between adjacent cameras. We could then minimise the distance between the 3D reconstruction (represented as a point cloud) and the CAD model of the vehicle. An overview of feature-based and direct methods is covered below.

2.1.1 Feature-Based Methods

Feature-based (or indirect) methods are some of the oldest and most researched groups of algorithms used for image reconstruction and camera pose estimation. Feature-based methods consist of two distinct steps. The first step is feature extraction, matching and outlier detection and the second step is camera pose and scene geometry estimation and global optimisation [3].

Feature extraction consists of extracting a discrete set of distinct features from each image. Descriptors of the local neighbourhoods around each feature are found and used to match corresponding features across multiple images. In general, corners, blobs or other distinct features are extracted, while homogeneous sections of the image are ignored. Since all data that cannot be used as feature points are eliminated, the resulting reconstruction will be sparse [4]. It is also possible to use higher dimensional features such as edge-based [5], [6] or region-based features [7]. There are several other line segment matching methods [8]–[10] which utilise obtained point matches to extract and match line segments, but in these cases, the line matching results rely heavily on previously found point matches. Li et al. propose a solution that does not rely on point matches, by finding the V-junction of lines and matches these between images [11]. However, the estimation of high-dimensional feature space is tedious and is rarely used in practice [3].

The camera poses and scene geometry are estimated by minimising reprojection error. The

Detector	Descriptor	Intensity	Rotation	Scale	Affine
Harris corner	2nd moment(s)	Yes	Yes	No	No
Mikolajczyk & Schmid '01, '02	2nd moment(s)	Yes	Yes	Yes	Yes
Tuytelaars, '00	2nd moment(s) Affine Inv.	Yes	Yes	No (Yes '04 ?)	Yes
Lowe '99 (DoG)	SIFT, PCA-SIFT	Yes	Yes	Yes	Yes
Van Gool 06	SURF	Yes	Yes	Yes	Yes
Matas, '02	MSER	Yes	Yes	Yes	?
others	others				

Figure 2.2: Comparitive table for invariance of various feature detectors and descriptors [12].

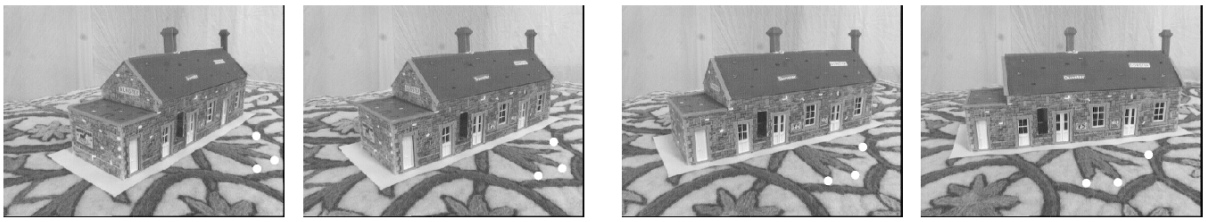


image 0 image 3 image 6 image 9

Figure 2.3: Example of wide-baseline images. The frame to frame transition in the figure has a small baseline, however, there is a large baseline between the first and last images [13].

reprojection error quantifies how closely an estimate of a 3D point matches the point's true location in an image. For each observed point, the estimated location of the point is projected into each camera image and the sum of square distances between all points and their observed locations are minimised. (See Fig. 2.1)

Feature-based methods can cope with large viewing angles and photometric distortion, which has enabled wide baseline stereo algorithms to be developed [14]. Wide baseline stereo refers to the case when the baseline between adjacent image frames being matched is considerably large. (Fig. 2.3 shows an example of this).

To obtain high accuracy, there should be a large number of features distributed evenly throughout the image [15]. If features are concentrated in a certain part of an image, feature binning can be used to ensure an even spread of features. This involves drawing a grid and keeping the k strongest features in each cell [16].

Features have a wide range of photometric invariance. The invariance arises because a discontinuity is still detectable even under large changes in illumination conditions between two images [14]. Certain features and descriptors also have a wide range of geometric invariance. Fig. 2.2 compares the invariance of various feature detectors and descriptors. Line features are invariant to projective transformations since lines are always mapped to lines under any projective transformation of an image [14].

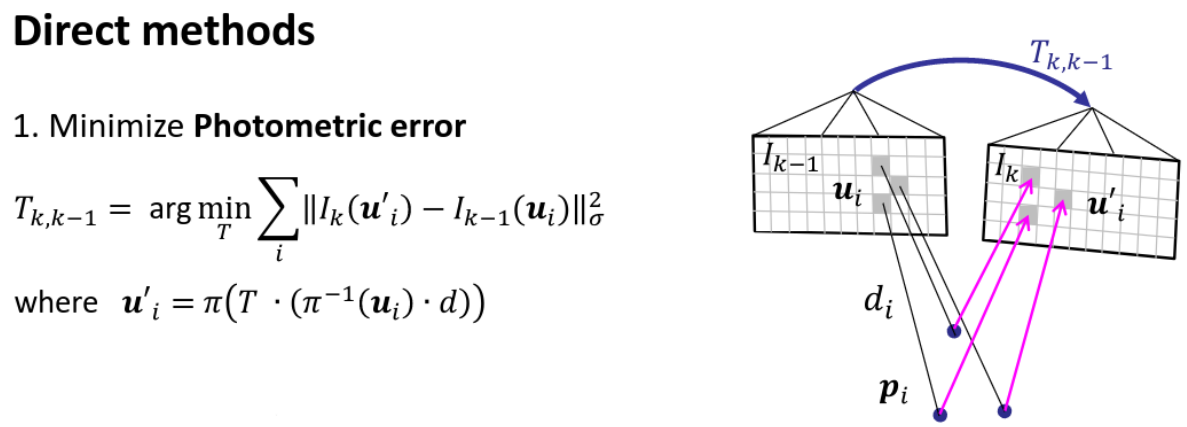


Figure 2.4: Overview of direct methods [2].

2.1.2 Direct Methods

Direct methods avoid the use of feature extraction and matching by optimising the geometry directly based on the image intensities, allowing these methods to use all the information in the image. These methods minimise the photometric error between images, which considers image brightness, brightness-based cross-correlation, etc. thereby removing the data association step required in feature-based methods. (See Fig.2.5) This generally leads to higher accuracy and robustness than feature-based methods and also allows for a denser reconstruction of the scene. This comes at the expense of higher computation times and algorithmic complexity. However, these methods are well suited for parallel processing which has facilitated broader use with the advent of high-performance GPUs.

Direct methods cannot handle outliers well, as they try to process all chosen pixels (the chosen pixels depend on the density of the method used) and implement them in the final map. This can introduce noise and subsequently outliers in homogeneous regions of the image as pixel correspondences in these regions cannot be established without additional constraints such as local smoothness [14], [17]. However, they can deal with images with no distinct feature points as long as there are sufficient image gradients along different directions in different parts of the image [18].

Many direct methods also require small-baseline stereo situations [2], [19]. They have been extended to handle image motions typically up to 10-15 percent of the image size, however, for large misalignments, an initial estimate is required [18].

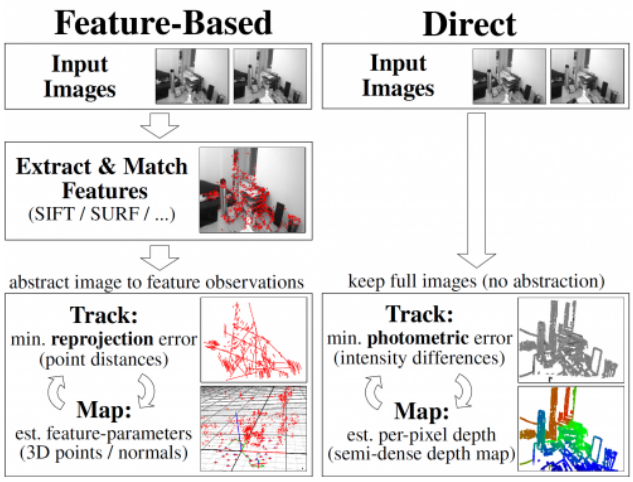


Figure 2.5: Comparison of feature-based and direct methods [19].

2.1.3 Dense v Feature-Based Methods

Image matching robustness: As stated above, direct methods can utilise more image information and deal better with images that have no distinct features as long as there are sufficient image gradients throughout the image. Feature based images on the other hand work best with an even distribution of distinct features throughout the image. However, they are more flexible at dealing with outliers, as these can be removed retroactively. They also have no need for good initialisation, and in fact can be used to initialise direct methods.

Processing time: Direct methods are more computationally expensive as they attempt to align denser regions of images. The dominant cost in feature-based methods is the feature extraction and matching process which incur a constant cost per frame. While this can be a bottleneck for visual odometry algorithms which need to work at extremely high frame rates, this will be less of an issue in the Capstone project as we have a fixed number of frames to process in the stationary case. In the moving case, the vehicle moves reasonably slowly which will allow low frame rate processing (compared with VO algorithms which often work on fast moving cars and drones).

Optimal estimation: Feature-based approaches lend themselves to bundle adjustment which is a maximum likelihood estimate of the camera parameters (intrinsics and extrinsics) and scene geometry. When minimising the distance between the back projected, reconstructed 3D point and its corresponding point in the image plane, there is evidence that these errors are independent and follow a zero mean Gaussian distribution. This confirms that the least squares is a valid maximum likelihood estimation. For direct methods, it is not straightforward to write down a practical likelihood function for all pixels. Modelling of noise and statistics is much more complicated, and simple assumptions of independence are invalid. Thus, attempting a global minimisation, treating all errors as if they were uncorrelated, will lead to a biased result [14].

Table 2.1: Table comparing feature-based and direct methods [4]

Feature-based	Direct
Can only use and reconstruct distinct features	Can use and reconstruct whole image (precision, robustness)
Faster	Slower (but well suited to parallel processing)
Slowed due to feature extraction and matching	Increasing camera frame-rate reduces computational cost per frame (higher frame rate reduces baseline between images)
Flexible: Outliers can be removed retroactively	Inflexible: difficult to remove outliers retroactively.
Robust to inconsistencies in the model/system (rolling shutter)	Not robust to inconsistencies in the model/system (rolling shutter).
Decisions (KP detection) based on less complete information	Decision (linearisation point) based on more complete information
No need for good initialisation	Needs good initialisation
Large frame-to-frame motions (wide baseline stereo)	Limited frame-to-frame motion (centremetres) without an initial estimate
20+ years of intensive research	4 years of research

2.1.4 Reconstruction Approach Effectiveness

Following the research from Capstone A, we decided to implement a feature-based approach since the direct method would not be robust in the wide-baseline scenario. During early testing of extracting point correspondences between adjacent cameras, we found that there were very few succesful feature point matches between cameras pairs. The small overlap between adjacent cameras limited the area in which we could detect feature points to match between cameras. It was then difficult to detect significant feature points within this limited area due to the poor lighting conditions in the tunnel. The wide baseline between cameras meant that the descriptors for the feature points that we did find were difficult to match between images as most descriptors were not robust to such large geometric shifts.

Outlier detection was also made more difficult due to the wide baseline setup of the cameras and because there was significant depth changes in each image (since certain cameras could see the outer and inner surface of the vehicle walls). These two properties meant that a homography could not be calculated between most adjacent images, so we could not use algorithms such as RANSAC to remove spurious point correspondences.

Therefore, we decided that a reconstruction based approach which relied on point correspondences between cameras was not a plausible solution.

2.2 Edge Matching Approach

The approach that we chose was to extract edges from the real camera images and align them with the edges extracted from the corresponding simulated cameras to estimate the pose offset of the vehicle. This new process meant that we no longer needed to rely on the overlap of adjacent cameras to find point matches, and we also didn't need to generate a 3D reconstruction of the vehicle. This process had three primary components which needed to be investigated: Detecting edges in the images, finding point correspondences between edges in the real and simulated camera images, and performing the optimisation to minimise the distance between the edges.

2.2.1 Edge Detection

According to [20], "there are two types of physical edges: (1) the set of points along which there is an abrupt change in local orientation of a physical surface and (2) the set of points describing the boundary between two or more materially distinct regions of a physical surface.". These boundaries are represented in an image by changes in image intensity. The generic algorithm for edge detection is described in [21] as follows:

- Firstly, take a grayscale image.
- Refining: Refining is used to remove as much noise as possible without affecting the true edges.
- Intensification: Apply differentiation to enhance the quality of edges.
- Threshold: Edge magnitude threshold is used to reject the noisy edge pixels.
- Localisation: Some applications estimate the location of an edge and spacing between pixels.

There are two categories of edge detection algorithms: gradient based and Laplacian based. Both methods convolve an image with a kernel based on the concept of finding the derivative of an image. Since the images are a discrete set of pixels, an approximation to the derivative must be used [22].

Gradient based methods use a discretised approximation of the first order derivative of the image. Two kernels are often used to find the gradients in the x and y directions of the image. An example of a Canny operator kernel, which is a type of gradient based edge detector, is shown in Fig. 2.6. Based off these two gradients, the gradient magnitude can be calculated which provides a measure of the strength of the edge. Since the gradient magnitude has a local maximum in a certain direction, the orientation of the edge can also be calculated [23], [24]. A threshold or set of thresholds is compared with the gradient magnitude to ensure that it is in fact an edge and not image noise.

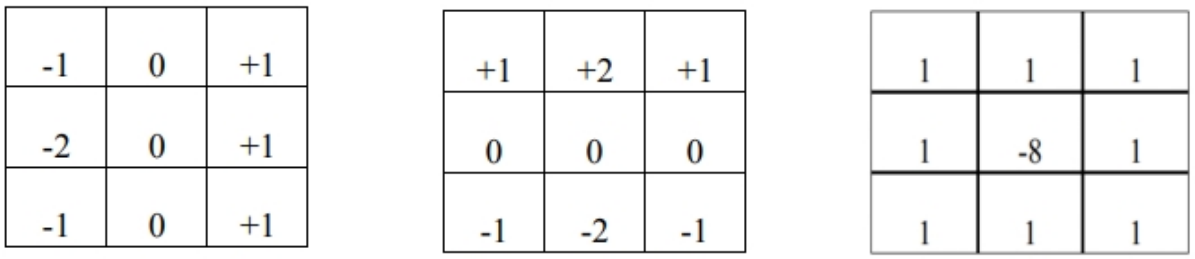


Figure 2.6: Left/Middle: Horizontal and Vertical first order derivative kernels of a commonly used gradient based edge detector. **Right**: Second order derivative kernel of a Laplacian filter.[21]

Laplacian based methods use a discretised approximation of the second order derivative of the image. Unlike the gradient, the Laplacian does not favour any particular edge orientation, so a single kernel can be used. An example of a Laplacian kernel to find the second order derivative is shown in Fig. 2.6. Since edges will have a negative gradient on one side and positive gradient on the other, the edge occurs at the peak of this curve, where the slope will be zero. Therefore, we can search for zero crossings in the second order derivatives of the image.

The Laplacian based edge detector is affected by noise in a number of ways. Firstly, the second order derivative is more sensitive to noise than the first order derivative. Secondly, noise adds variation to the regions that would have constant intensity in the noise-free image which show up as false edge contours. Thirdly, image noise can alter the location of the zero-crossing points which affects edge localisation. Therefore, a test on the zero crossing point is often used to deal with the issues caused by image noise. A common technique is to only define a zero crossing point as an edge if the gray level variance around the point is larger than a threshold. [22]

Comparisons of various edge detectors were made in [25] and [26]. They concluded that there is no single edge detector which consistently performs the best, rather, the quality of the results is heavily dependent on the image itself. They also suggested that it is difficult to predict which edge detector will work best for a given image. Therefore, the best way to choose the most suitable edge detector for our project is to test the system with various edge detectors and compare the results using each.

2.2.2 Optimisation

The goal of our optimisation is to choose the optimal values of the translation in X and Y and rotation about Z that best align the real camera edges and the projected simulated camera edges. The goal of aligning two sets of points is a very common problem in robotics and computer vision, and is often solved with the Iterative Closest Point (ICP) algorithm [27], or some variant of it. The goal of ICP is to "find the optimal rotation and translation that aligns a model shape and a data shape... where the shape could be a set of points, polylines, parametric curves, implicit curves, or implicit surfaces" [28]. A cost function, which defines the error of the alignment of the two sets, is specified and iteratively minimised by incrementally modifying the rotation and translation until the error converges on a global minimum.

Although there are numerous metrics to use in an ICP algorithm, our problem can be reduced to a problem of minimising the distance between two sets of lines, two sets of points, or a set of points and a set of lines. The point-to-point problem is solved by the original ICP algorithm [27] while the line-to-line problem has been addressed for registration of 3D point clouds in [29]. We have based our approach off a point-to-line variant of ICP which was used to solve registration of two laser scans in [30].

Importantly, ICP does not perform well when there are many points in the data set that don't correspond with any points in the model set [28]. This is largely because the ICP algorithm uses a least squares cost function in the optimisation process which assumes that the residuals are normally distributed. Any points in the data set that don't have a corresponding point in the model set will have a large error, which will be squared by the cost function. Therefore, outlier points will have an increasing impact on the optimisation. This problem and how we alleviate it is further explored in Sec. 5.1.

Further research of the literature is covered throughout the course of this document where it relates to specific components of our system.

Chapter 3

Capstone Work

3.1 Overall Pipeline

A general overview of the system pipeline is described below and a flowchart of the system is shown in Fig. 3.1. The specifics of each component of the system are covered in detail in the subsequent sections of this chapter.

1. **Image Capture**: Images are captured from the real cameras in the manufacturing plant and the corresponding cameras in the Blender simulation.
2. **Edge Extraction**: Edges are extracted from the real camera images using a Sobel edge detector. Edges are extracted from the simulated camera images using Blender's Freestyle toolbox [31].
3. **Point Cloud Construction**: The depth corresponding to each edge pixel from the simulated camera images are found. The edge pixels are then converted into real world units and transformed into the world frame.
4. **Initial Estimate**: An initial estimate for the vehicle pose (x, y and θ) is found. In the real system, these values will be taken from the existing pose estimate of the vehicle (See Sec. 1.2 for details). For our simulated tests, these values are set to 0.
5. **Point Cloud Transformation and Projection**: The simulated point cloud in the world frame is transformed using the current pose estimate of the vehicle. For the first iteration, this pose value will be the initial pose estimate. For all subsequent iterations, this value will be the optimised pose value calculated in Step 9. The transformed point cloud from each simulated camera is then projected onto that camera's image plane.
6. **Point Correspondences**: For each real and simulated camera pair, the closest two points in the real edge image are found for each point in the projection of the simulated edge image.
7. **Normal Extraction**: For each set of point correspondences (matching one simulated edge point to two real edge points), the unit normal to the line joining the two points found in the real edge image is found.
8. **Cost Function Formulation**: The cost function is formulated to minimise the distance between the sets of simulated and real edge points. The cost function incorporates the corresponding point sets from each camera pair.
9. **Optimisation** The optimal pose of the vehicle is iteratively estimated by minimising the cost function. The Gauss-Newton algorithm is used to select the best change in vehicle pose that minimises the value of the cost function at every iteration.
10. **Termination Conditions**: The system iterates by returning to Step 5 until the termination conditions have been met.

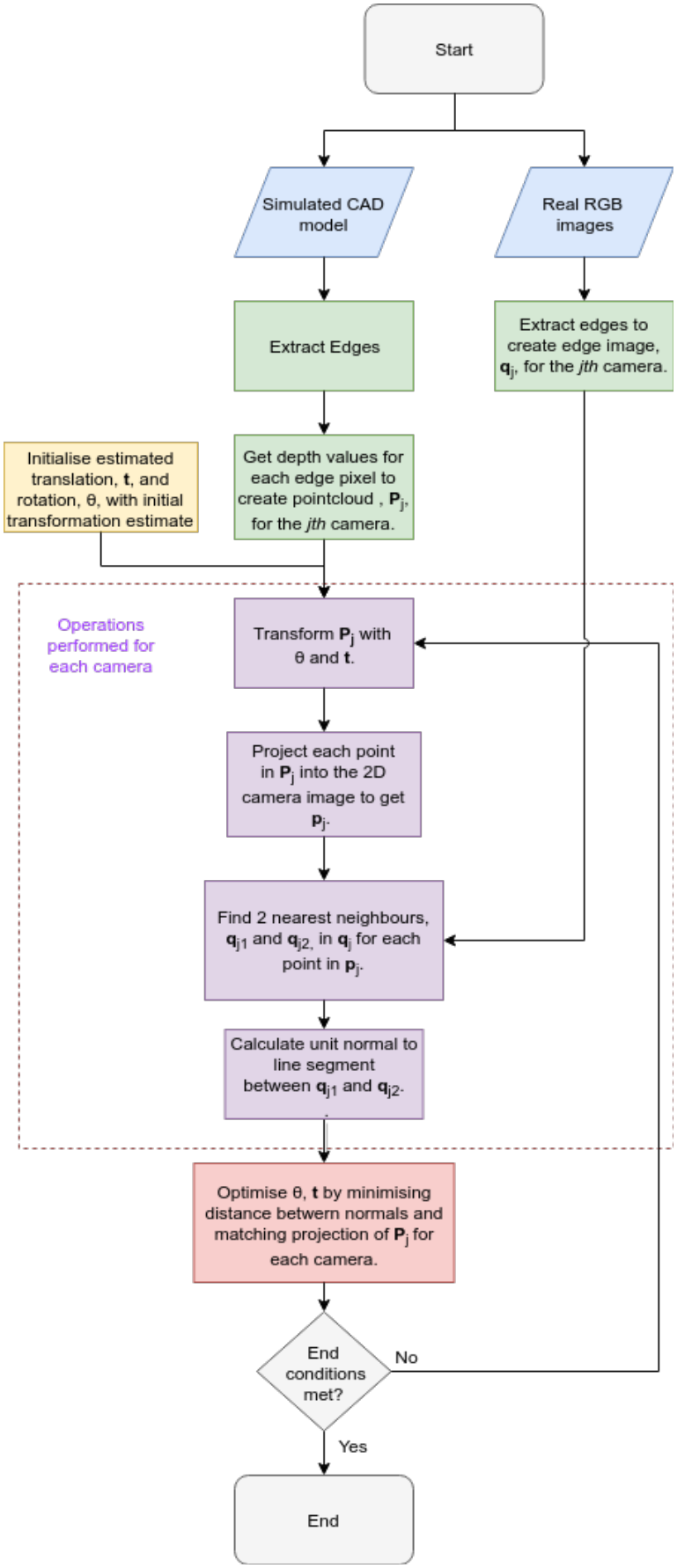


Figure 3.1: Flowchart of overall process.

3.2 Blender

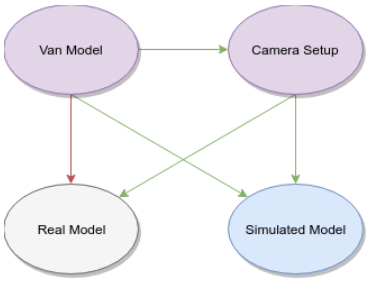


Figure 3.2: Blender file structure. Green arrows indicate linked objects. Red arrows indicate proxy objects created from linked objects.

Blender is an open-source 3D computer graphics software toolset used for a range of purposes including creating animated films, 3D models, visual effects and interactive 3D applications [32]. Blender supports realistic texture and lighting effects and also renders images using cameras modelled with realistic intrinsic parameters. This makes Blender a suitable choice for this project, as it allows us to visualise a model of the vehicle and capture images with realistic cameras. Blender also has a python API which allows most functionality to be automated using a python script.

Currently, since we don't have the intrinsic parameters of the real cameras, we are using images taken of the simulated model as our 'real images'. This allows us to specify the exact pose offset of the real vehicle with respect to the simulated vehicle so that we have an accurate ground truth. The file structure that I use for generating the real and simulated images consists of four files (See Fig. 3.2):

- **Van Model**: Contains the original model of the vehicle including the vehicle texture/paint.
- **Camera Setup**: Contains the array of 23 cameras which observe the vehicle.
- **Real Model**: Contains a link to the camera setup and a proxy object created from the link to the van model.
- **Simulated Model**: Contains a link to the camera setup and a link to the van model.

Linking an object in Blender: Creates a link to an object in another file without replicating that object. This means that any changes made to the object in the parent file is carried over to the child file. It also ensures that the child file cannot edit the properties or poses of the object. *Proxy of a linked object in Blender*: Creates a proxy of a linked object which allows the child to edit the properties and pose of an object without requiring replication of the object. Editing the proxy object in the child file has no effect on the object in the parent file.

By using a link to the camera array in the real and simulated models, certain cameras can be hidden in the camera setup file which will prevent them from being used in either of the model files. This allows the cameras to be chosen in one place to ensure that the real and simulated models are always using the exact same cameras. In my experiments, I use different subsets of

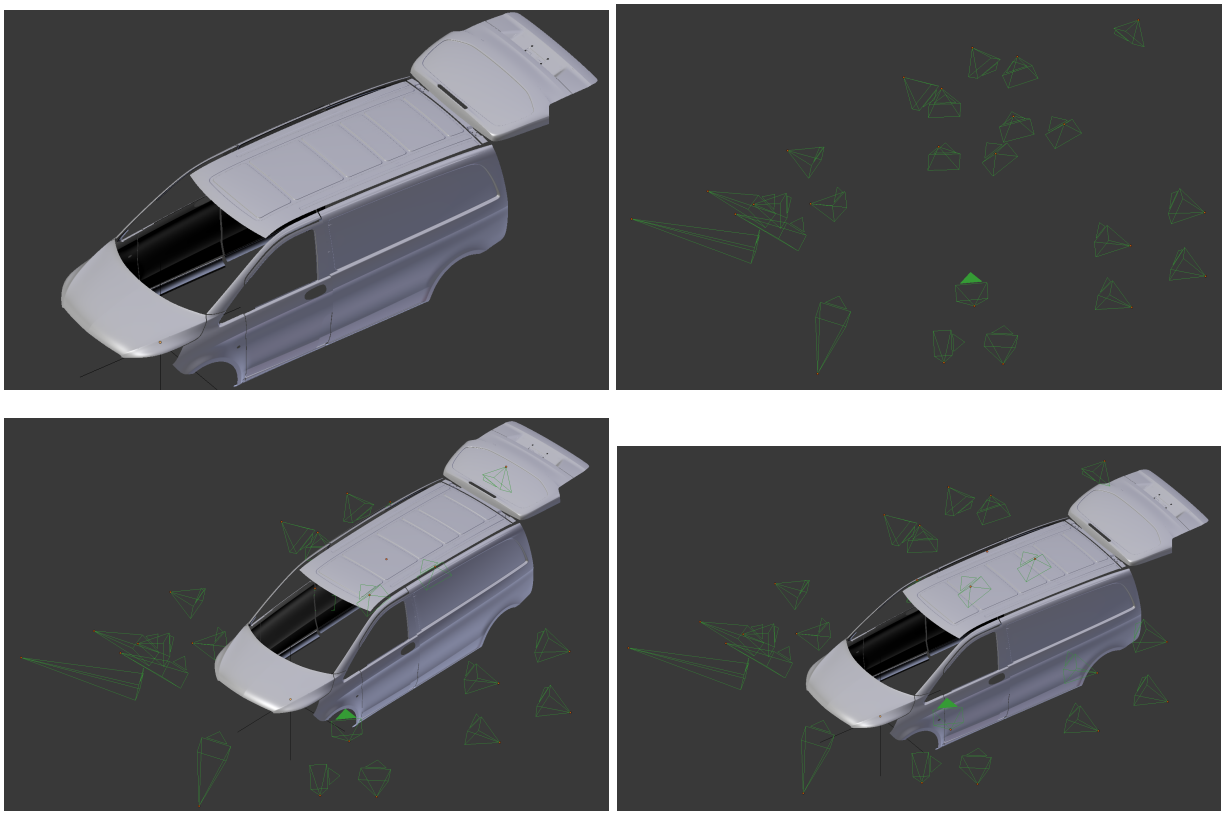


Figure 3.3: Images of Blender files corresponding to Fig. 3.2. **Top left**: Van Model. **Top right**: Camera setup. **Bottom left**: Real model that has been transformed. **Bottom right**: Simulated model.

the camera array so this functionality is vital.

A similar philosophy is used for the vehicle model. Since the model is linked in the real and simulated models, it allows us to change the model of the vehicle, the textures used, add or remove objects in the environment etc in one place. However, we have an additional requirement for the vehicle model; we need to be able to offset the pose of the vehicle in the real model file while leaving the pose of the simulated model vehicle unchanged. One option is to append the vehicle model into the real model file which copies all of the vehicle's properties and allows the child file to make changes to the model. However, this replicates the vehicle which can cause issues with version control systems since any minor change to the vehicle pose will require the entire file including the model (which is ~40MB) to be reuploaded. A solution to keep the file size small while also allowing the vehicle model to be imported is to add a link to the vehicle model and then create a proxy of that linked object.

I use a python script in the simulated and real model files to automate the linking process to construct the objects in the environment. I then use a second script which generates the rendered images from the models and saves them to a directory for use in the optimisation. The rendered images from both models are 1036x819, 8-bit grayscale images. This image resolution is half that of the real images taken from the manufacturing plant to reduce rendering time in Blender.

The images from the simulated model are processed using Blender's Freestyle toolbox which generate edges based on the rendered images. Comparisons of the edges extracted using the Freestyle toolbox and the Sobel detector is discussed in Sec. 3.3

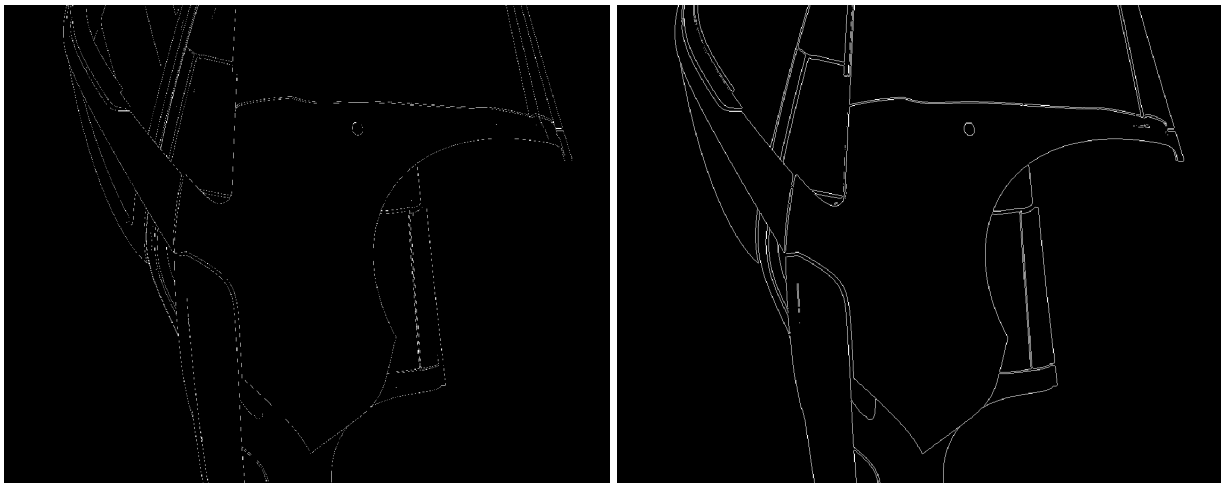


Figure 3.4: Left: Edge image produced using Blender's Freestyle toolbox. **Right**: Edge image produced using Sobel edge detector.

3.3 Edge Detection

We use two different methods for detecting the edges in the simulated camera images and the real camera images. The justification for and comparison of these methods is covered in Sec. 5.3.

- **Simulated Camera Images:** For the simulated camera images, we use the Blender Freestyle Toolbox which contains various options for edge detection. The method currently used detects the edges from the 2D rendered image of the Blender simulation by identifying contours in the image. However, the primary reason for using the Freestyle toolbox is that it allows us to manually select edges on the 3D vehicle model to show up in the final image. This functionality can be used to manually select every edge in the final image or used in conjunction with the regular detector to add any edges that were missed.
- **Real Camera Images:** For the real camera images, we use a Sobel edge detector with a different threshold for each camera. The Sobel detector finds edges at the points where the image gradient is at a maximum by convolving the image with the Sobel approximation of the first order derivative.

A comparison of the edge images produced using the two methods is shown in Fig. 3.4. Although the Freestyle toolbox does not produce continuous edges, this hasn't significantly degraded the accuracy of the optimisation. In fact, by reducing the number of points it has reduced the computation time of the algorithm. If we want to increase the number of points along the edge, we can manually select the edges using the Freestyle toolbox.

The thresholds used for the Sobel detector have been manually chosen for each camera. Since the tolerance of the system is approximately ±2cm, the images from each camera are going to

look similar regardless of the pose offset of the vehicle. Therefore, we can select thresholds using an image from each camera and be reasonably confident that those edges will show up for any pose offset of the vehicle within the specified tolerance. The chosen Sobel thresholds for the first 12 cameras (this is the set of robust cameras found in Sec. 4.1.1) are shown in Table. 3.1.

Table 3.1: Table showing sensitivity threshold used for Sobel edge detector for each camera in the set of robust cameras.

Camera	**Sensitivity Threshold**
1	0.09
2	0.075
3	0.055
4	0.08
5	0.05
6	0.03
7	0.04
8	0.05
9	0.035
10	0.055
11	0.035
12	0.06

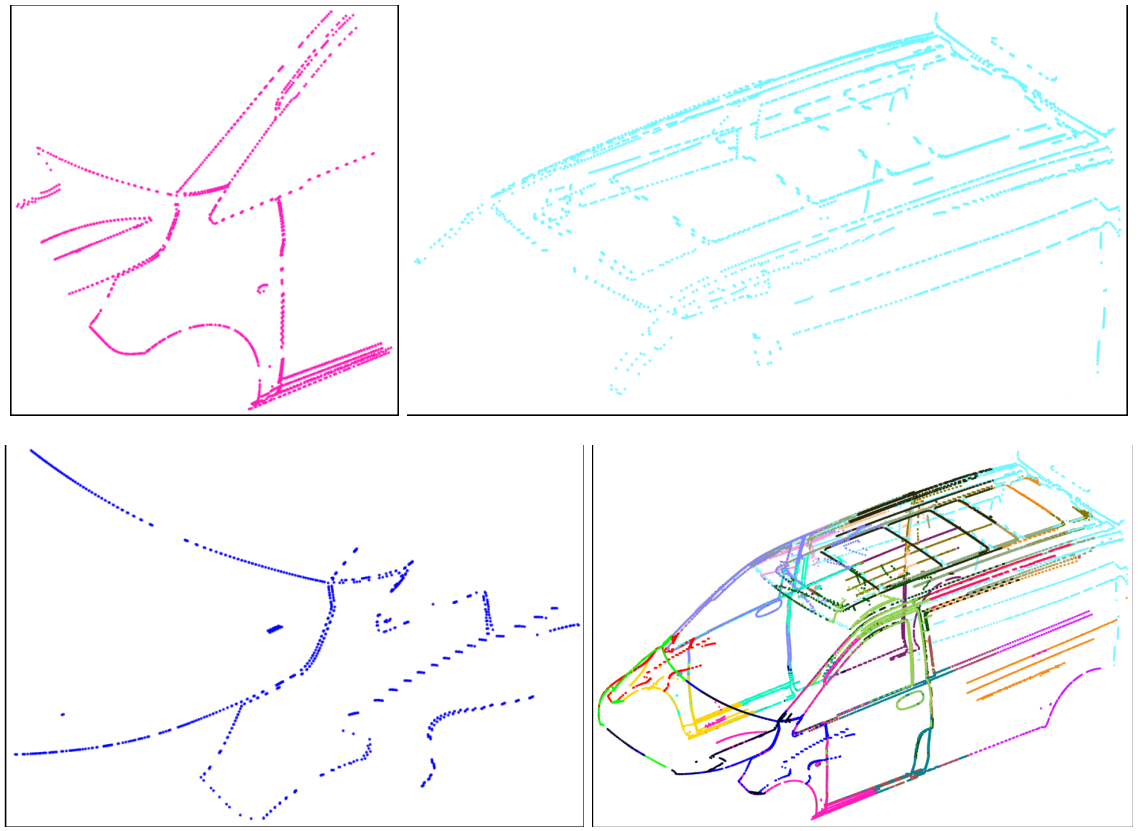


Figure 3.5: Top Left, **Top Right**, **Bottom Left**: Pointclouds created from individual cameras. **Bottom Right**: Pointclouds from each camera with respect to the world frame.

3.4 Point Cloud Construction

The goal of this project is to estimate the 3D pose of the vehicle in the world frame. Therefore, we need a way of applying 3D transformations to the edges in our simulated camera images, so that we can align them with the edges in the real camera images. To this end, we construct a pointcloud in the world frame of the edge pixels from each each simulated camera image.

Given the 2D edge image and corresponding depth image from each simulated camera, we find the distance from the camera to each edge pixel. We now have a set of n points, $\{[u_1, v_1, z_1]^T, \ldots, [u_n, v_n, z_n]^T\}$ where u_i and v_i are the pixel coordinates of the ith point in the image frame and z_i is the distance from the camera to the ith point in real world units in the camera frame.

We now convert u_i and v_i from image coordinates into real world units to get, ${}^{cam}\mathbf{P}_i = [x_i, y_i, z_i]^T$. The conversion is given by:

$$\begin{aligned} x_i &= \frac{(u_i - p_x) \cdot z_i}{f} \\ y_i &= \frac{(v_i - p_y) \cdot z_i}{f} \end{aligned} \tag{3.1}$$

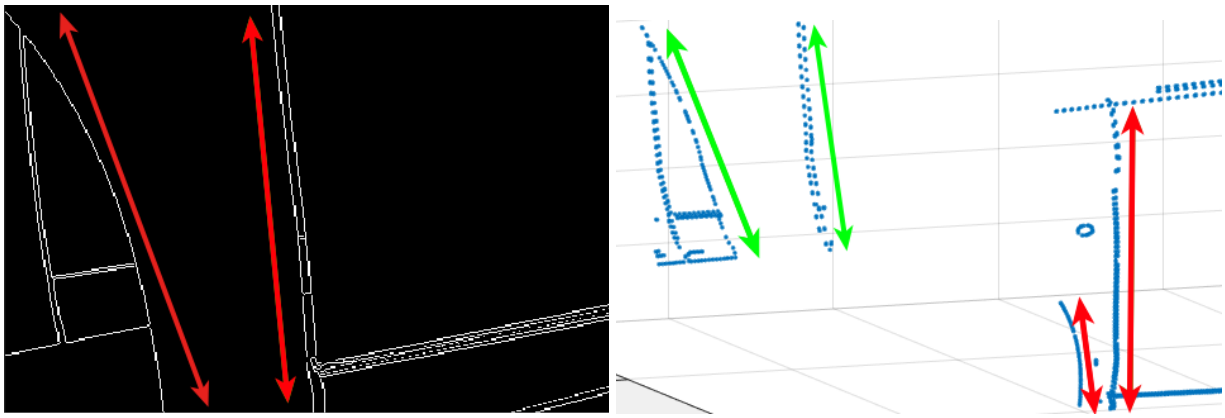


Figure 3.6: Left: Edges detected on front face of vehicle indicated by red arrows. **Right**: Correct points of corresponding edges in pointcloud shown in red and incorrect edge points shown in green.

where, $(p_x, p_y)^T$ is the principal point of the camera and f is the focal length of the camera.

Since the edge detection is not perfect, we perform 2 filtering steps:

- If an outer edge of the vehicle from the current viewpoint is detected, there will occasionally be pixels on that edge which do not lie on the vehicle. Blender assigns depth values of 1^{10} when there is no model surface at a pixel, so we search for these depth values and remove the points.
- If an edge on the closest face of the vehicle is detected, there will occasionally be pixels on the edge which lie on the interior face of the vehicle due to inaccuracies in the edge detection. This will cause significant depth discontinuities for the pixels along that edge as shown in Fig. 3.6. We manually find a reasonable thresholding distance for each camera to limit the depth of points observed from that camera. We search for points with depth values greater than this threshold and remove them.

We now need to convert the remaining points from the camera frame into the world frame. The extrinsics of the camera are given by a known transformation matrix, ${}^{World}\mathbf{T}_{Cam} \in \mathbb{R}^{4x4}$. Therefore, the conversion is:

$$ {}^{World}\mathbf{P}_i = {}^{World}\mathbf{T}_{Cam} \cdot {}^{Cam}\mathbf{P}_i \tag{3.2} $$

Since the camera coordinate systems used in Blender and Matlab are rotated about the X axis by 180^o with respect to each other, this transformation becomes:

$$ {}^{World}\mathbf{P}_i = {}^{World}\mathbf{T}_{cam} \cdot {}^{Blender\_Cam}\mathbf{T}_{Matlab\_Cam} \cdot {}^{Cam}\mathbf{P}_i \tag{3.3} $$

where, ${}^{Blender\_Cam}\mathbf{T}_{Matlab\_Cam} = Rot_x(180^o)$.

3.5 Point Projection and Transformation into world

In order to find nearest neighbour correspondences and then to minimise the distance between these correspondences in the optimisation, we need to be able to transform each point in the simulated point cloud, generated in the previous section, according to the current pose estimate. We then project each point into the image frame so that it can be compared with the edge points from the real camera.

We take a 3D point from the simulated point cloud, ${}^{World}\mathbf{P}_i^0 = [X,Y,Z,1]^T$, where the superscript 0 indicates that the point has not been transformed. We want to apply a 3D transformation and then find its 2D projection into the image frame of camera c, to give $\mathbf{p}_i = [x,y,b]^T$. The transformation and projection is given by:

$$\mathbf{p}_i = \mathbf{K}_c({}^c\mathbf{R}_w|{}^c\mathbf{t}_w)\mathbf{T}(\theta,\mathbf{t}){}^{World}\mathbf{P}_i \tag{3.4}$$

where ${}^c\mathbf{R}_w \in SO(3)$ and ${}^c\mathbf{t}_w \in \mathbb{R}^3$ are the rotation and translation of the world frame with respect to the camera frame, respectively. $\mathbf{T}(\theta,\mathbf{t}) \in \mathbb{R}^{4x4}$ is a transformation matrix containing the translation and rotation of given by the current pose estimate. $(\mathbf{A}|\mathbf{B})$ is a horizontal concatentation of two matrices with an equal number of rows. $\mathbf{K}_c$ is the intrinsic matrix of camera c, parameterised by the focal length, f, the principal point, $[p_x,p_y]^T$, and the axis skew, γ, given by:

$$\mathbf{K} = \begin{bmatrix} f & \gamma & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

and since the current pose estimate consists of only translation along the X and Y axes and rotation about Z axes,

$$\mathbf{T}(\theta,\mathbf{t}) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & t_x \\ \sin\theta & \cos\theta & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

Therefore, expanding Eq. 3.4 gives:

$$\mathbf{p}_i = \begin{bmatrix} x \\ y \\ b \end{bmatrix} = \begin{bmatrix} f & \gamma & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r11 & r12 & r13 & l_x \\ r21 & r22 & r23 & l_y \\ r31 & r32 & r33 & l_z \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 & t_x \\ \sin\theta & \cos\theta & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} {}^{World}\mathbf{P}_i \tag{3.7}$$

We can then find the individual components of $\mathbf{p}_i$ with:

$$\begin{aligned} x &= \begin{bmatrix} a\cos\theta + b\sin\theta & -a\sin\theta + b\cos\theta & c & at_x + bt_y + d \end{bmatrix} {}^{w}\mathbf{P} \\ y &= \begin{bmatrix} e\cos\theta + f\sin\theta & -e\sin\theta + f\cos\theta & g & et_x + ft_y + h \end{bmatrix} {}^{w}\mathbf{P} \\ b &= \begin{bmatrix} i\cos\theta & -i\sin\theta + j\cos\theta & k & it_x + jt_y + l \end{bmatrix} {}^{w}\mathbf{P} \end{aligned} \tag{3.8}$$

where,

$$\begin{aligned} a &= fr_{11} + p_x r_{31}, & b &= fr_{12} + p_x r_{32}, & c &= fr_{13} + p_x r_{33}, & d &= fl_x + p_x l_z, \\ e &= fr_{21} + p_y r_{31}, & f &= fr_{22} + p_y r_{32}, & g &= fr_{23} + p_y r_{33}, & h &= fl_y + p_y l_z, \\ i &= r_{31}, & j &= r_{32}, & k &= r_{33}, & l &= l_z \end{aligned} \tag{3.9}$$

The point $\mathbf{p}_i$ is then converted into pixel coordinates by normalising by the b term and removing the last element which becomes 1 after normalisation:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x/b \\ y/b \\ b/b \end{bmatrix} \tag{3.10}$$

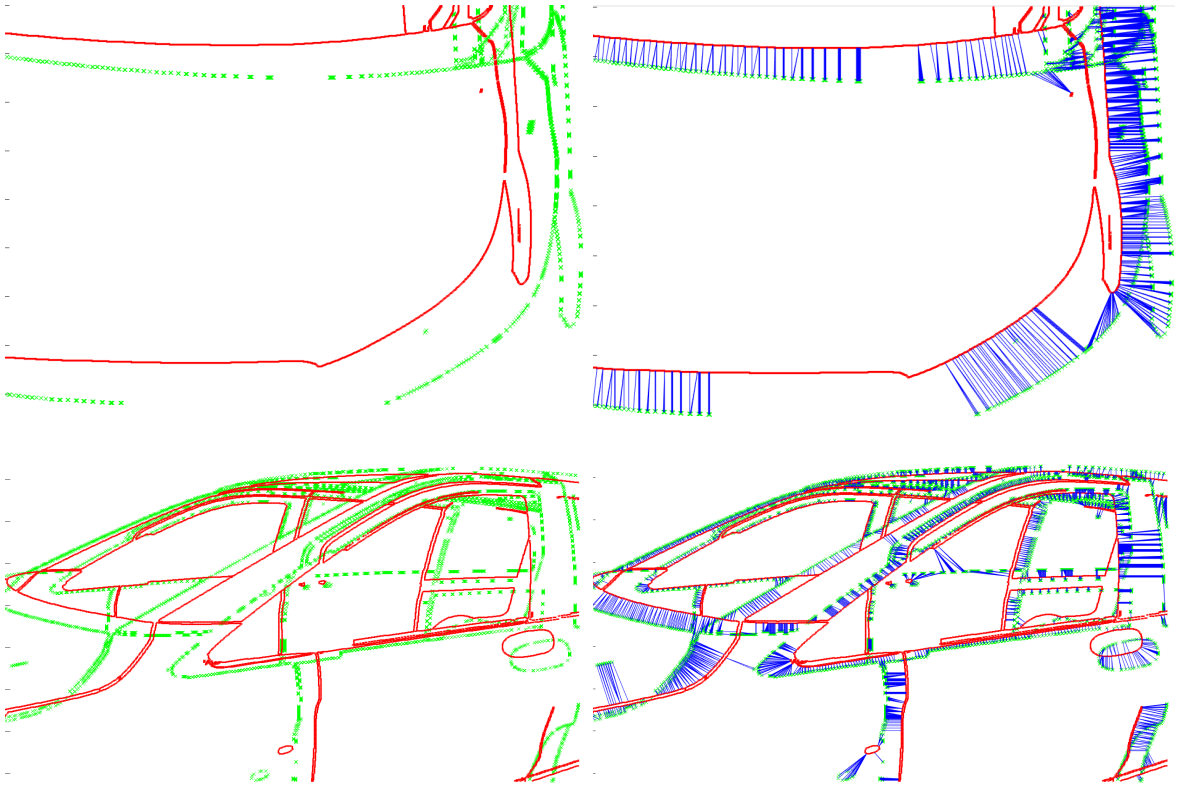


Figure 3.7: Real camera edge points shown in red, projection of simulated camera edge points shown in green and correspondence between nearest neighbours shown in blue.

3.6 Nearest Neighbour Point Correspondences

For each point in the projection of a simulated camera image, $\mathbf{p}_i$, we find the two nearest neighbours in the real edge image. The two points in the real edge image corresponding to $\mathbf{p}_i$ are given by $\mathbf{q}_{k_1^i}$ and $\mathbf{q}_{k_2^i}$. The nearest neighbours are found by representing the sets of points in a Kd-Tree [33] and finding the closest point using a euclidean distance metric.

3.7 Optimisation

The optimisation tries to find the optimal pose offset of the simulated CAD model to minimise the distance between the projected simulated camera edges and the real camera edges. We use two types of M-estimators instead of least squares in our cost function to make the optimisation robust to outliers in the data. The outliers are caused by points in the simulated camera images which don't have corresponding points in the real camera images. The use of M-estimators is discussed further in Sec. 5.1. Each component of the optimisation procedure is detailed in following sections.

3.7.1 M-Estimation

Consider a set of n data points, $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, and a model function, $\mathbf{y} = f(\mathbf{x}, \boldsymbol{\beta})$, where $\mathbf{y}$ is a function of variable $\mathbf{x}$ and m parameters, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_m)$.

We want to find the vector of parameters, $\boldsymbol{\beta}$, that minimises the function:

$$\mathbf{S} = \sum_{i=1}^{n} \rho(e_i) \tag{3.11}$$

where the residuals are the difference between predicted and actual values given by $e_i = y_i - f(x_i, \boldsymbol{\beta})$ for $i = 1, \ldots, n$.

We choose two functions for $\rho(x)$: Huber's function [34], ρ_H, and Tukey's bisquare function, ρ_B. The two functions are shown in Table 3.2.

The minimum value of S occurs when the gradient of the cost function is zero, given by:

$$\begin{aligned} \frac{\partial S}{\partial \beta_j} &= \sum_{i=1}^{n} \frac{\partial \rho(e_i)}{\partial \beta_j}, \qquad \text{for } j = 1, \ldots, m \\ &= \sum_{i=1}^{n} \psi(e_i) \frac{\partial e_i}{\partial \beta_j} = 0 \end{aligned} \tag{3.12}$$

where $\psi(e_i) = d\rho(e_i)/de_i$ is called the *influence function*. We can then choose a *weight function*, $w(e_i)$, such that:

$$w(e_i) = \frac{\psi(e_i)}{e_i} \tag{3.13}$$

When we substitute $\psi(e_i) = w(e_i)e_i$ into Eq. 3.12, it becomes:

$$\frac{\partial S}{\partial \beta_j} = \sum_{i=1}^{n} w(e_i)e_i \frac{\partial e_i}{\partial \beta_j} \qquad \text{for } j = 1, \ldots, m \tag{3.14}$$

This is the same system of equations we would obtain if we were solving an Iteratively Reweighted Least-Squares (IRLS) problem of the following form:

$$\min \sum_{i=1}^{n} w(e_i)e_i^2 \tag{3.15}$$

Table 3.2: Table showing the Huber and Tukey (bisquare) M-estimators

Type	$\rho(x)$	$\psi(x)$	$w(x)$
Huber $\begin{cases} if \mid x \mid \leq k \\ if \mid x \mid > k \end{cases}$	$\begin{cases} x^2/2 \\ k(\mid x \mid -k/2) \end{cases}$	$\begin{cases} x \\ ksgn(x) \end{cases}$	$\begin{cases} 1 \\ k/ \mid x \mid \end{cases}$
Tukey $\begin{cases} if \mid x \mid \leq k \\ if \mid x \mid > k \end{cases}$	$\begin{cases} \frac{k^2}{6}\left(1 - \left[1 - (x/k)^2\right]^3\right) \\ k^2/ \mid x \mid \end{cases}$	$\begin{cases} x\left[1 - (x/k)^2\right]^2 \\ 0 \end{cases}$	$\begin{cases} \left[1 - (x/k)^2\right]^2 \\ 0 \end{cases}$

The value k in both the Huber and bisquare estimators is called the *tuning constant*.

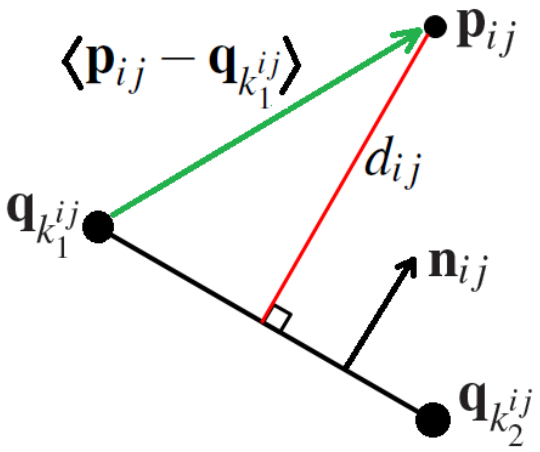


Figure 3.8: Image showing perpendicular distance between a point, $\mathbf{p}_{ij}$, and the line joining two points, $\mathbf{q}_{k_1^{ij}}$ and $\mathbf{q}_{k_2^{ij}}$.

3.7.2 Cost Function

We formulate a cost function to minimise the perpendicular distance between each simulated camera point, $\mathbf{p}_{ij}$, and the line segment joining the two corresponding real camera points, $\mathbf{q}_{k_1^{ij}}$ and $\mathbf{q}_{k_2^{ij}}$, for the ith point of the jth camera.

The perpendicular distance between a point, $\mathbf{x}$, and a line, $\mathbf{l}$, is given by:

$$d = \mid \mathbf{n} \cdot \mathbf{v} \mid \tag{3.16}$$

where $\mathbf{n}$ is the unit normal vector to the line $\mathbf{l}$ and $\mathbf{v}$ is a vector from a point on the line $\mathbf{l}$ to the point $\mathbf{x}$.

Therefore, the perpendicular distance between the point, $\mathbf{p}_{ij}$, and the line segment $\mathbf{q}_{k_1^{ij}} \rightarrow \mathbf{q}_{k_2^{ij}}$, is given by (See Fig. 3.8):

$$\begin{aligned} d_{ij} &= \mathbf{n}_{ij}^T (\mathbf{p}_{ij} - \mathbf{q}_{k_1^{ij}}) \\ &= \mathbf{n}_{ij}^T \left(\begin{bmatrix} \frac{x_{ij}(\theta, \mathbf{t}, \mathbf{P}_{ij})}{b_{ij}(\theta, \mathbf{t}, \mathbf{P}_{ij})} \\ \frac{y_{ij}(\theta, \mathbf{t}, \mathbf{P}_{ij})}{b_{ij}(\theta, \mathbf{t}, \mathbf{P}_{ij})} \end{bmatrix} - \mathbf{q}_{k_1^{ij}} \right) \end{aligned} \tag{3.17}$$

Therefore, the cost function is given by:

$$S = \min_{\theta, \mathbf{t}} \sum_{j}^{c} \rho \left(\sum_{i}^{m_j} \mathbf{n}_{ij}^T \left(\begin{bmatrix} \frac{x_{ij}(\theta, \mathbf{t}, \mathbf{P}_{ij})}{b_{ij}(\theta, \mathbf{t}, \mathbf{P}_{ij})} \\ \frac{y_{ij}(\theta, \mathbf{t}, \mathbf{P}_{ij})}{b_{ij}(\theta, \mathbf{t}, \mathbf{P}_{ij})} \end{bmatrix} - \mathbf{q}_{k_1^{ij}} \right) \right) \tag{3.18}$$

where c is the number of cameras, m_j is the number of points observed by the jth camera, $\mathbf{n}_{ij}$ is the unit normal to the line segment $\mathbf{q}_{k_1^{ij}} \rightarrow \mathbf{q}_{k_2^{ij}}$ and $\rho()$ is an M-estimator function. Currently, we use the Huber function for a set number of iterations and then switch to the bisquare function. x_{ij}, y_{ij} and b_{ij} are results from the projection of $\mathbf{P}_{ij}$, found in Eq. 3.8.

3.7.3 Jacobian

We can calculate the Jacobian, $\mathbf{J}(\theta,\mathbf{t})$ as:

$$J(\theta,\mathbf{t}) = \begin{bmatrix} \dfrac{\partial f_{1,1}(\theta,\mathbf{t})}{\partial\theta} & \dfrac{\partial f_{1,1}(\theta,\mathbf{t})}{\partial\mathbf{t}} \\ \dfrac{\partial f_{1,2}(\theta,\mathbf{t})}{\partial\theta} & \dfrac{\partial f_{1,2}(\theta,\mathbf{t})}{\partial\mathbf{t}} \\ \vdots & \\ \dfrac{\partial f_{i,j}(\theta,\mathbf{t})}{\partial\theta} & \dfrac{\partial f_{i,j}(\theta,\mathbf{t})}{\partial\mathbf{t}} \end{bmatrix} \tag{3.19}$$

where

$$f_{i,j}(\theta,\mathbf{t}) = \mathbf{n}_{ij}^T \left(\begin{bmatrix} \dfrac{x_{ij}(\theta,\mathbf{t},\mathbf{P}_{ij})}{b_{ij}(\theta,\mathbf{t},\mathbf{P}_{ij})} \\ \dfrac{y_{ij}(\theta,\mathbf{t},\mathbf{P}_{ij})}{b_{ij}(\theta,\mathbf{t},\mathbf{P}_{ij})} \end{bmatrix} - \mathbf{q}_{k_1^{ij}} \right) \tag{3.20}$$

and

$$\frac{\partial f_{i,j}(\theta,\mathbf{t})}{\partial\theta} = \mathbf{n}_i^T \begin{bmatrix} \dfrac{b\dfrac{\partial x}{\partial\theta} - x\dfrac{\partial b}{\partial\theta}}{b^2} \\ \dfrac{b\dfrac{\partial y}{\partial\theta} - y\dfrac{\partial b}{\partial\theta}}{b^2} \end{bmatrix} \qquad \frac{\partial f_{i,j}(\theta,\mathbf{t})}{\partial\mathbf{t}} = \mathbf{n}_i^T \begin{bmatrix} \dfrac{b\dfrac{\partial x}{\partial\mathbf{t}} - x\dfrac{\partial b}{\partial\mathbf{t}}}{b^2} \\ \dfrac{b\dfrac{\partial y}{\partial\mathbf{t}} - y\dfrac{\partial b}{\partial\mathbf{t}}}{b^2} \end{bmatrix} \tag{3.21}$$

$$\begin{aligned} \frac{\partial x}{\partial\theta} &= (-a\sin\theta + b\cos\theta)X + (-a\cos\theta - b\sin\theta)Y \\ \frac{\partial x}{\partial\mathbf{t}} &= \begin{bmatrix} a & b \end{bmatrix} \\ \frac{\partial y}{\partial\theta} &= (-e\sin\theta + f\cos\theta)X + (-e\cos\theta - f\sin\theta)Y \\ \frac{\partial y}{\partial\mathbf{t}} &= \begin{bmatrix} e & f \end{bmatrix} \\ \frac{\partial b}{\partial\theta} &= (-i\sin\theta + j\cos\theta)X + (-i\cos\theta - j\sin\theta)Y \\ \frac{\partial b}{\partial\mathbf{t}} &= \begin{bmatrix} i & j \end{bmatrix} \end{aligned} \tag{3.22}$$

where $\mathbf{P}_{ij} = [X\ Y\ Z]^T$ and a, b, c, d, e, f, g, h, i and j are taken from 3.8 .

3.7.4 Minimisation

Since our cost function is non-linear, the gradient equations in Eq. 3.12 will contain both the parameters and independent variables. Therefore, we cannot find a closed form solution for the gradient equations. Instead, we refine the parameters iteratively using the Gauss-Newton algorithm, according to:

$$\Delta\boldsymbol{\beta}^t = (\mathbf{J}^T\mathbf{W}^{t-1}\mathbf{J})^{-1}(\mathbf{J}^T\mathbf{W}^{t-1})(\mathbf{We}) \tag{3.23}$$

$$\boldsymbol{\beta}^t = \boldsymbol{\beta}^{t-1} + \Delta\boldsymbol{\beta}^t \tag{3.24}$$

where $W = diag(w_1, \ldots, w_m)$ found according to Eq. 3.13, $\mathbf{e}$ is the vector of residuals, $[e_1, \ldots, e_m]^T$, $\boldsymbol{\beta}$ is the vector of parameters to be optimised, $[\beta_1, \ldots, \beta_m]^T$, $\mathbf{J}$ is the Jacobian and the superscript $t-1$ indicates that the value should be calculated during the previous iteration.

Chapter 4

Results

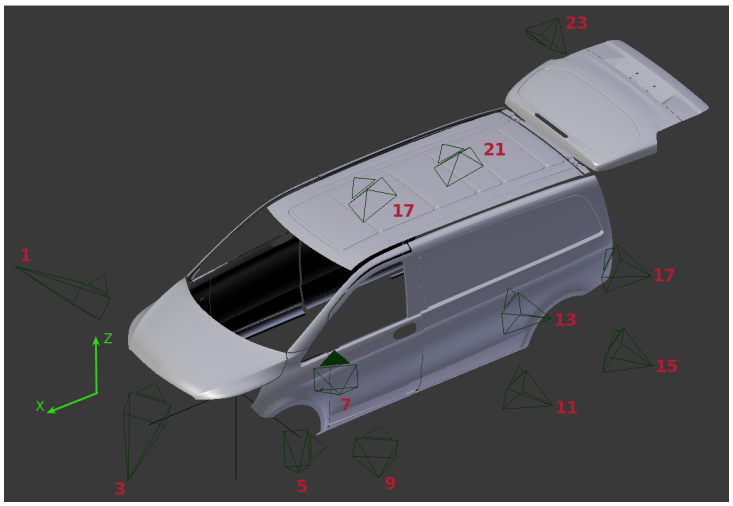


Figure 4.1: Numbering used for camera array. All odd numbered cameras, c_i for $i = 1, \ldots, n/2$, are shown here. Each camera c_i (except for camera 23) has a corresponding camera, $c_i + 1$, that is mirrored about the x-z plane. E.g. camera 2 is the mirror of camera 1 about the x-z plane.

4.1 Experiment Methodology

In this chapter, we cover three experiments that we performed to test our system:

1. Test robustness of individual cameras.
2. Test the best subset of cameras.
3. Compare usage of M-estimators.

The parameters used for all experiments unless otherwise stated are shown in Table. 4.1. The tuning constant of the M-estimator being used and the Median Absolute Residual (MAR) is updated every 10 iterations.

The values for the Huber and bisquare tuning constants are chosen as $k = 1.345\sigma$ and $k = 4.685\sigma$, respectively, where σ is the standard deviation of the residuals [35]. This results in 95% asymptotic efficiency when the residuals are normally distributed (i.e. when there are no outliers). Since standard deviation is not a robust measure of statistical spread, we instead use the MAR to get, $\hat{\sigma} = MAR/0.6745$ [36].

Table 4.1: Table showing the parameters used during the experiments.

Parameter	**Symbol**	**Value**
Robust Std. Dev. Estimate	$\hat{\sigma}$	MAR / 0.6745
Huber tuning constant	k_h	$1.345\hat{\sigma}$
Bisquare tuning constant	k_b	$4.6851\hat{\sigma}$
Iterations using Huber estimator	-	20
Iterations using Bisquare estimator	-	20

4.1.1 Experiment 1: Individual Cameras

The first experiment we ran was to compare the performance of individual cameras with varying pose offsets of the vehicle (the camera numbering is shown in Fig. 4.1). The optimisation was run separately for each camera with the vehicle offset in x, y or θ. We only offset the vehicle along a single axis so that we can observe which cameras are more robust to specific movements of the vehicle.

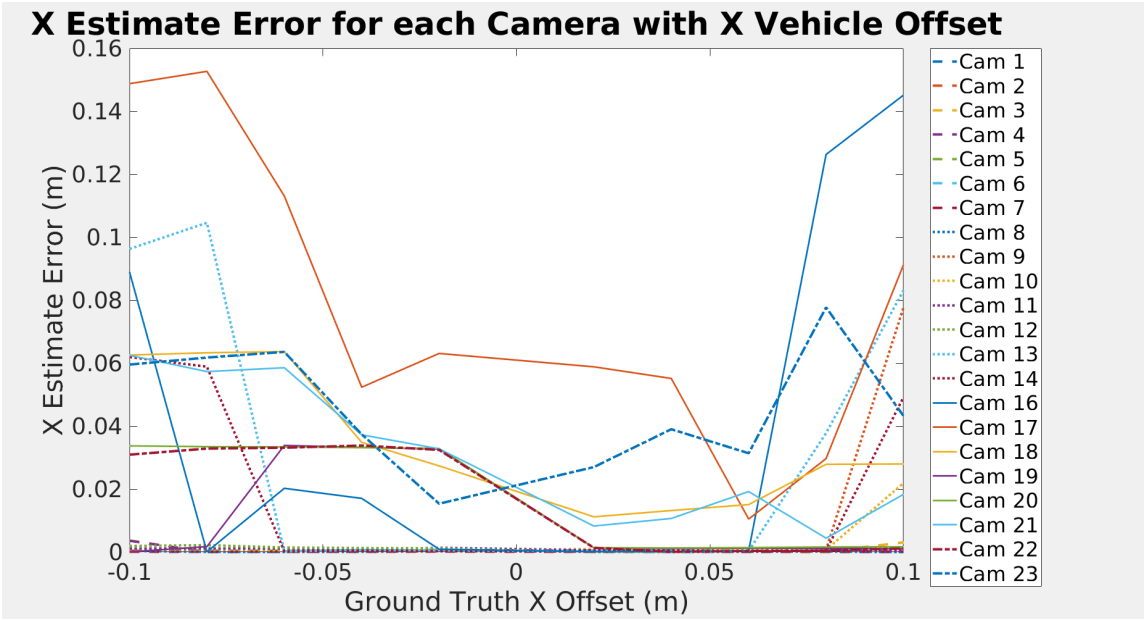
The results are shown on the following pages where each figure shows the x, y or θ estimation error for each of the 23 cameras except for camera 15. The optimisation for camera 15 diverged so it is not included in the graphs.

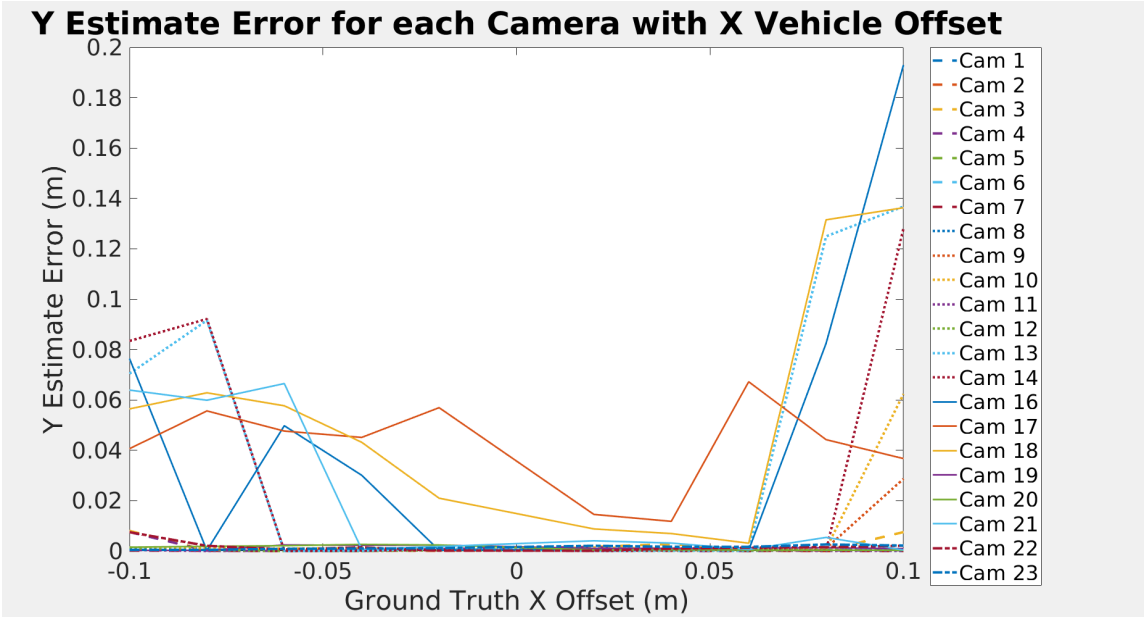
Based off this experiment, the cameras were manually placed into three different groups:

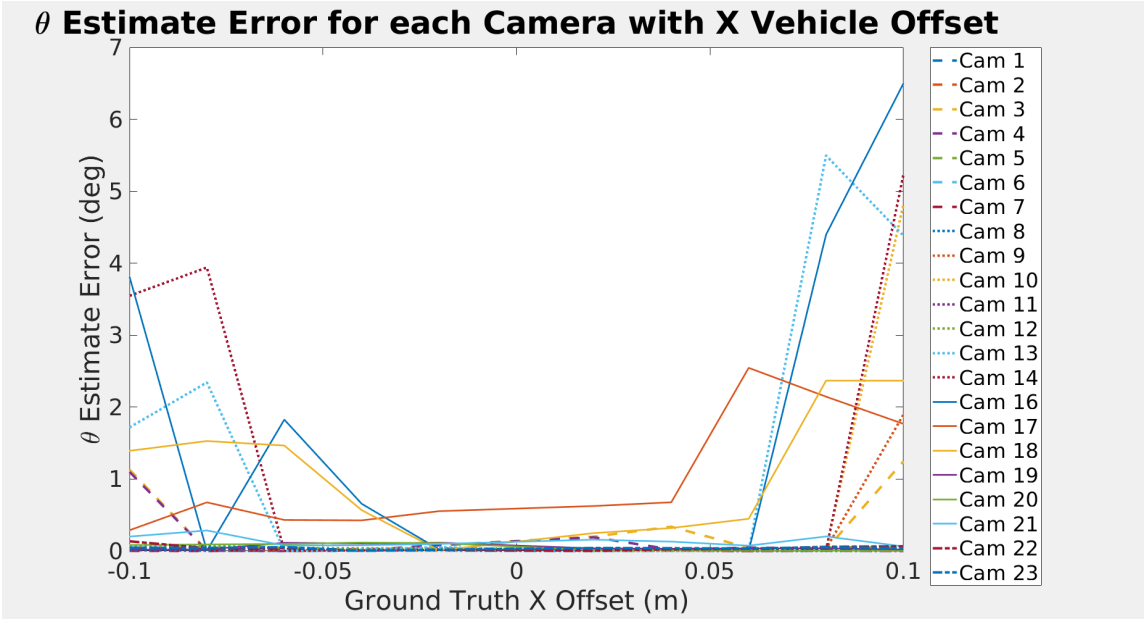
- **Robust Cameras**: These are the cameras that were robust across the entire test range. These cameras will be the primary candidates for use in the final system.
- **Semi-Robust Cameras**: These are the cameras that were robust across a large portion of the test range, particularly over the required tolerance of the system of ±2cm. Since there were numerous cameras in the first group, it's likely that we won't need to use any cameras from this group. However, the robust group consists of cameras that are all towards the front of the vehicle. We may benefit from having a more even spread of cameras around the vehicle depending on the conditions in the actual manufacturing plant. Therefore, these cameras can be considered if needed.
- **Non-Robust Cameras**: These are the cameras that performed poorly across all portions of the test range. Importantly, these cameras were not robust in the region of required tolerance, therefore they are not useful to us for the vehicle model we tested with.

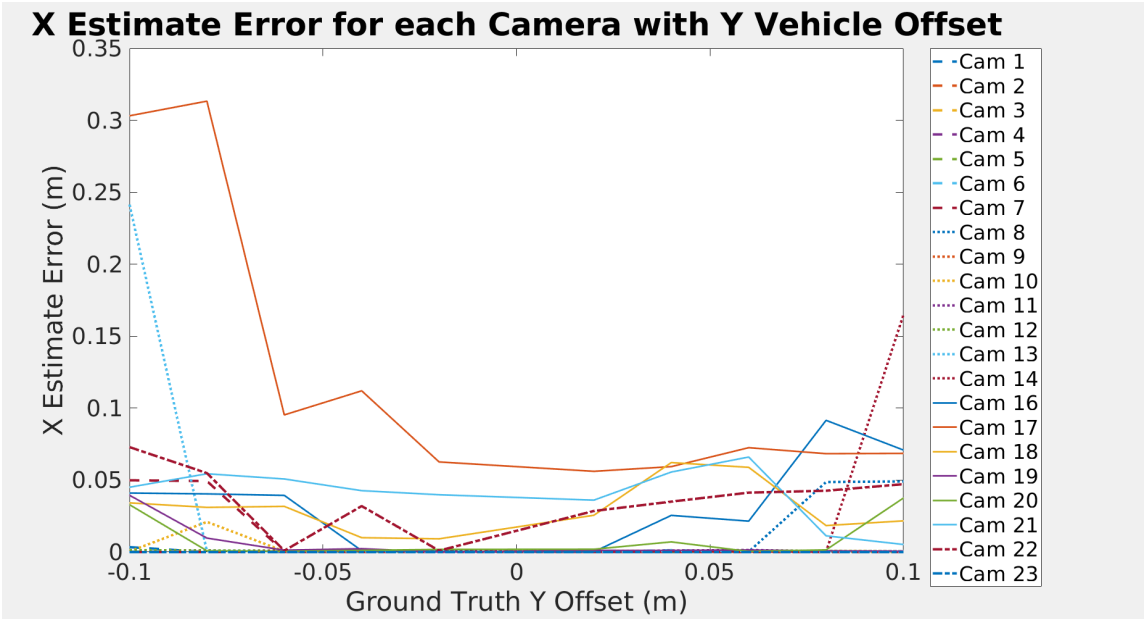
The cameras belonging to each group are as follows:

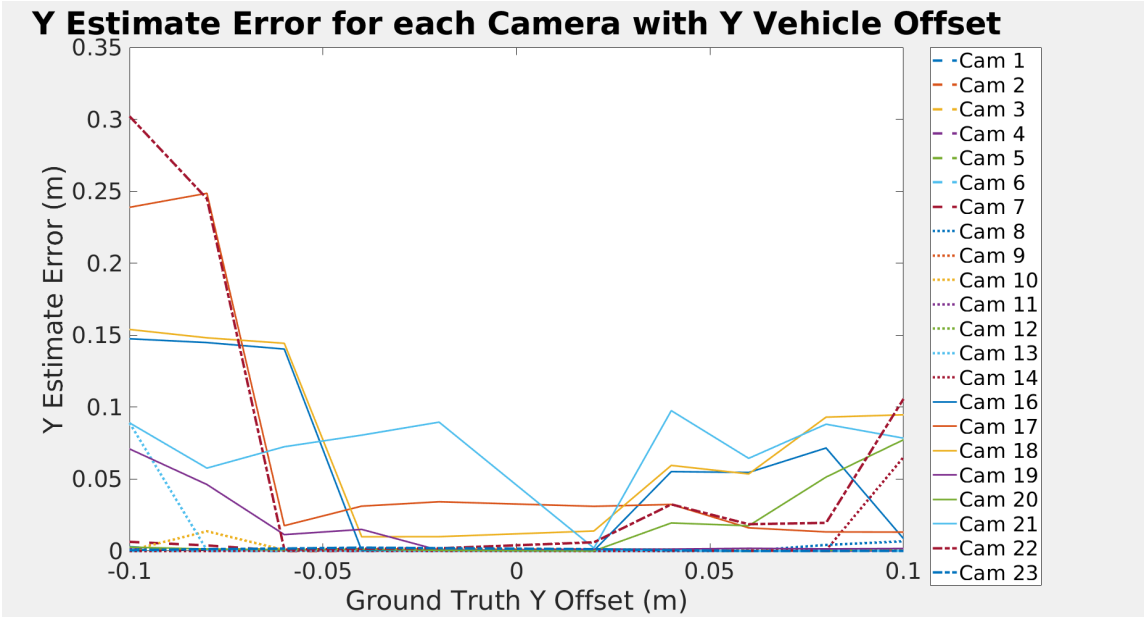
- **Robust Cameras**: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
- **Semi-Robust Cameras**: {13, 14, 19, 20}
- **Non-Robust Cameras**: {15, 16, 17, 18, 21, 22, 23}

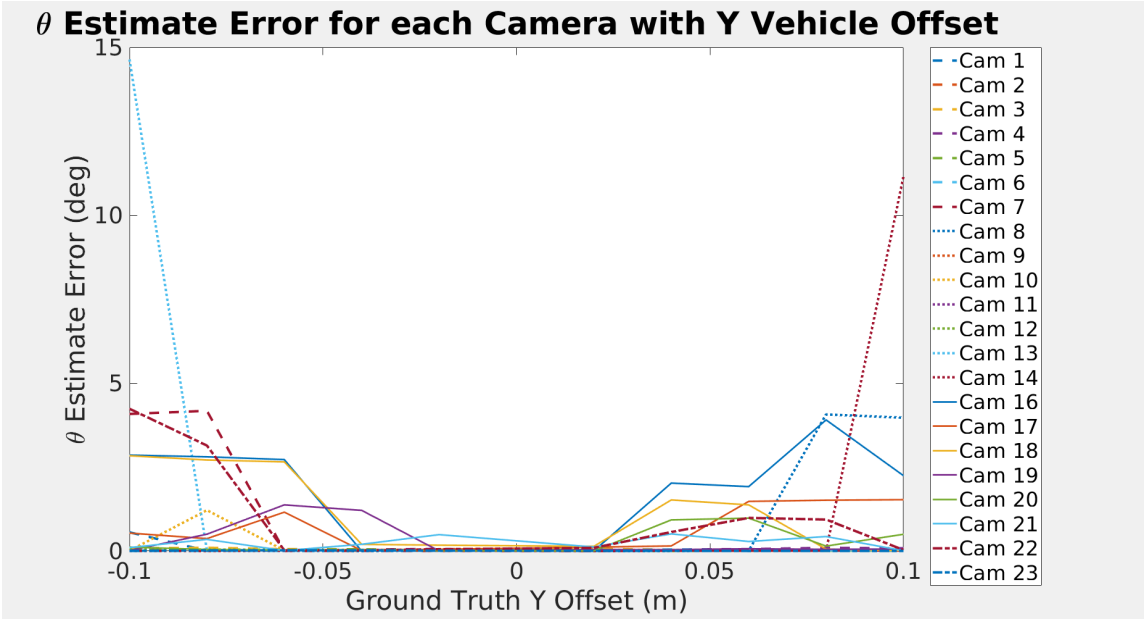

X Estimate Error for each Camera with X Vehicle Offset
X Estimate Error (m)
Ground Truth X Offset (m)
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.1
-0.05
0
0.05
0.1
Cam 1
Cam 2
Cam 3
Cam 4
Cam 5
Cam 6
Cam 7
Cam 8
Cam 9
Cam 10
Cam 11
Cam 12
Cam 13
Cam 14
Cam 16
Cam 17
Cam 18
Cam 19
Cam 20
Cam 21
Cam 22
Cam 23

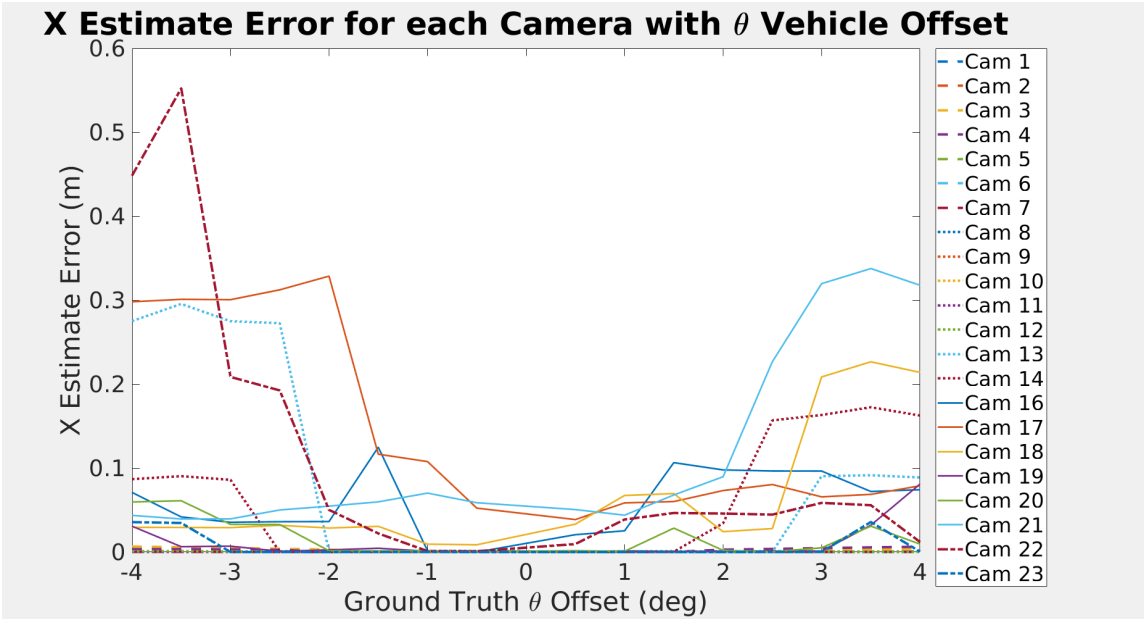

Y Estimate Error for each Camera with X Vehicle Offset
Y Estimate Error (m)
Ground Truth X Offset (m)
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.1
-0.05
0
0.05
0.1
Cam 1
Cam 2
Cam 3
Cam 4
Cam 5
Cam 6
Cam 7
Cam 8
Cam 9
Cam 10
Cam 11
Cam 12
Cam 13
Cam 14
Cam 16
Cam 17
Cam 18
Cam 19
Cam 20
Cam 21
Cam 22
Cam 23

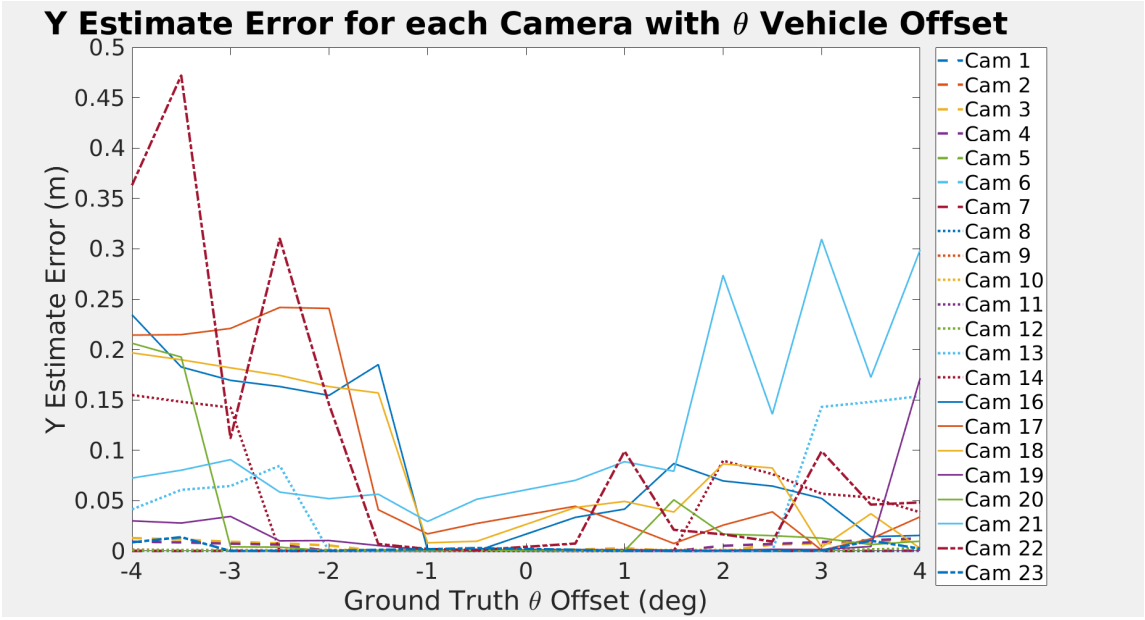

θ Estimate Error for each Camera with X Vehicle Offset
θ Estimate Error (deg)
Ground Truth X Offset (m)
7
6
5
4
3
2
1
0
-0.1
-0.05
0
0.05
0.1
Cam 1
Cam 2
Cam 3
Cam 4
Cam 5
Cam 6
Cam 7
Cam 8
Cam 9
Cam 10
Cam 11
Cam 12
Cam 13
Cam 14
Cam 16
Cam 17
Cam 18
Cam 19
Cam 20
Cam 21
Cam 22
Cam 23

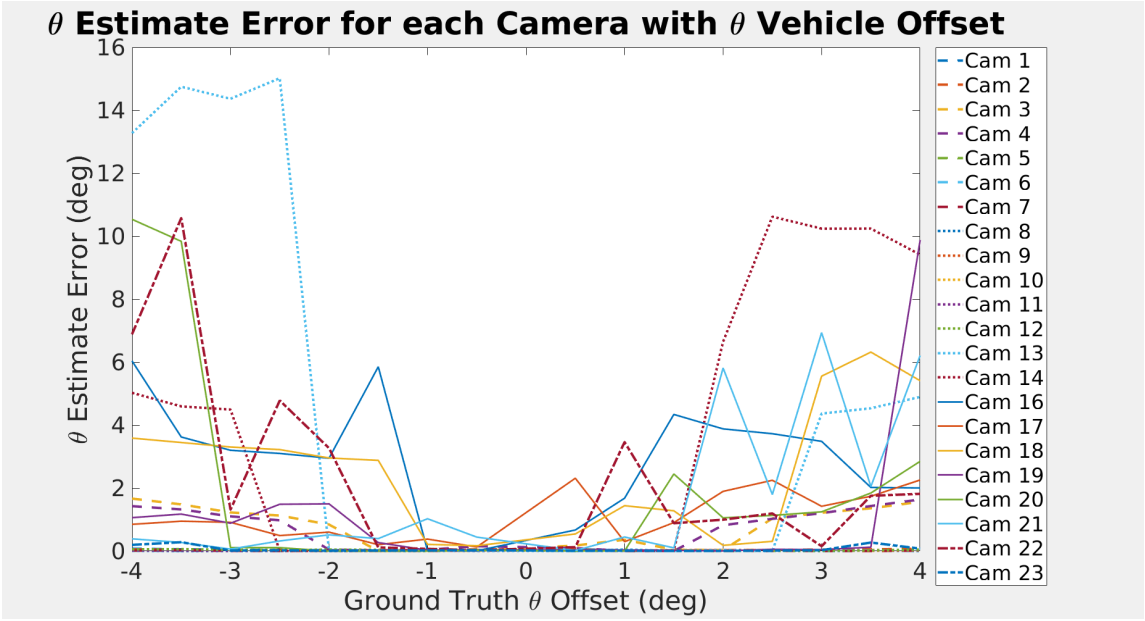

X Estimate Error for each Camera with Y Vehicle Offset
X Estimate Error (m)
Ground Truth Y Offset (m)
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.1
-0.05
0
0.05
0.1
Cam 1
Cam 2
Cam 3
Cam 4
Cam 5
Cam 6
Cam 7
Cam 8
Cam 9
Cam 10
Cam 11
Cam 12
Cam 13
Cam 14
Cam 16
Cam 17
Cam 18
Cam 19
Cam 20
Cam 21
Cam 22
Cam 23


Y Estimate Error for each Camera with Y Vehicle Offset
Y Estimate Error (m)
Ground Truth Y Offset (m)
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.1
-0.05
0
0.05
0.1
Cam 1
Cam 2
Cam 3
Cam 4
Cam 5
Cam 6
Cam 7
Cam 8
Cam 9
Cam 10
Cam 11
Cam 12
Cam 13
Cam 14
Cam 16
Cam 17
Cam 18
Cam 19
Cam 20
Cam 21
Cam 22
Cam 23


θ Estimate Error for each Camera with Y Vehicle Offset
θ Estimate Error (deg)
Ground Truth Y Offset (m)
15
10
5
0
-0.1
-0.05
0
0.05
0.1
Cam 1
Cam 2
Cam 3
Cam 4
Cam 5
Cam 6
Cam 7
Cam 8
Cam 9
Cam 10
Cam 11
Cam 12
Cam 13
Cam 14
Cam 16
Cam 17
Cam 18
Cam 19
Cam 20
Cam 21
Cam 22
Cam 23


X Estimate Error for each Camera with θ Vehicle Offset
X Estimate Error (m)
Ground Truth θ Offset (deg)
Cam 1
Cam 2
Cam 3
Cam 4
Cam 5
Cam 6
Cam 7
Cam 8
Cam 9
Cam 10
Cam 11
Cam 12
Cam 13
Cam 14
Cam 16
Cam 17
Cam 18
Cam 19
Cam 20
Cam 21
Cam 22
Cam 23


Y Estimate for each Camera with θ Vehicle Offset
Y Estimate Error (m)
Ground Truth θ Offset (deg)
Cam 1
Cam 2
Cam 3
Cam 4
Cam 5
Cam 6
Cam 7
Cam 8
Cam 9
Cam 10
Cam 11
Cam 12
Cam 13
Cam 14
Cam 16
Cam 17
Cam 18
Cam 19
Cam 20
Cam 21
Cam 22
Cam 23


θ Estimate Error for each Camera with θ Vehicle Offset
θ Estimate Error (deg)
Ground Truth θ Offset (deg)
Cam 1
Cam 2
Cam 3
Cam 4
Cam 5
Cam 6
Cam 7
Cam 8
Cam 9
Cam 10
Cam 11
Cam 12
Cam 13
Cam 14
Cam 16
Cam 17
Cam 18
Cam 19
Cam 20
Cam 21
Cam 22
Cam 23

4.1.2 Experiment 2: Best Subset of Cameras

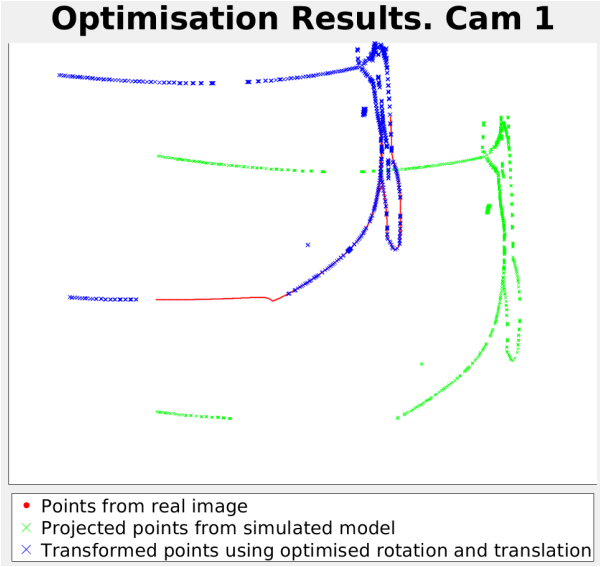
This experiment tests the optimisation using the set of robust cameras identified in Experiment 1. The final estimation results are shown in Table. 4.2 and the resulting transformations are shown in the following pages.

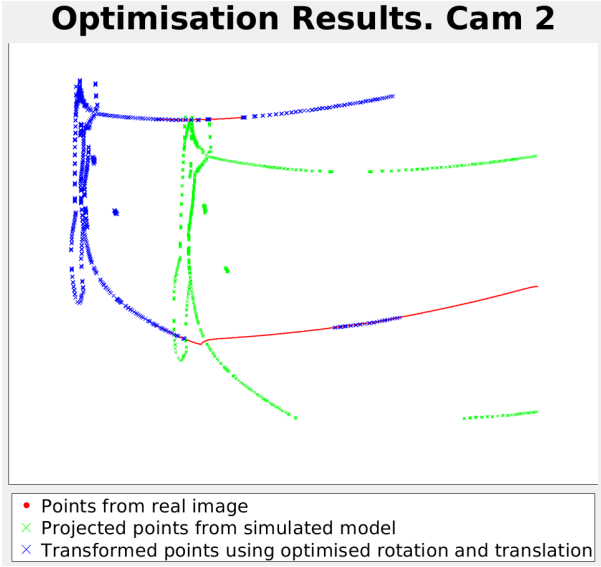
The purpose of this experiment is to show the robustness of the system. In reality, the required tolerance of the system is approximately ±2cm. Therefore, the pose offsets in this experiment are an extreme case that we will not see in the real scenario. As a result, an even smaller subset of cameras could be chosen to reduce the optimisation time. This subset could be chosen by examining the most robust cameras within the tolerance range. Making this selection is redundant at this stage of the project because the optimisation currently works well enough with the simulated data that we could choose a single camera that could handle all offsets in the tolerance range. Therefore, this selection should be made based off tests with real data, as these images are more likely to have noise and outliers that will result in the optimisation benefiting from additional cameras.

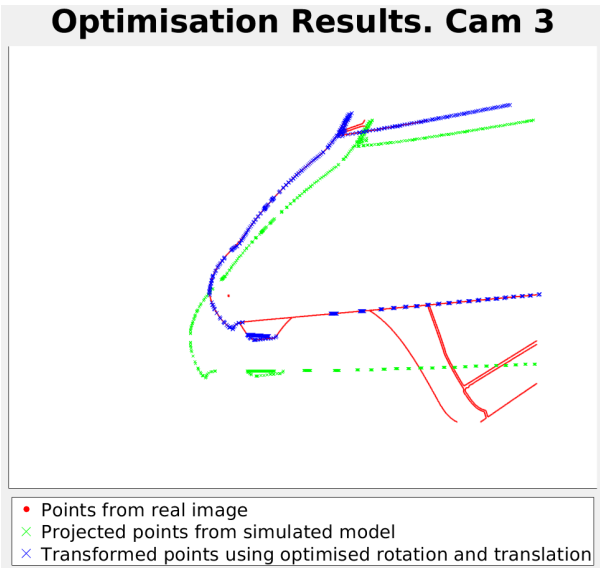
It should also be noted that the best subset of cameras will depend on the model of the vehicle. For example, cameras 15 and 17 at the moment are part of the non-robust group because they are viewing the flat face of the vehicle's side and therefore do not detect many significant edges. With a different model of vehicle, their viewpoint may be different and therefore may detect more useful edges.

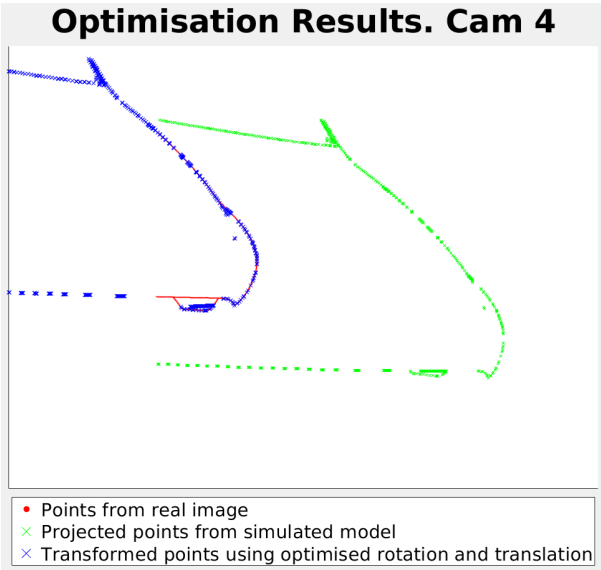
Table 4.2: Table showing results of Experiment 2.

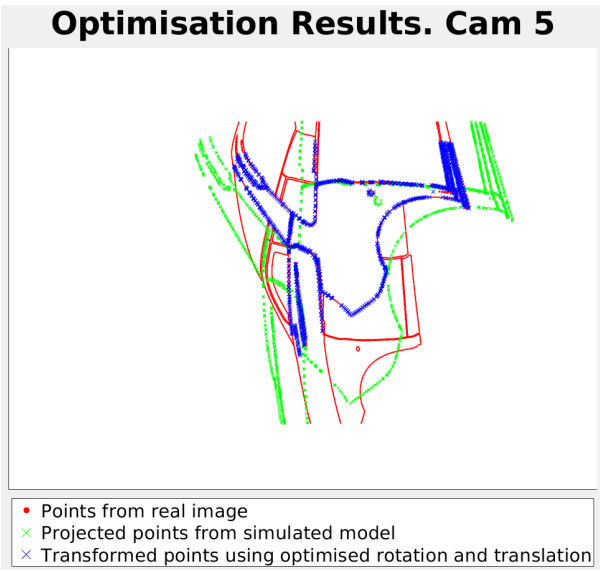
$\theta_{gt}\ (deg)$	$x_{gt}\ (m)$	$y_{gt}\ (m)$	θ *error* (deg)	*x error* (m)	*y error* (m)
2	0.1	0.1	0.000258	0.000012	0.0
-2	-0.1	-0.1	0.000099	0.000052	0.000006
3	0.2	0.2	0.000111	0.00001	0.000003
-3	-0.2	-0.2	0.001417	0.000097	0.000001
4	0.3	0.3	0.000077	0.000004	0.000023
-4	-0.3	-0.3	0.306384	0.003539	0.005124
5	0.4	0.4	0.109704	0.003142	0.000667
-5	-0.4	-0.4	0.511136	0.03954	0.055021
5	0.5	0.5	13.551265	0.346056	0.231594

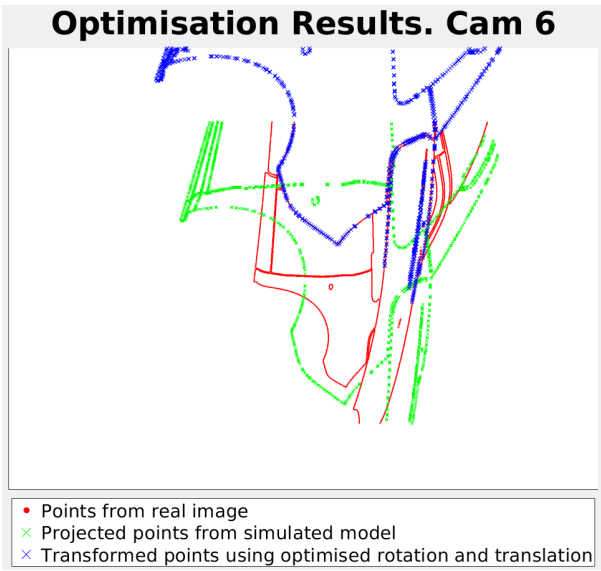

Optimisation Results. Cam 1
Points from real image
Projected points from simulated model
Transformed points using optimised rotation and translation


Optimisation Results. Cam 2
Points from real image
Projected points from simulated model
Transformed points using optimised rotation and translation

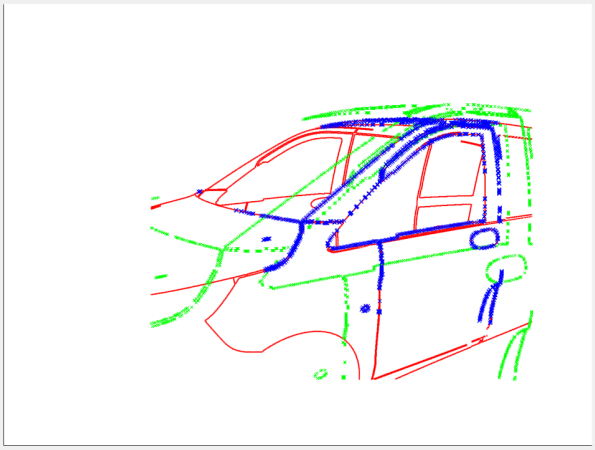

Optimisation Results. Cam 3
Points from real image
Projected points from simulated model
Transformed points using optimised rotation and translation


Optimisation Results. Cam 4
Points from real image
Projected points from simulated model
Transformed points using optimised rotation and translation


Optimisation Results. Cam 5
Points from real image
Projected points from simulated model
Transformed points using optimised rotation and translation


Optimisation Results. Cam 6
Points from real image
Projected points from simulated model
Transformed points using optimised rotation and translation

Optimisation Results. Cam 7

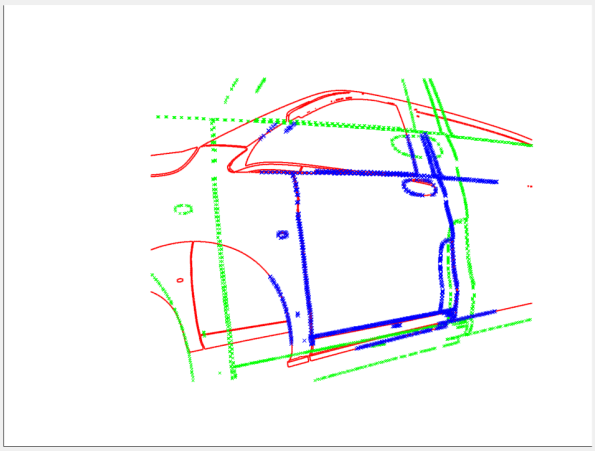


- Points from real image
- Projected points from simulated model
- Transformed points using optimised rotation and translation

Optimisation Results. Cam 8

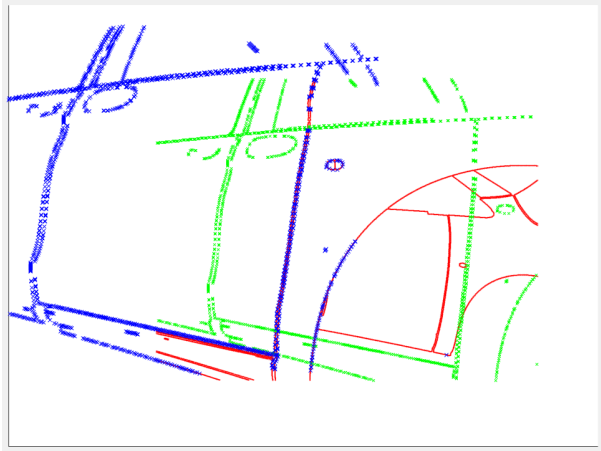
- Points from real image
- Projected points from simulated model
- Transformed points using optimised rotation and translation

Optimisation Results. Cam 9



- Points from real image
- Projected points from simulated model
- Transformed points using optimised rotation and translation

Optimisation Results. Cam 10

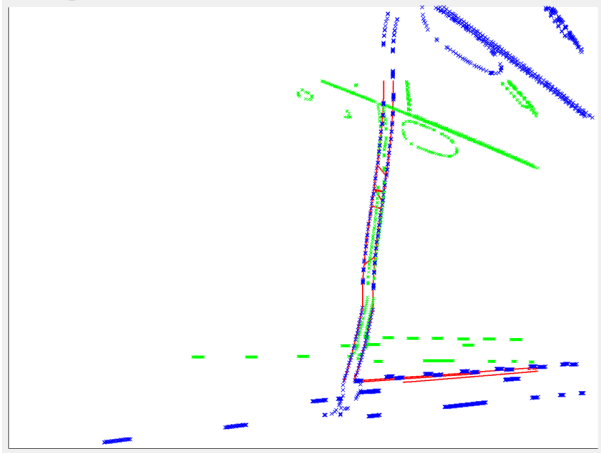


- Points from real image
- Projected points from simulated model
- Transformed points using optimised rotation and translation

Optimisation Results. Cam 11

- Points from real image
- Projected points from simulated model
- Transformed points using optimised rotation and translation

Optimisation Results. Cam 12



- Points from real image
- Projected points from simulated model
- Transformed points using optimised rotation and translation

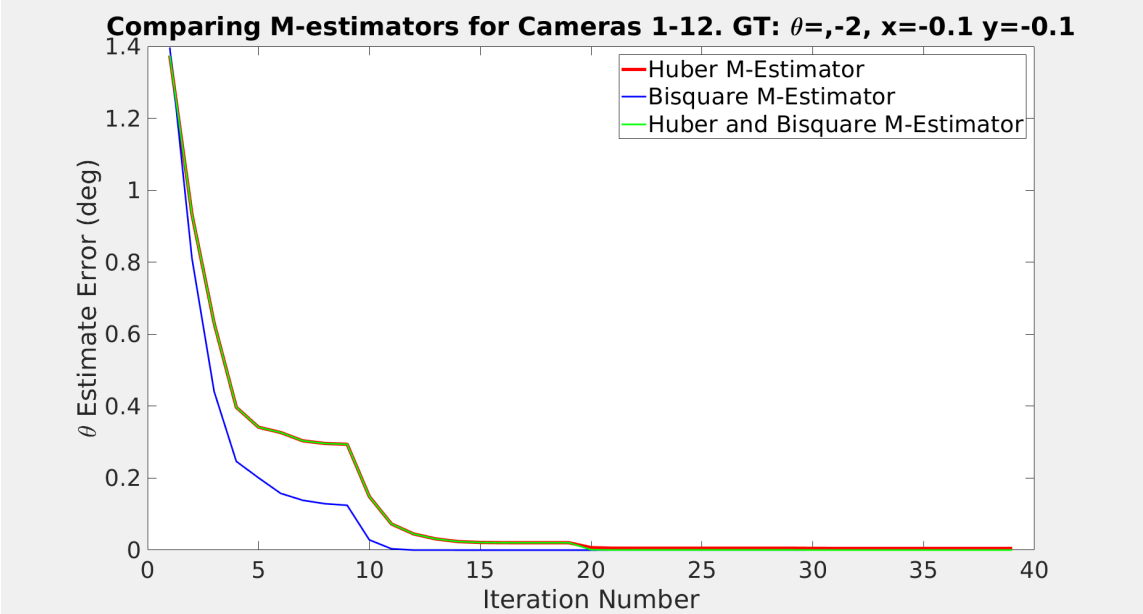
4.1.3 Experiment 3: Comparing Huber and Bisquare M-estimators

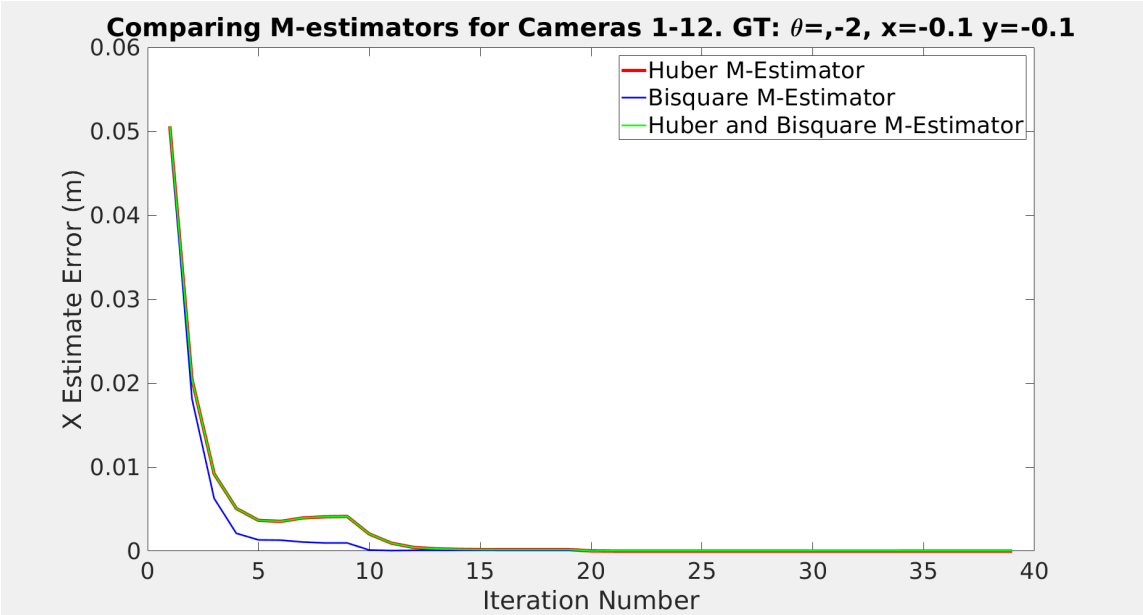
This experiment is to compare the estimation results when using Huber, Bisquare and a combination of the two M-estimators.

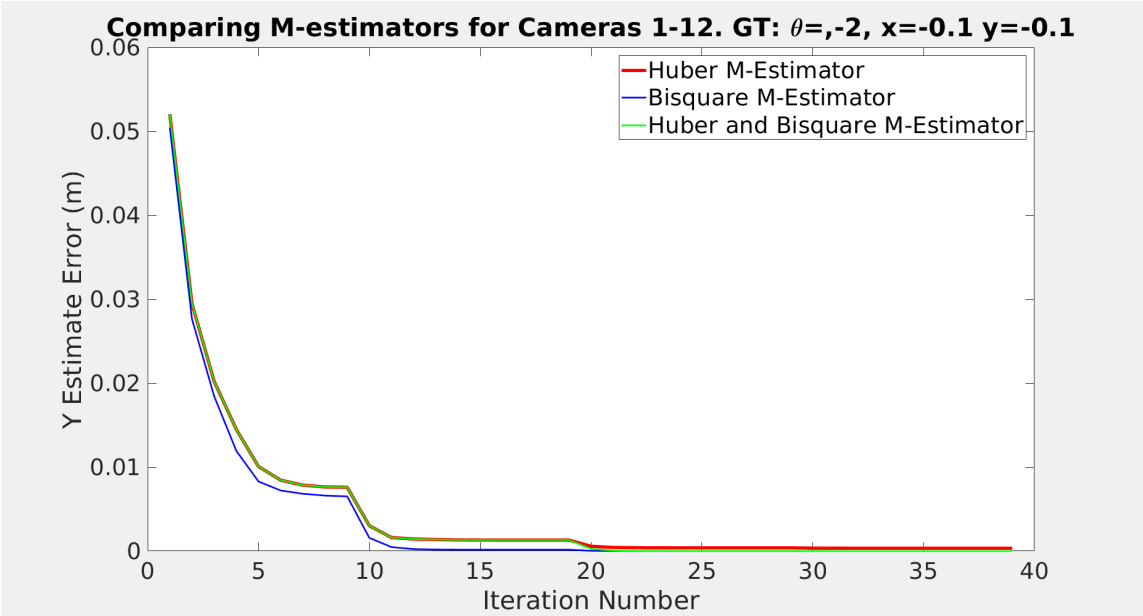
The first test is to compare the error at every iteration of the optimisation for the robust subset of cameras, $\{1\text{-}12\}$, when using the Huber M-estimator, bisquare m-estimator and a combination of both. All tests were run for 40 iterations with the tuning constant, k, being recalculated every 10 iterations. For the combined tests, the Huber M-estimator was used for the first 20 iterations and the bisquare M-estimator was used for the next 20 iterations.

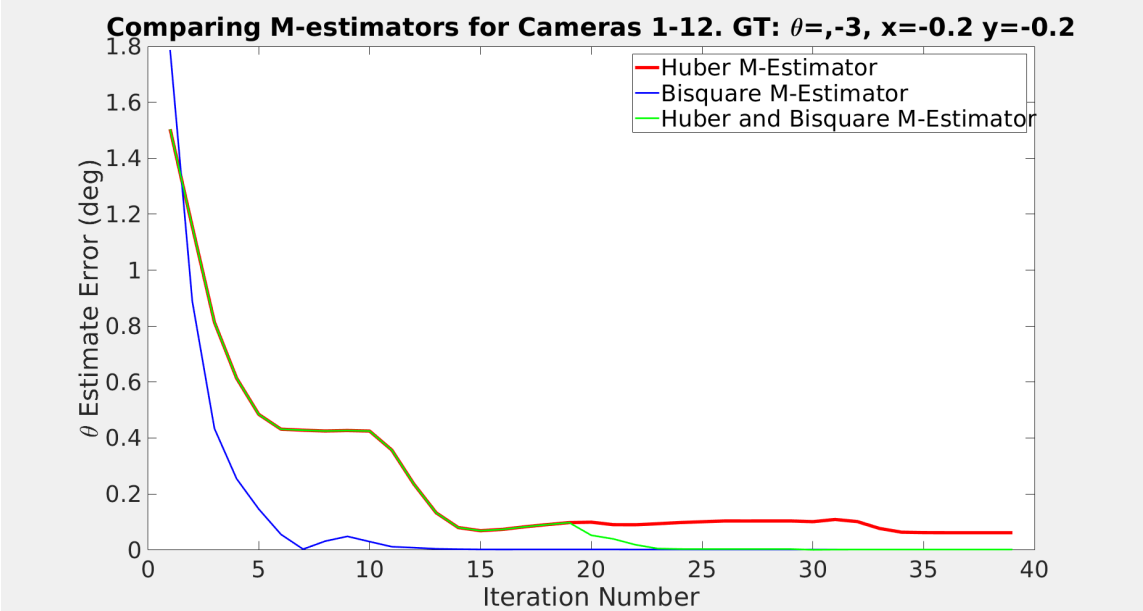
The expectation is that the Huber function will be less prone to local minima, while the bisquare function will converge on a more accurate final error. When stuck in a local minimum, there will still be point correspondences with large residuals due to the pose estimate being incorrect. The Huber function will still give these residuals a weight, albeit a small weight, which will give it a greater chance of leaving the local minimum. On the other hand, the bisquare function will set the weights of these residuals, or some portion of them depending on the tuning constant, to zero, and thus is more likely to get stuck in this minimum. However, when the optimisation is near the global minimum, the bisquare function should converge closer to the actual value as it sets the weights of most or all of the outliers to zero, while the Huber function only reduces their weights.

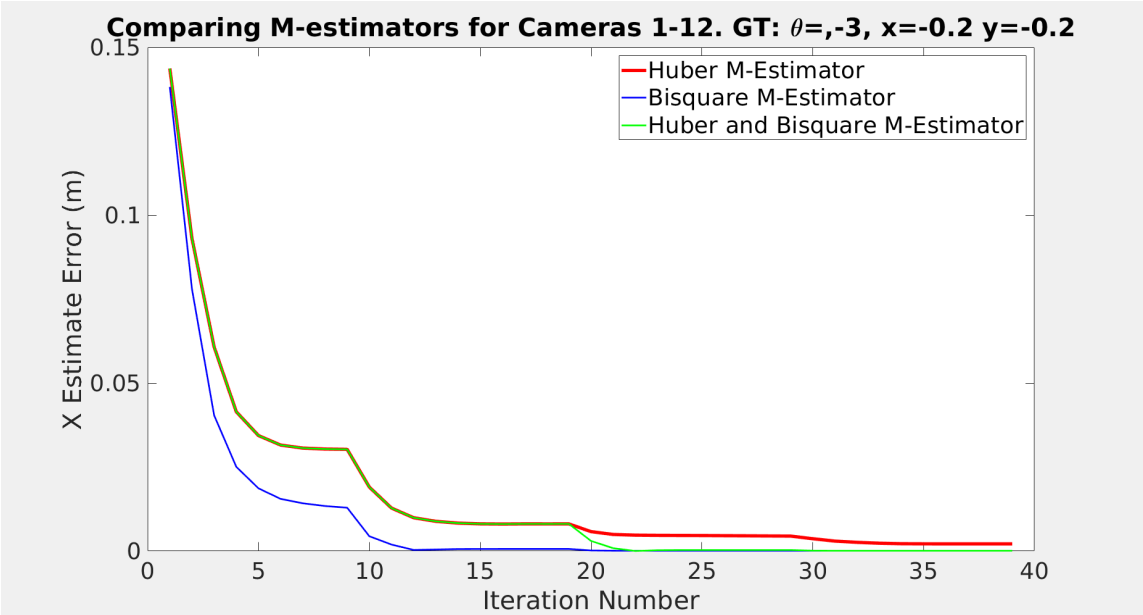
In the first set of results, the faster and more accurate convergence of the bisquare estimator compared with the Huber estimator is shown.

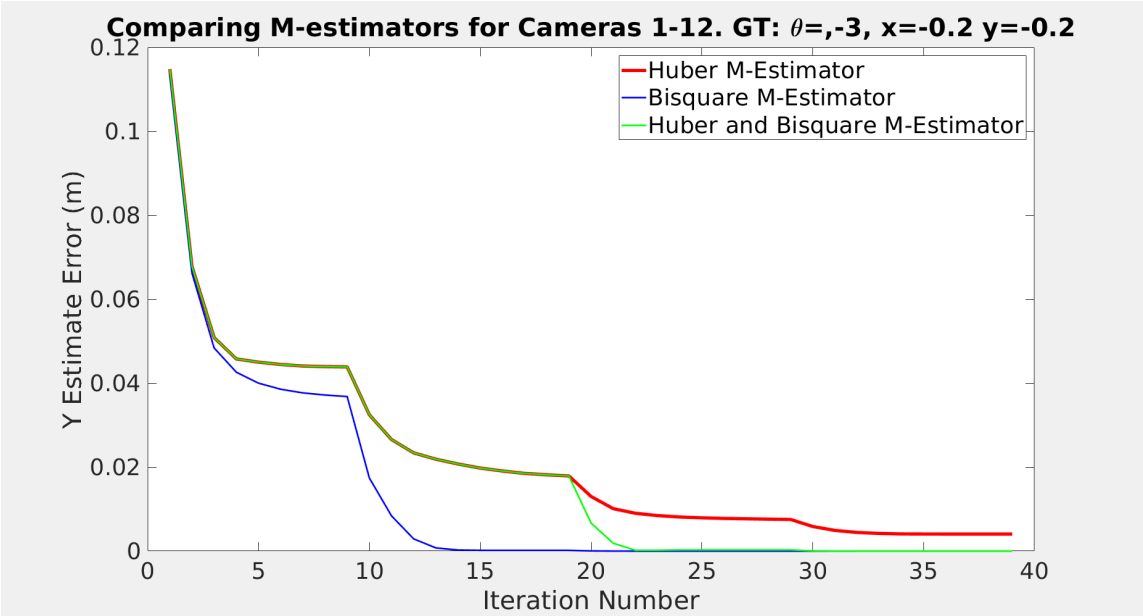

Comparing M-estimators for Cameras 1-12. GT: θ=,-2, x=-0.1 y=-0.1
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
θ Estimate Error (deg)
Iteration Number

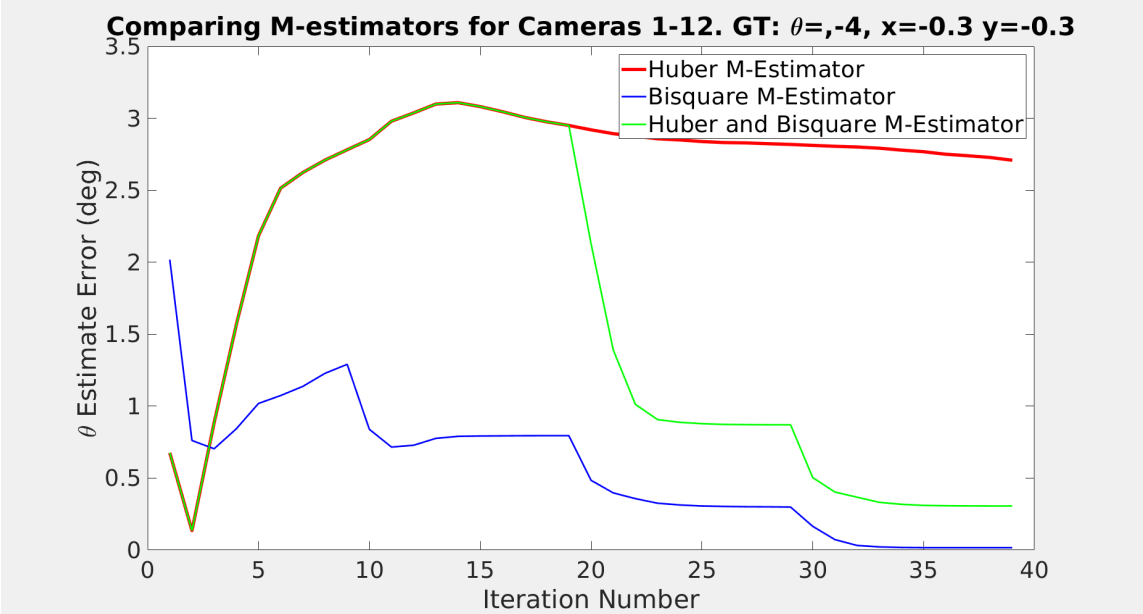

Comparing M-estimators for Cameras 1-12. GT: θ=,-2, x=-0.1 y=-0.1
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
X Estimate Error (m)
Iteration Number

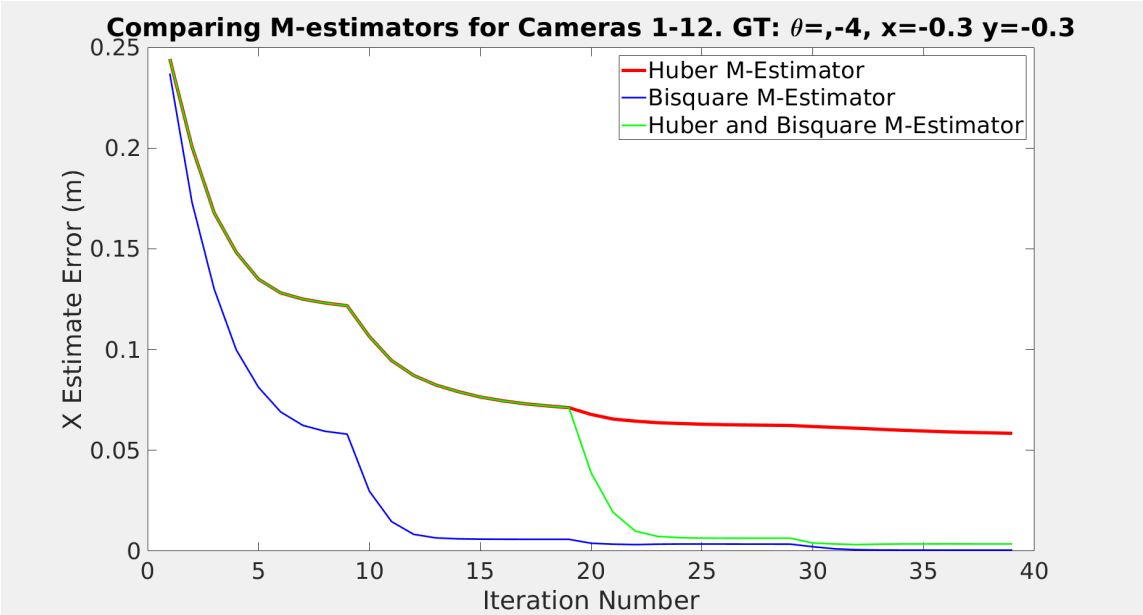

Comparing M-estimators for Cameras 1-12. GT: θ=,-2, x=-0.1 y=-0.1
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
Y Estimate Error (m)
Iteration Number

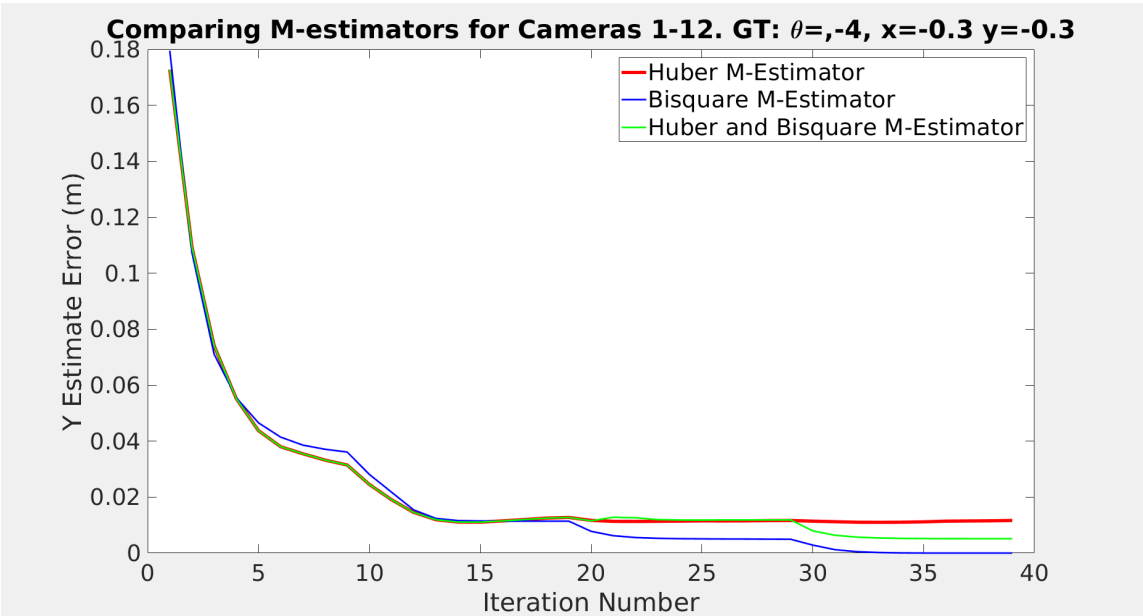

Comparing M-estimators for Cameras 1-12. GT: θ=,-3, x=-0.2 y=-0.2
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
θ Estimate Error (deg)
Iteration Number

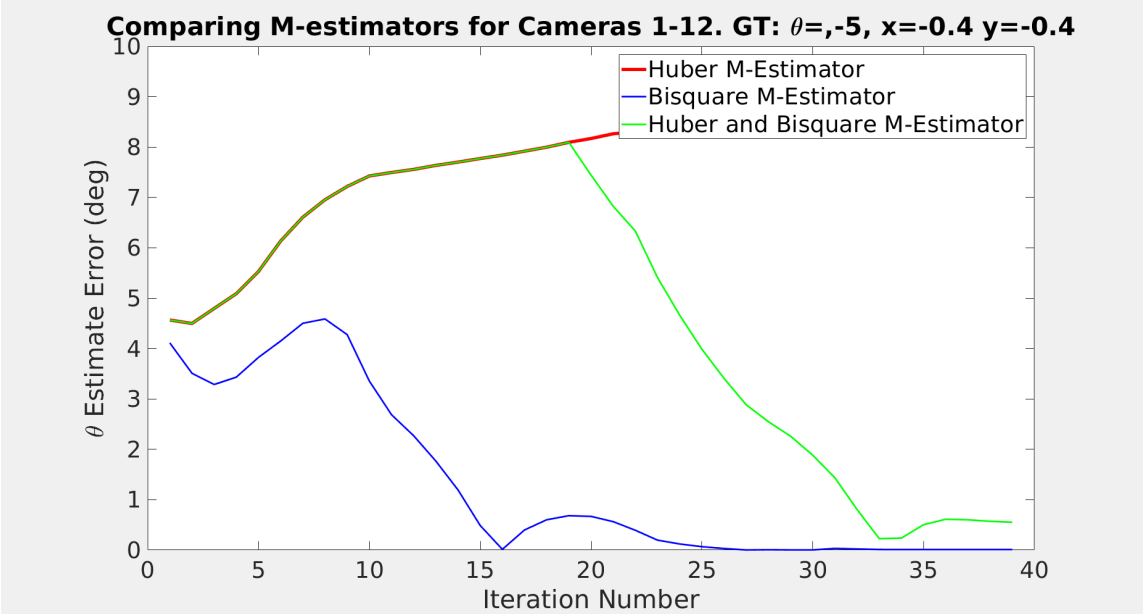

Comparing M-estimators for Cameras 1-12. GT: θ=,-3, x=-0.2 y=-0.2
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
X Estimate Error (m)
Iteration Number

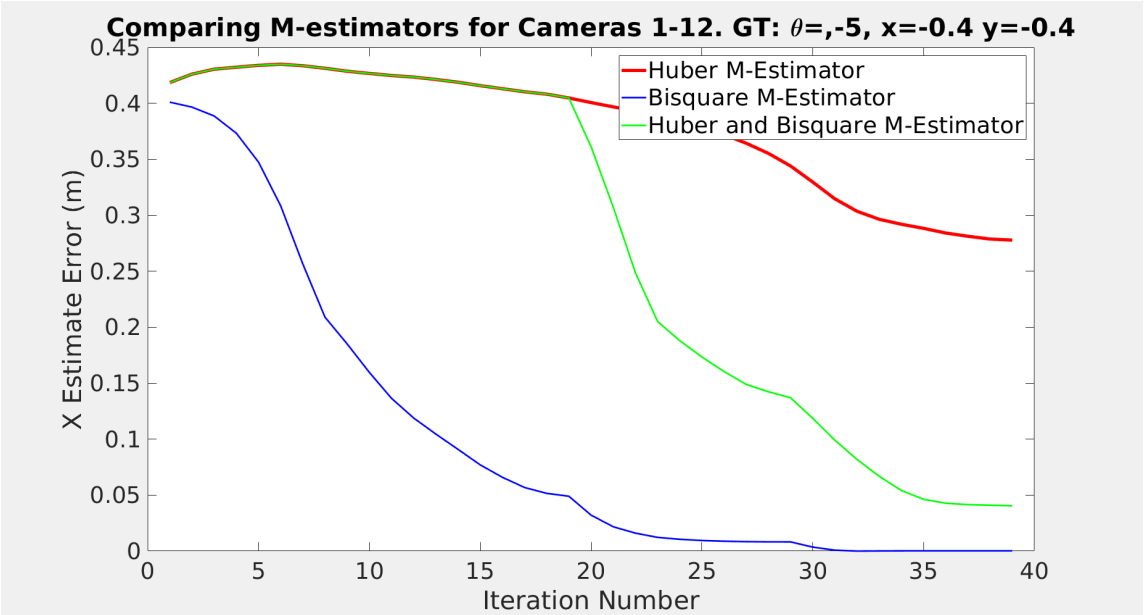

Comparing M-estimators for Cameras 1-12. GT: θ=,-3, x=-0.2 y=-0.2
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
Y Estimate Error (m)
Iteration Number

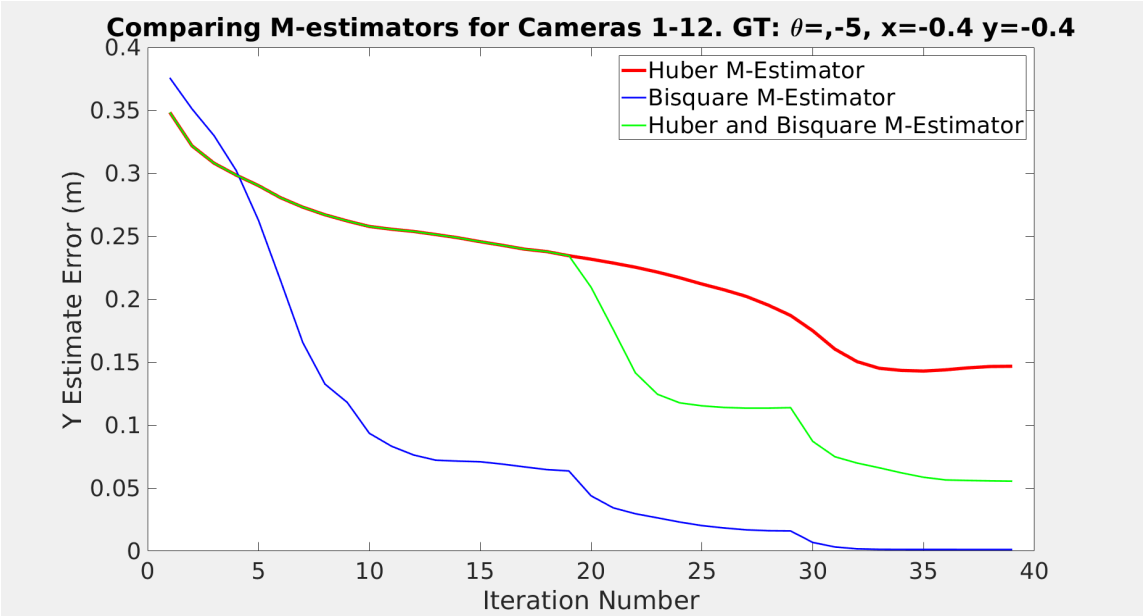

Comparing M-estimators for Cameras 1-12. GT: θ=,-4, x=-0.3 y=-0.3
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
θ Estimate Error (deg)
Iteration Number

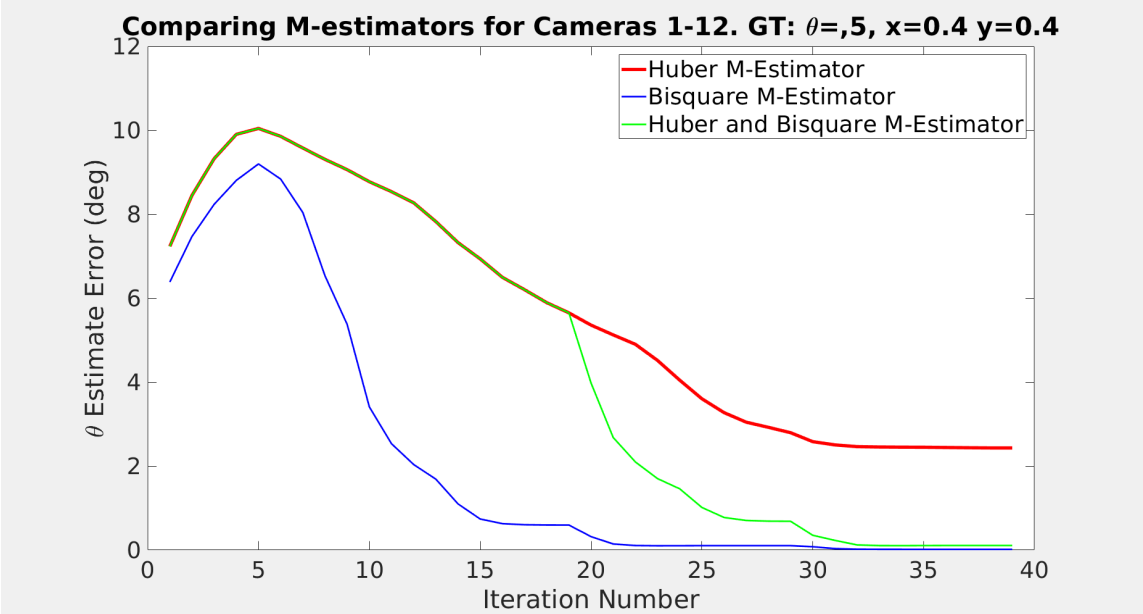

Comparing M-estimators for Cameras 1-12. GT: θ=,-4, x=-0.3 y=-0.3
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
X Estimate Error (m)
Iteration Number

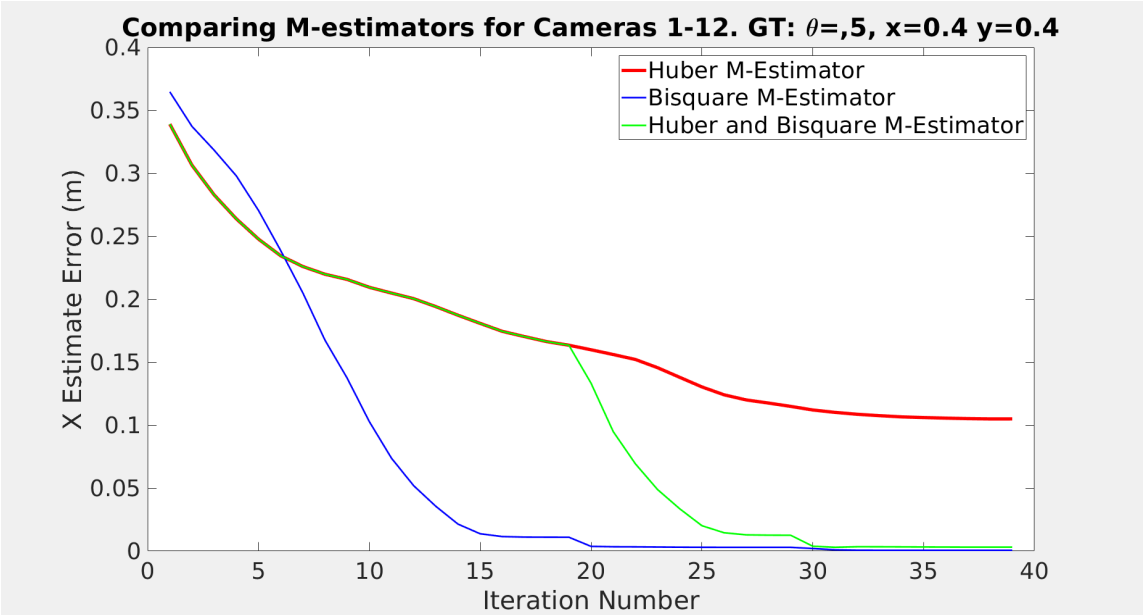

Comparing M-estimators for Cameras 1-12. GT: θ=,-4, x=-0.3 y=-0.3
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
Y Estimate Error (m)
Iteration Number

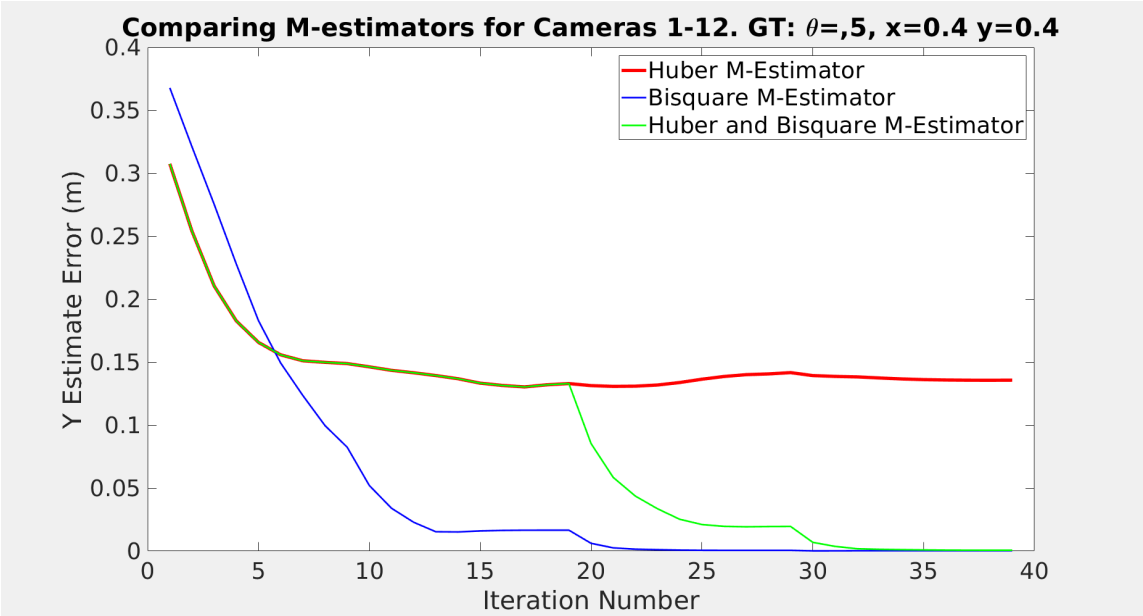

Comparing M-estimators for Cameras 1-12. GT: θ=,-5, x=-0.4 y=-0.4
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
θ Estimate Error (deg)
Iteration Number


Comparing M-estimators for Cameras 1-12. GT: θ=,-5, x=-0.4 y=-0.4
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
X Estimate Error (m)
Iteration Number

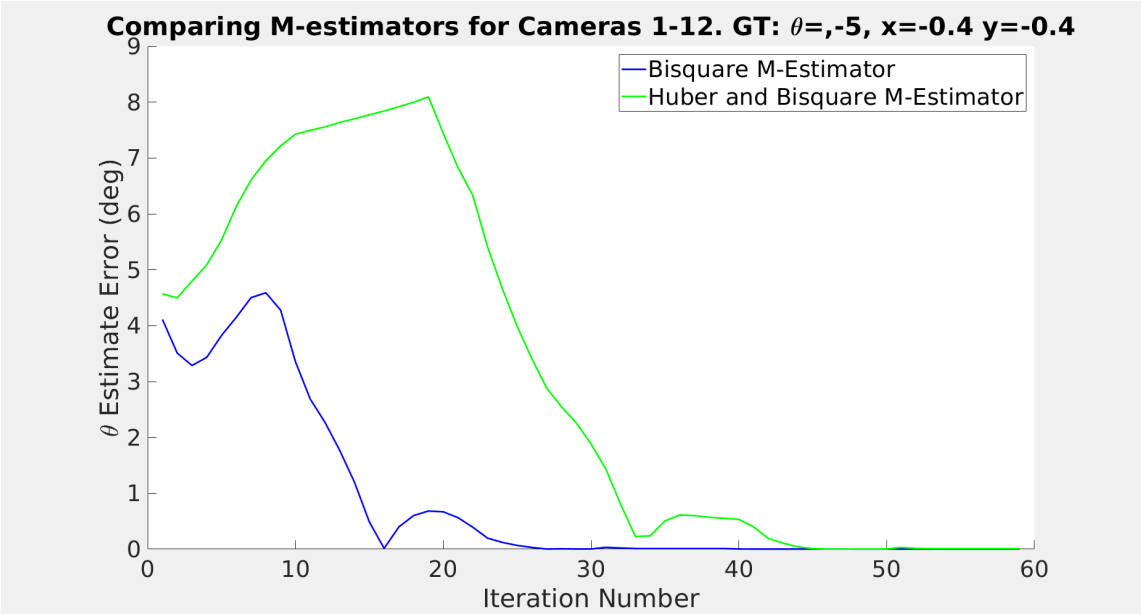

Comparing M-estimators for Cameras 1-12. GT: θ=,-5, x=-0.4 y=-0.4
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
Y Estimate Error (m)
Iteration Number

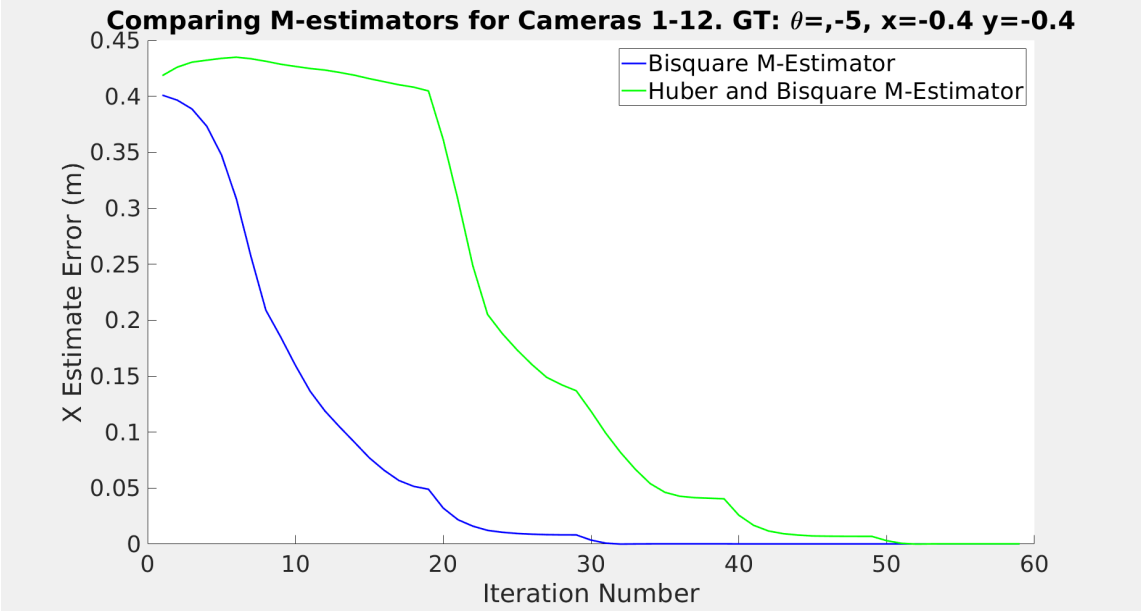

Comparing M-estimators for Cameras 1-12. GT: θ=,5, x=0.4 y=0.4
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
θ Estimate Error (deg)
Iteration Number

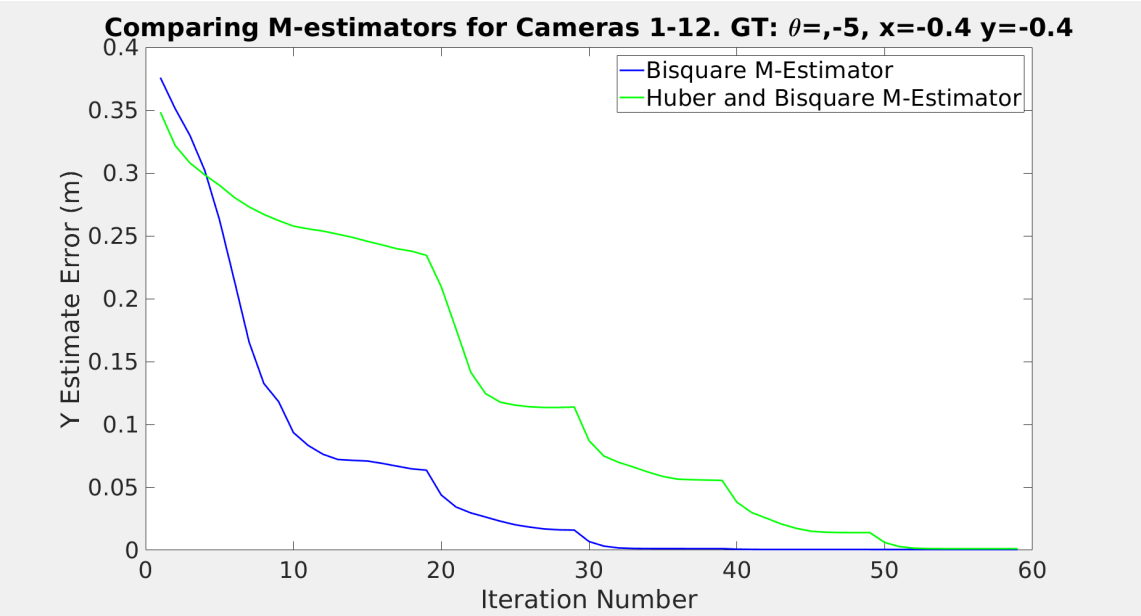

Comparing M-estimators for Cameras 1-12. GT: θ=,5, x=0.4 y=0.4
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
X Estimate Error (m)
Iteration Number


Comparing M-estimators for Cameras 1-12. GT: θ=,5, x=0.4 y=0.4
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
Y Estimate Error (m)
Iteration Number

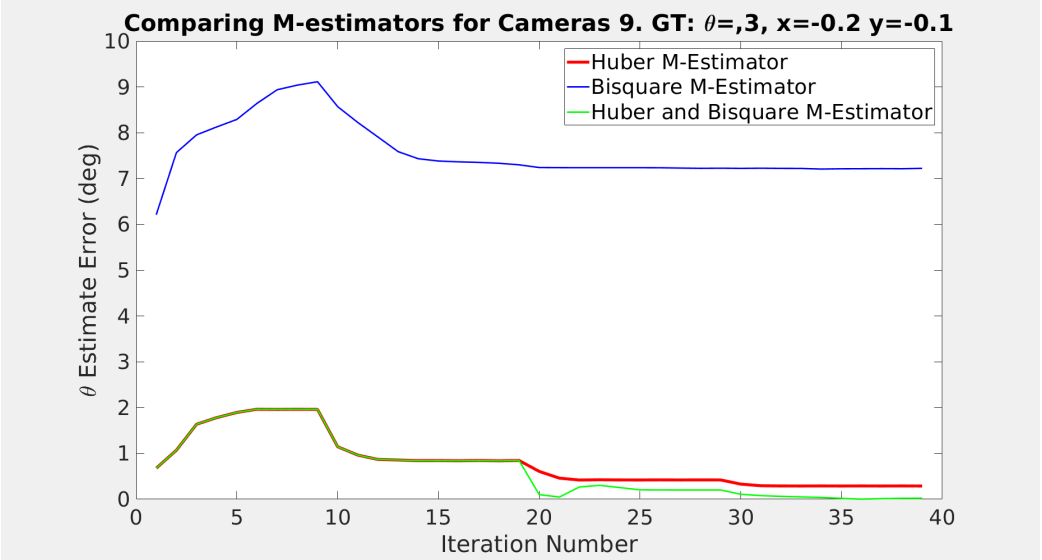
The reason why the final estimation error is better when using only the bisquare estimator rather using the combined estimators is due to the value of the tuning constant, k, which is found every 10 iterations. At iteration 30, since the MAR is lower using only the bisquare estimator, assuming the estimation is not stuck in a local minima, it will be removing more outliers than the combined estimators and therefore will converge on a lower error estimate. When the optimisation is run for more iterations, they converge on the same value, as can be seen below.

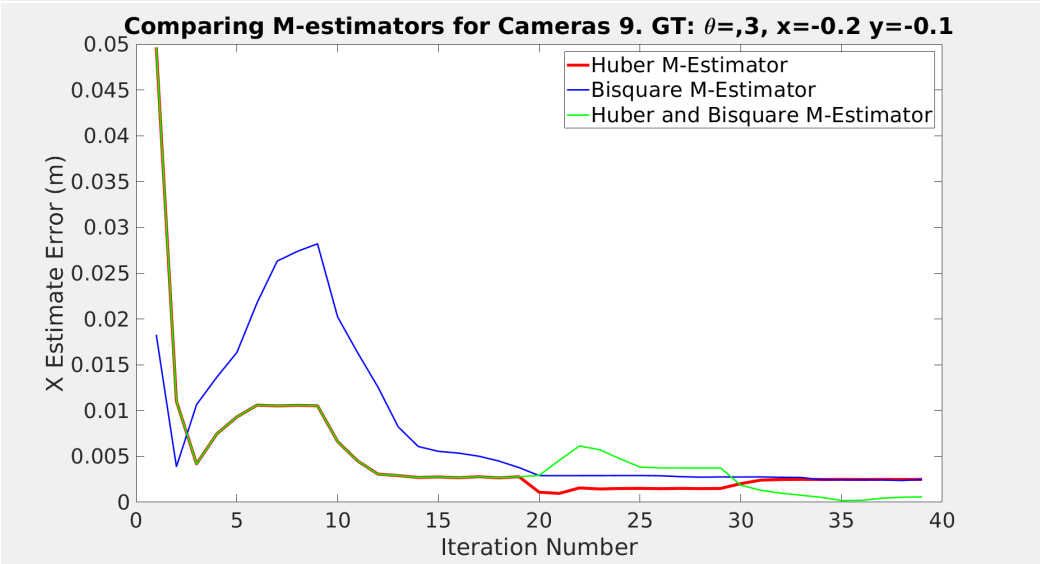


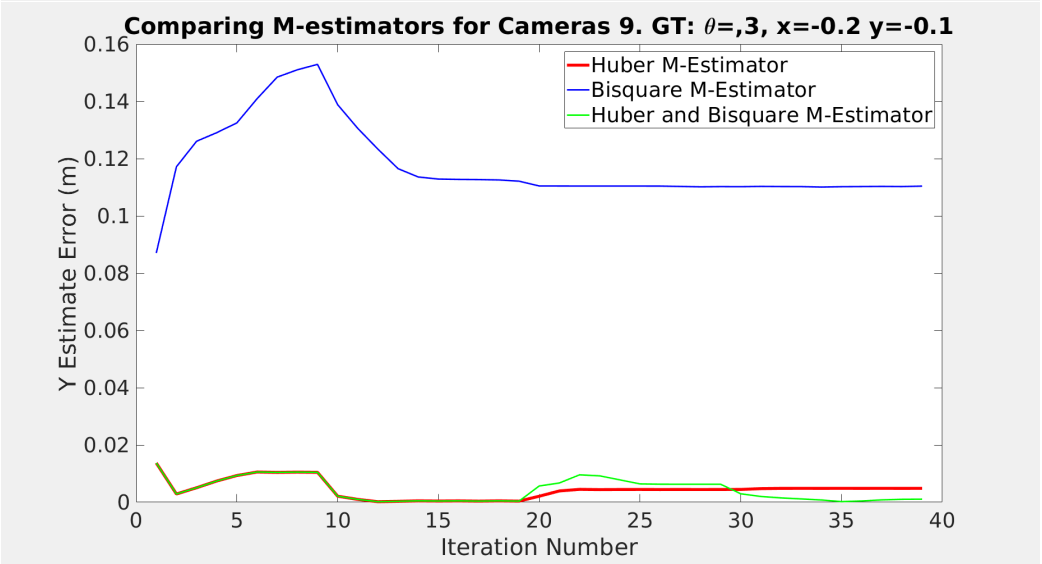


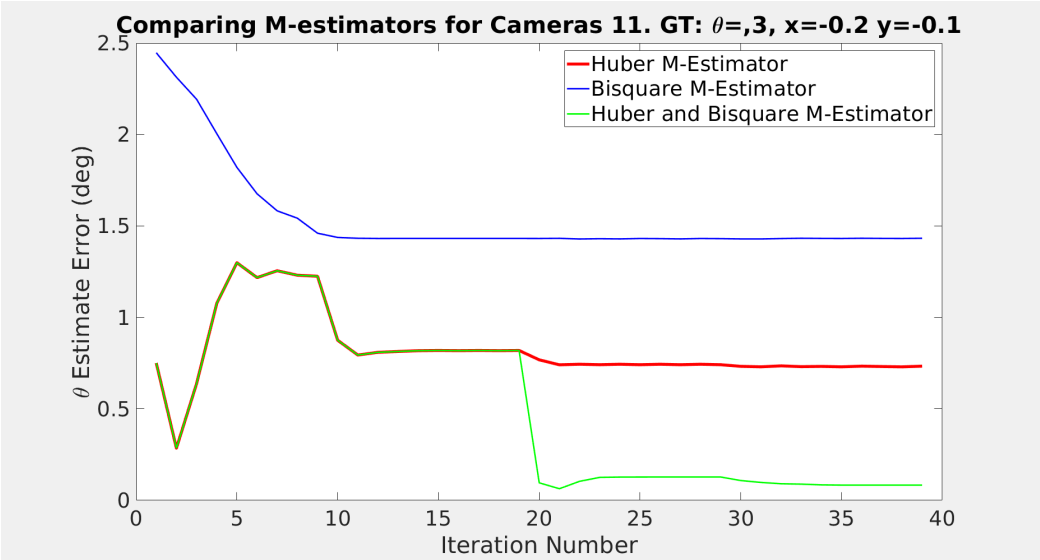


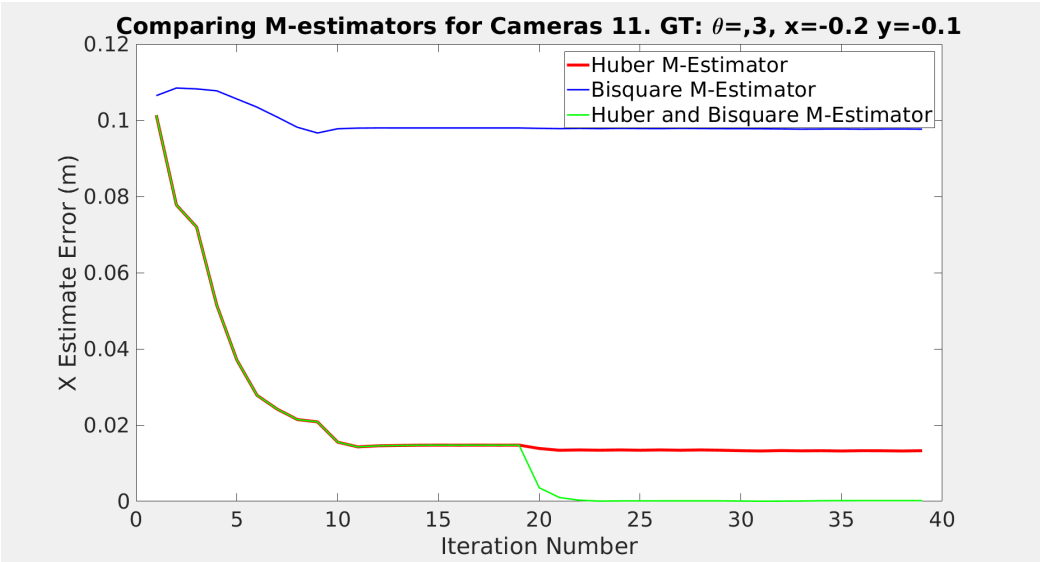
The results shown above indicate that the bisquare estimator outperforms the Huber estimator when the optimisation is not near a local minima. However, in cases where there are local minima, the bisquare estimator is more prone to getting stuck while the Huber estimator does a better job of finding the global minimum. The results below demonstrate this.

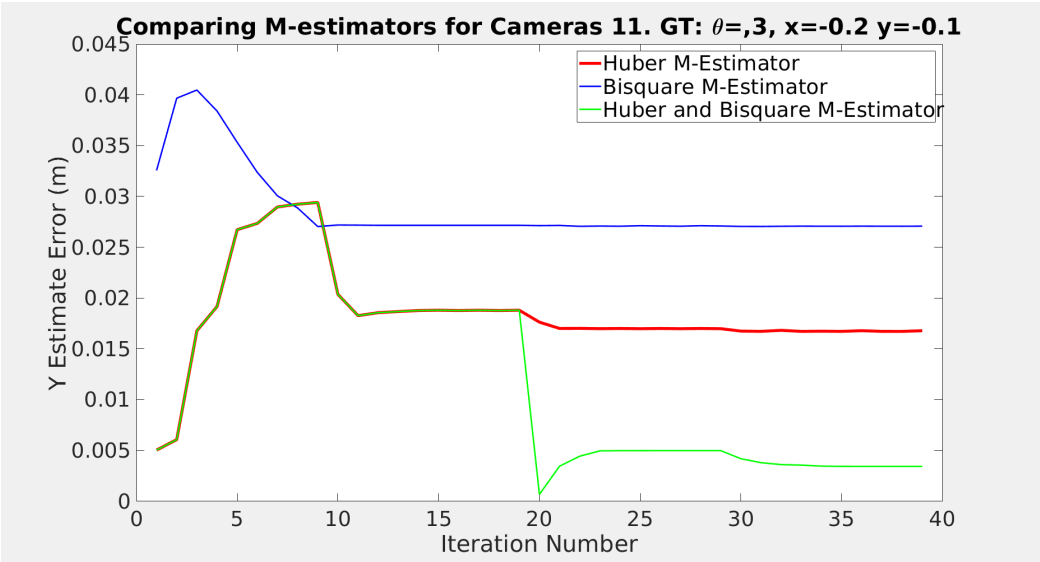

Comparing M-estimators for Cameras 9. GT: θ=,3, x=-0.2 y=-0.1
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
θ Estimate Error (deg)
Iteration Number

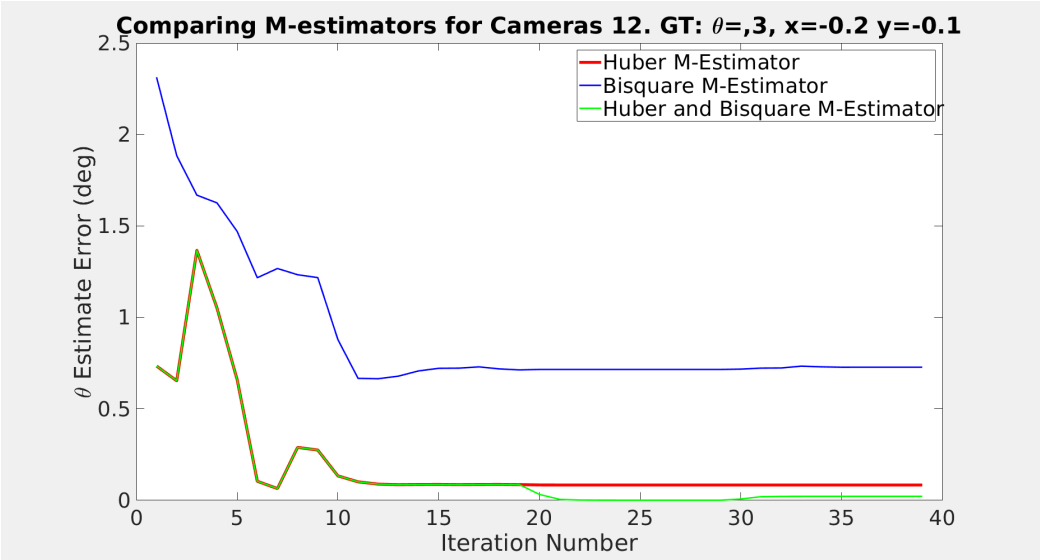

Comparing M-estimators for Cameras 9. GT: θ=,3, x=-0.2 y=-0.1
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
X Estimate Error (m)
Iteration Number

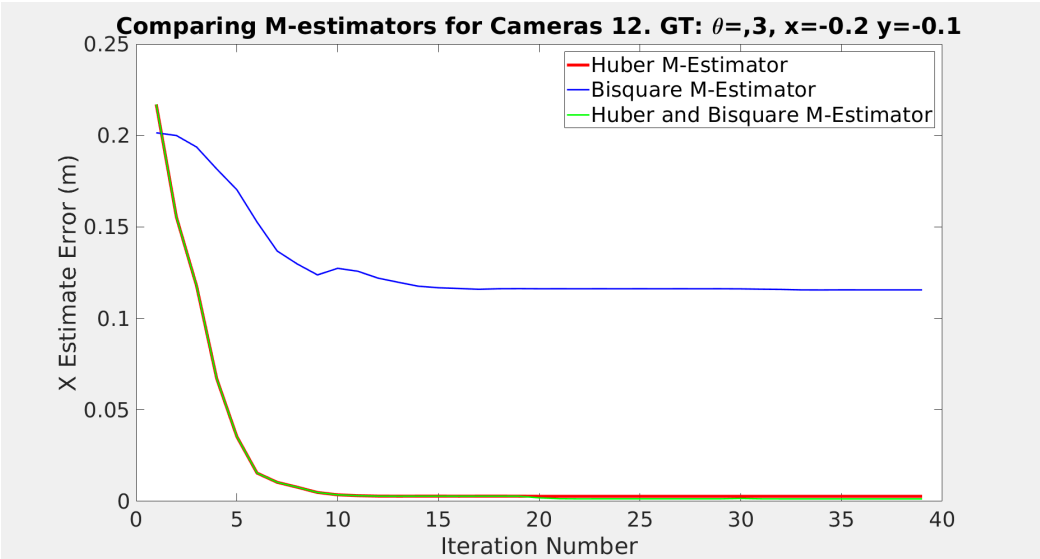

Comparing M-estimators for Cameras 9. GT: θ=,3, x=-0.2 y=-0.1
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
Y Estimate Error (m)
Iteration Number

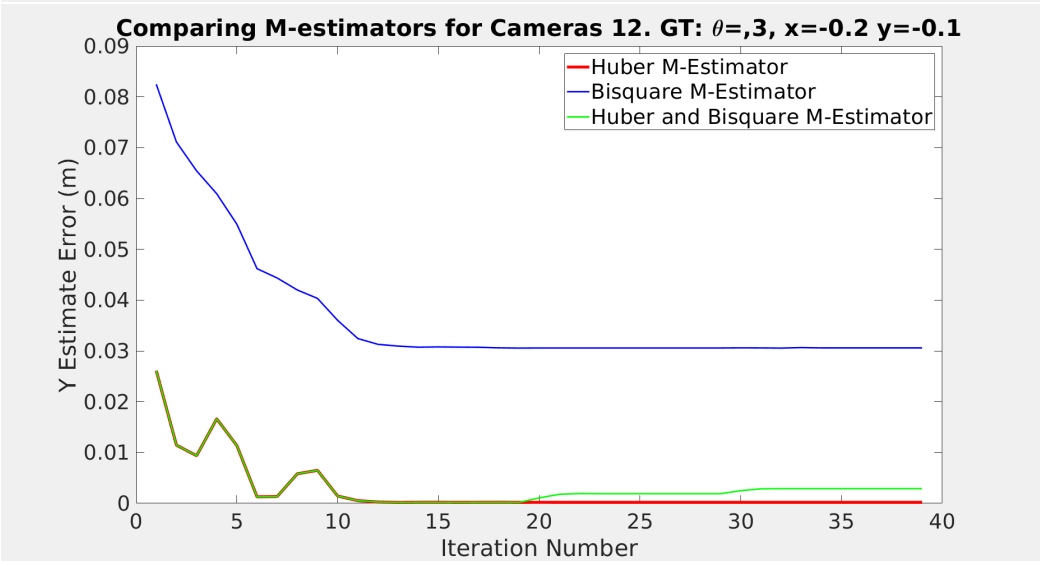

Comparing M-estimators for Cameras 11. GT: θ=,3, x=-0.2 y=-0.1
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
θ Estimate Error (deg)
Iteration Number


Comparing M-estimators for Cameras 11. GT: θ=,3, x=-0.2 y=-0.1
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
X Estimate Error (m)
Iteration Number

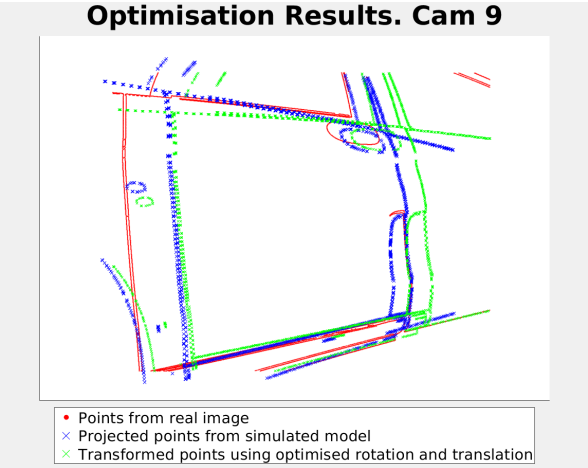

Comparing M-estimators for Cameras 11. GT: θ=,3, x=-0.2 y=-0.1
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
Y Estimate Error (m)
Iteration Number

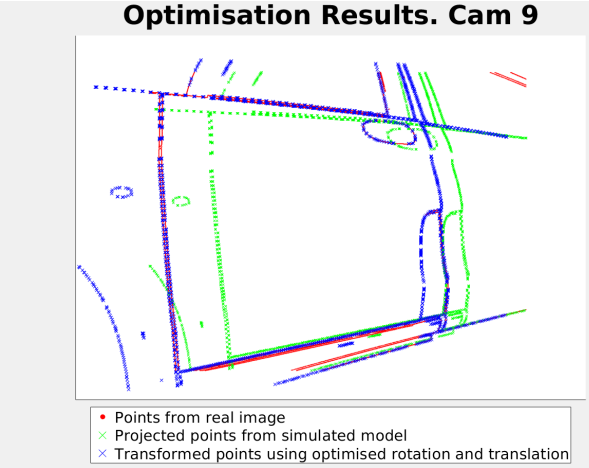

Comparing M-estimators for Cameras 12. GT: θ=,3, x=-0.2 y=-0.1
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
θ Estimate Error (deg)
Iteration Number

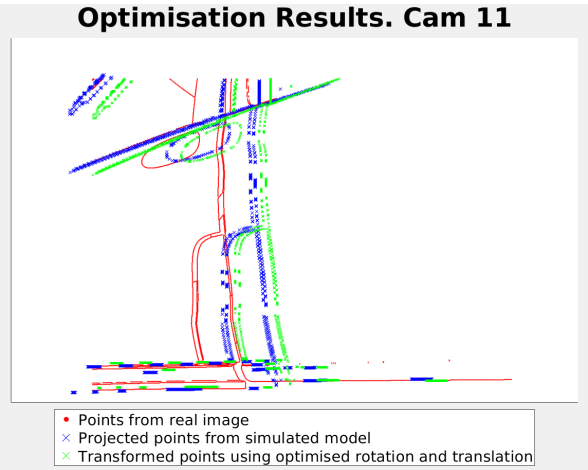

Comparing M-estimators for Cameras 12. GT: θ=,3, x=-0.2 y=-0.1
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
X Estimate Error (m)
Iteration Number

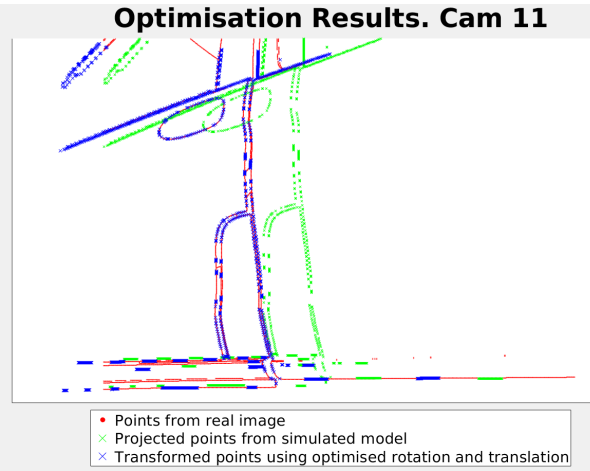

Comparing M-estimators for Cameras 12. GT: θ=,3, x=-0.2 y=-0.1
Huber M-Estimator
Bisquare M-Estimator
Huber and Bisquare M-Estimator
Y Estimate Error (m)
Iteration Number

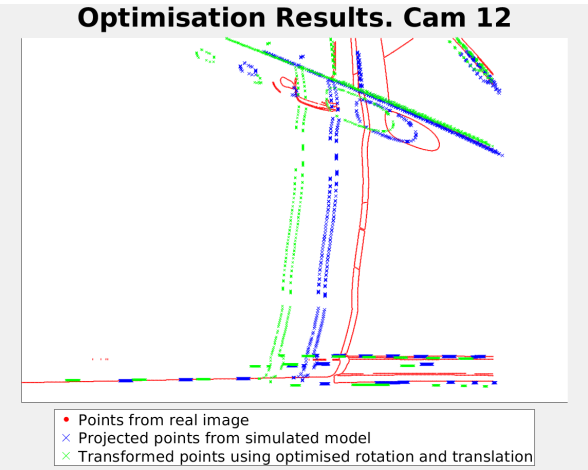
The resulting plots of the transformed points using the optimised pose estimate show the local minima that the optimisation got stuck in using only the bisquare function for cameras 9, 11, and 12. The first column shows the optimised results using only the bisquare estimator and the second column shows the results using the combined estimators.

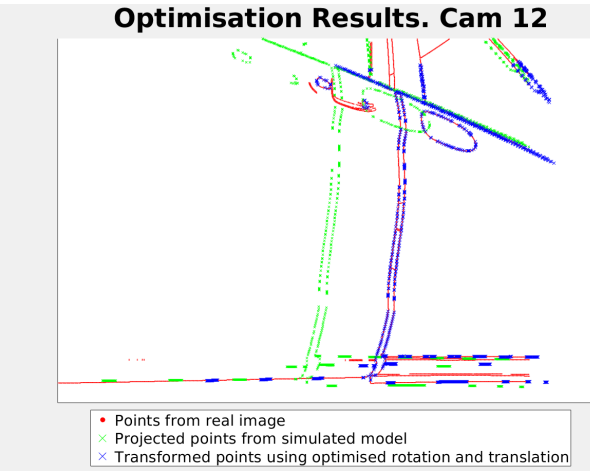












Chapter 5

Discussion

5.1 Robust Estimation

5.1.1 M-Estimation

Least squares estimation assumes that the residuals of the cost function are normally distributed. Outliers in the data break this assumption of normality as they tend to follow a longer-tailed distribution than a Gaussian. Least squares estimation is sensitive to these outliers as they square the residual, increasing the influence of the outlier on the optimisation. Therefore, if a single outlier is located sufficiently far from the rest of the data, it can completely spoil the least squares analysis [37]. M-estimators are a group of functions which relax the assumption of normality and make the estimation more robust to outliers. When an estimator is robust, it can be inferred that the influence of any single observation will not have a significant impact on the result [38].

A suitable M-estimator, $\rho(x)$ should have the following properties [36]:

- Always non-negative, $\rho(x) \geq 0$
- Equal to zero when its argument is zero, $\rho(0) = 0$
- Symmetric, $\rho(x) = \rho(-x)$
- Monotone in $\mid x_i \mid$, $\rho(x_i) \geq \rho(x_{i'})$ for $\mid x_i \mid > \mid x_{i'} \mid$

M-estimator functions work by applying a weight to the residuals, giving larger residuals a smaller weight. The specific values of these weights and how they decrease as the residual grows depends on the function used. The two M-estimators used in this work are the Huber and Tukey bisquare functions. The Huber function behaves as a parabolic least-squares function as long as it is below a threshold specified by the tuning constant. For residuals larger than the tuning constant, it behaves linearly like a Least Absolute Error (LAE) function. The bisquare function increases as the residual grows up until the tuning constant, after which it plateaus.

The Huber function assigns a weight of 1 to all residuals less than the tuning constant and decreasing weights to all residuals greater than the tuning constant. The bisquare function assigns decreasing weights as residuals grow up to the tuning constant, after which all residuals are assigned a weight of zero. A comparison of the M-estimator functions, the influence functions and the weight functions for least-squares, Huber and bisquare are shown in Fig. 5.1 and a more comprehensive overview is covered in [39].

While we have tested using Huber and bisquare M-estimators, student-t has been shown to consistently work well for various datasets in [40] and therefore should be investigated in future work.

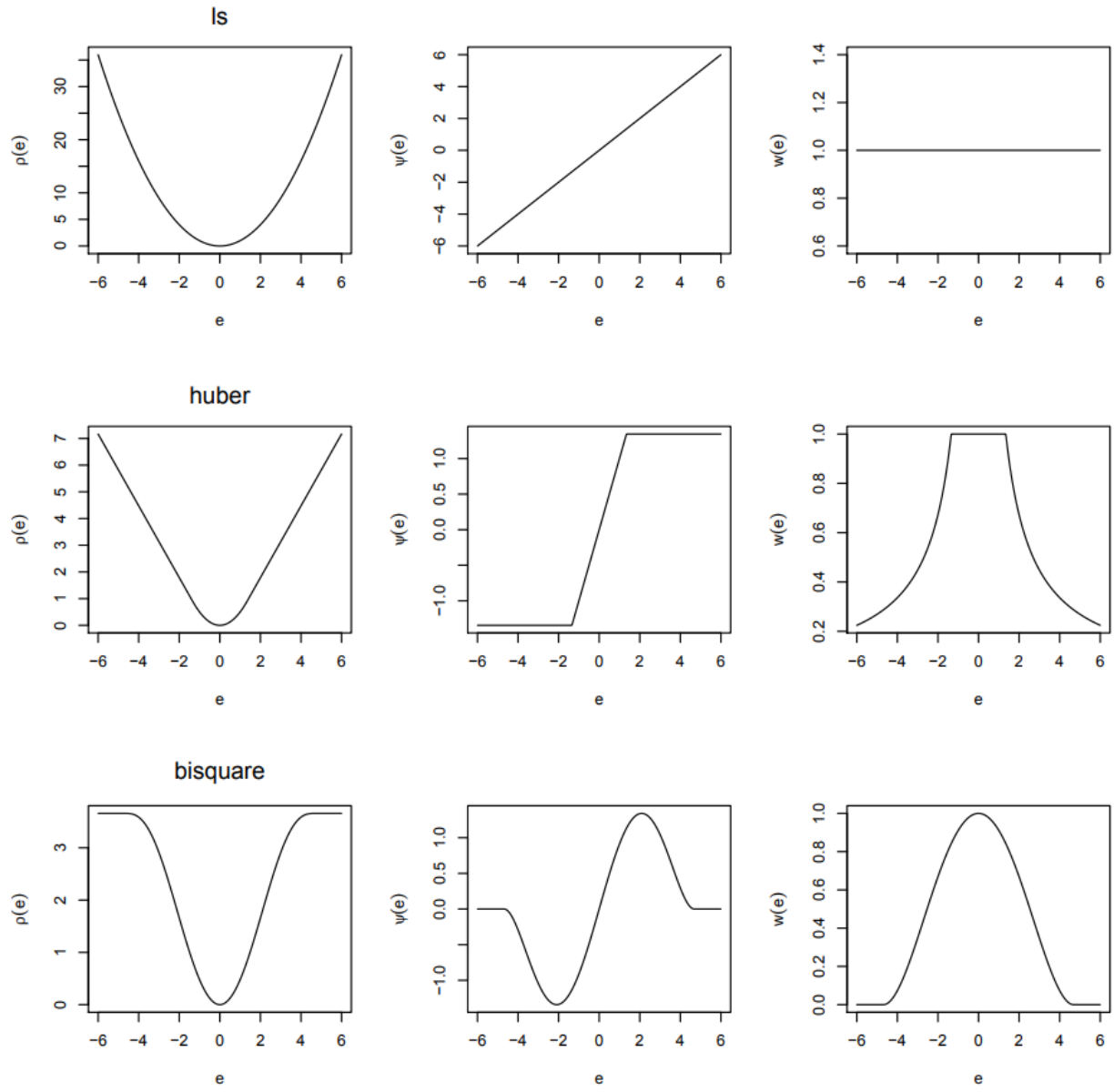


Figure 5.1: Comparison of objective ($\rho(e)$), influence ($\psi(e)$) and weight ($w(e)$) functions for least-squares (top), Huber (middle), and bisquare (bottom) estimators. [36]

5.1.2 Trimming Point Correspondence

ICP makes the assumption that there is a one-to-one point correspondence between the two sets of points being matched. Therefore, a solution is proposed in [41] that trims the correspondences based on the distance between the matched points. The process for trimming is as follows:

1. All point correspondences across each camera are aggregated into a single matrix containing the point indices and the correspondence distances.
2. The point correspondences are sorted according to the distance between corresponding points in ascending order.
3. The first $\gamma\%$ of values are taken and used in the optimisation.

Since each camera has a different number of outliers, trimming points for each individual camera could result in removing too many points from good images and not removing enough points from bad iamges. So we aggregated the point correspondences across every camera so that the good images would have fewer points trimmed and therefore would have a greater impact on the optimisation.

One of the issues with this method is that the parameter, γ, is set based on an estimate of the percentage overlap of the two point sets. Since the level of overlap between the real and simulated camera images depends on the pose offset of the vehicle, it makes it difficult to choose a robust value for this parameter. Additionally, it is widely agreed that correcting the data before parameter estimation, in this case by trimming outliers, is a poor choice. According to Huber and Ronchetti, there are two significant issues associated doing this: "(a) outliers have to be determined correctly or the situation will even get worse (usually there will be both false rejections and false retentions), and (b) outliers may obscure each other so that they cannot be detected." [42]. Therefore, we instead use an M-estimator which reweights rather than removing residuals at every iteration of the optimisation.

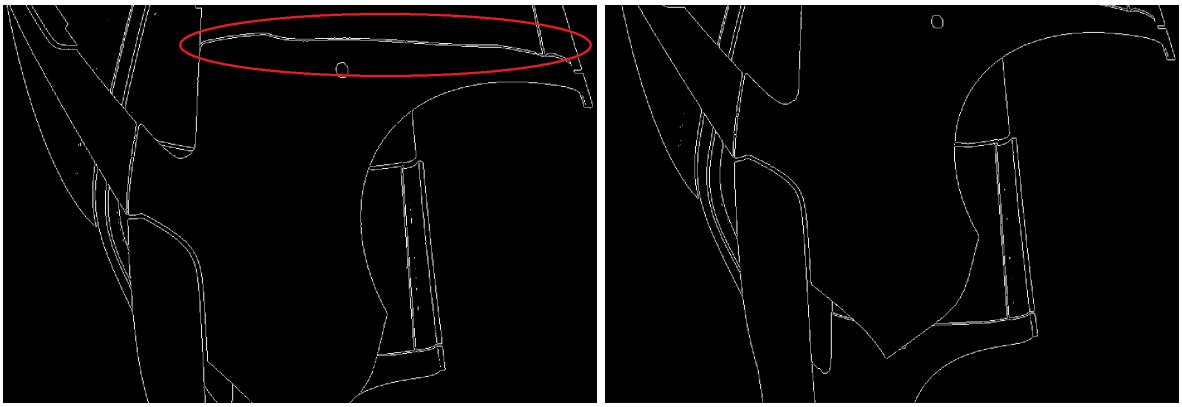


Figure 5.2: Edge that was observed in the real camera image (left) has been occluded in the simulated camera image (right), due to vehicle motion.

5.2 Direction of Point Correspondence

During the point correspondence step, the k-nearest neighbour search finds 2 nearest neighbours in the data set for every element in the query set. This means that the number of correspondences will be equal to the size of the query set. In our project, we need to find nearest neighbours between the edges of the real camera images and the projection of the edges of the simulated camera images. We therefore need to decide which of these will be the data set and which will be the query set.

Both options have advantages and disadvantages which are summarised below:

Setting the real camera edges as the query set

In this scenario, we will be finding point correspondences for every point in the real camera edge images. The advantage of this option is that we have control over the simulated edges since they are derived from the CAD model. This means that we can choose the bounding box for the edges that are detected by modifying the simulated camera. For example, we can widen the field of view of the simulated camera so that it captures more edges. This means that if the real vehicle has been transformed such that an edge that appeared in the simulated camera image does not appear in the real camera image (See Fig. 5.2), increasing the width of the viewpoint could capture that missing edge.

However, the issue with this approach is that the nearest neighbour search will find corresponding points for any spurious edges detected from the real cameras. This could be caused by the real vehicle having more detail than the model (See Fig. 5.3), or if there are spurious edges detected on the vehicle. Additionally, the cameras capture part of the background of the viewing tunnel which will also produce edges in the final image. This will have an even greater impact on the optimisation since these points are guaranteed to not have actual point correspondences in the simulated edge image so the resulting correspondences are likely to be at a larger dis-

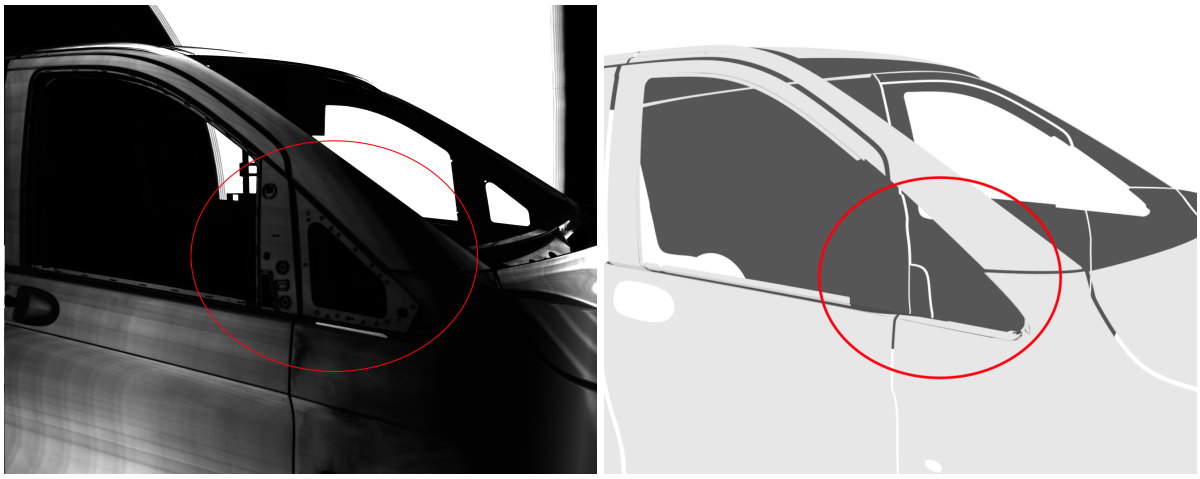


Figure 5.3: Left: Real image of vehicle. **Right**: Blender model of vehicle which lacks some of the detail that is in the real image.

tance. This introduction of large, spurious distances in the point correspondences will result in us relying on the system's robustness to outliers to get a good pose estimate. One possible solution for dealing with this is to find a way to automatically mask the background in each real camera image.

Setting the simulated camera edges as the query set

In this scenario, we will be finding point correspondences for every point in the simulated edge images. Since we have *a priori* knowledge about the simulated edge images from the CAD model, we can tune the edge detection (or manually select edges if using Blender's Freestyle toolbox) to ensure that there is minimal noise and spurious edge detections. This means that we will not have the same issue of using spurious edges as query points as in the previous option. Fig. 5.4 compares the point correspondences using the two options. Another advantage is that we can limit the depth of the simulated camera so that it does not detect internal edges of the vehicle. These edges are not useful because often transformations of the car cause edges on the interior of the vehicle to be occluded in the real camera images.

The disadvantage of this option is that we don't have any control over the real camera images. This means that if the vehicle has been moved such that an edge visible in the simulated camera image is not visible in the real camera image, there's nothing we can do.

Ultimately, since the required tolerance of the system is approximately ±2cm, there will not be a significant amount of edges that are missing from the simulated edge images due to movement of the vehicle. The noise in the real edge images is much more likely to be a issue. Therefore, we currently use the second option of setting the simulated edges as the query set.

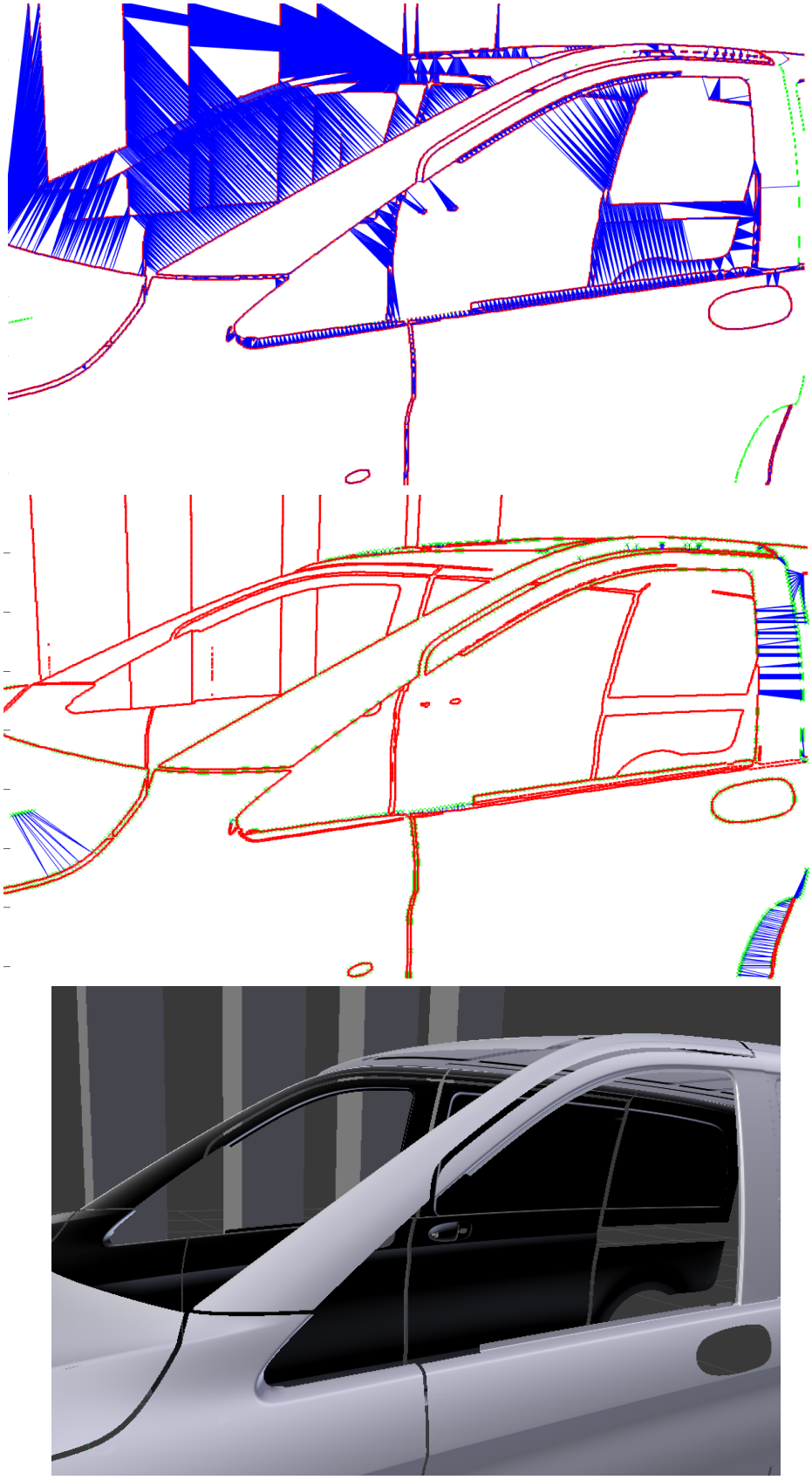


Figure 5.4: Top: Correspondences when the real camera edges are the query set. **Middle**: Correspondences when the simulated camera edges are the query set. **Bottom**: Original image from which we detect edges and point correspondences.

Figure 5.5: Left: Light strip can be seen behind the vehicle. **Right**: Sections of the vehicle directly in front of the light strip produce strong edges.

5.3 Edge Detection

The goal of the edge detection is to maximise the number of edges in the real camera image that exist in the simulated camera image. This is because, as discussed in Sec. 5.2, when we calculate the nearest neighbour point correspondences, we query every point in the simulated camera image and find their neighbours in the real camera image. So any point that exists in the simulated edge image that does not have a true correspondence in the real edge image will be an outlier in the optimisation. On the other hand, any point that exists in the real edge image that does not have a true correspondence in the simulated edge image will add noise to the point correspondences and may lead to local minima, but should not impact the optimisation once it is near the global minimum.

5.3.1 Detecting Edges in Real Images

After inspecting the real images supplied from the cameras in the manufacturing plant and testing various edge detection methods, we have identified a few key points:

- The Blender model and real images of the vehicle differ slightly as shown in Fig. 5.3. This could either be because the model provided to use is not the correct model for the images provided or it could be that the model simply has certain details omitted.
- Since the images we are using are the superimposition of a series of images taken as the tunnel lights move along the vehicle, certain images contain a strip of constant light as the backdrop to the vehicle. The portions of the vehicle that are in front of these light strips produce very strong edges as shown in Fig. 5.5.
- Edges on the vehicle which are clearly prominent in the CAD model may not show up in

Figure 5.6: Shows the effect of lighting on the edges extracted using Sobel detector.

the real images due to the lighting conditions.

- There is a trade-off when choosing the threshold for the edge detector between minimising noise in the image and ensuring that prominent edges are still detected. The thresholds therefore have to be chosen by testing how well the optimisation deals with these two cases. Since the vehicle will be ±2cm, it's likely that noise in the image will not cause issues with the optimisation getting stuck at local minima which can occur at larger vehicle offsets. Therefore, prioritising detection of prominent edges at the expense of more noise in the image is likely to be preferable.

Since we can gain a reasonable idea of which edges will be detected in each real image, this can help us choose which edges should be detected in our simulated camera images using Blender's Freestyle toolbox. The two primary considerations are that, firstly, edges against the light strip should be prioritised as they are strong edges and consistently detected. For example, camera 5 produces prominent edges from the interior of the vehicle since it's against the light strip. Secondly, edges on the vehicle surface are dependent on the lighting conditions in that area of the vehicle. Less prominent edges in well lit areas are more likely to be detected than more prominent edges in poorly lit areas. This can be seen in Fig. 5.6 in which the wheel arch would normally be a more prominent edge than those on the face of the door, however, due to the poor lighting the wheel arch isn't detected at all. By taking both of these into account, we can carefully select the edges in the simulated camera images to maximise point correspondences with the real image points.

5.3.2 Sobel v Canny Edge Detectors

For the real camera images, we currently use a Sobel edge detector with a manually chosen threshold for each camera. According to [43], "Sobel is optimum for objects with strong edges",

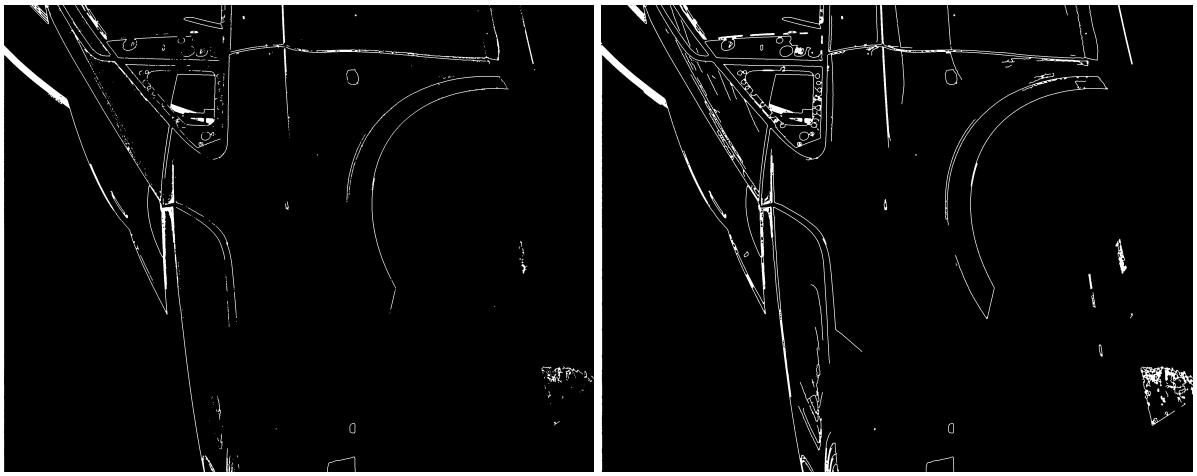
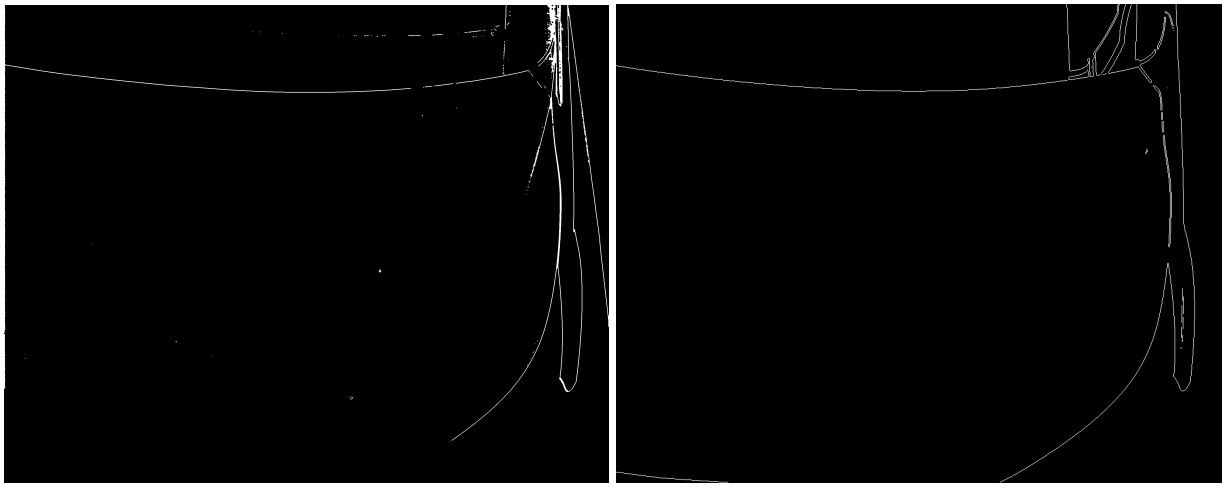


Figure 5.7: Left: Edge image using Sobel edge detector. **Right**: Edge image using Canny edge detector.

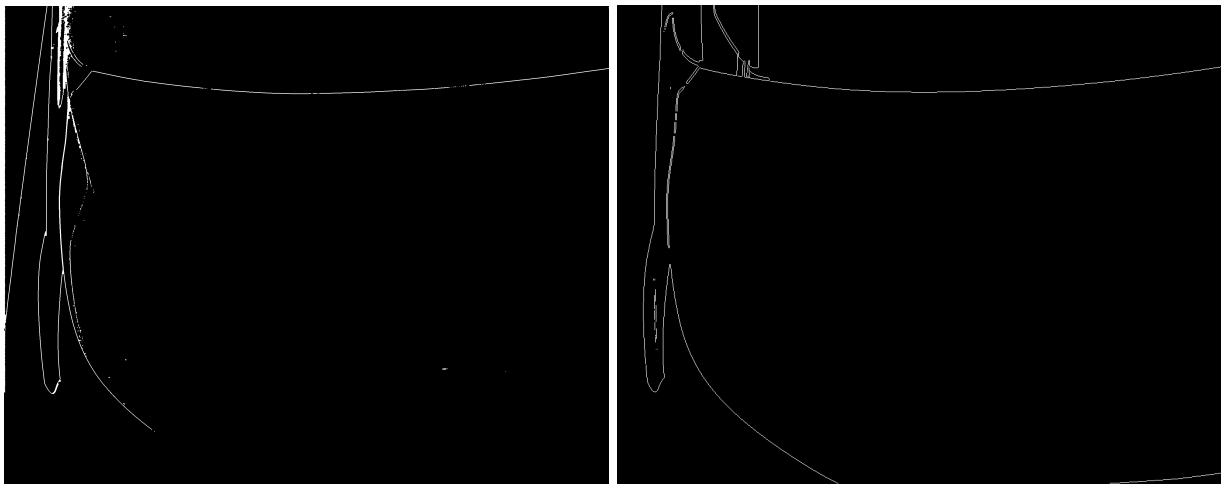
while they suggest that the "Canny method outperforms all other methods". We have found that the Canny detector indeed does detect more edges, however, the Sobel detector tends to produce fewer less prominent edges. Since Canny detects more edges, the noise (in this case edges that do not exist in the simulated edge camera images) tends to be lines while in the Sobel image they tend to be points. This is evident in Fig. 5.7 which compares the two edge detection methods. If we find that noise in the edge image affects the optimisation, it will be easier to filter out small points in the Sobel image rather than identifying insignificant edges in the Canny image. Therefore, at this stage, we use the Sobel edge detector. However, further testing should be done once we have access to real data to compare how each method affects the optimisation.

5.3.3 Comparing Real v Simulated Images

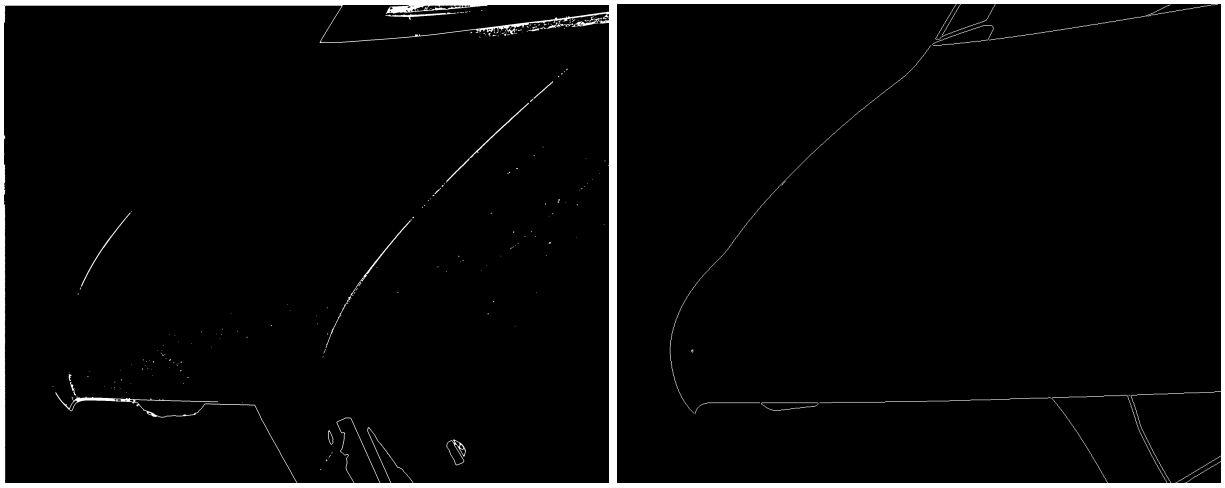
Since we don't have the intrinsic parameters of the real cameras in the manufacturing plant, we cannot test our system using the real images. However, we make a visual comparison below between the edges extracted from the real camera images taken from the plant, and the simulated camera images from the Blender simulation. All edges are extracted using the Sobel detector. While the simulated images clearly produce less noise, many of the prominent edges still appear in the real images. We also have the benefit of being able to see which edges are detected in the real images and manually select these edges in the simulated images to ensure they appear.



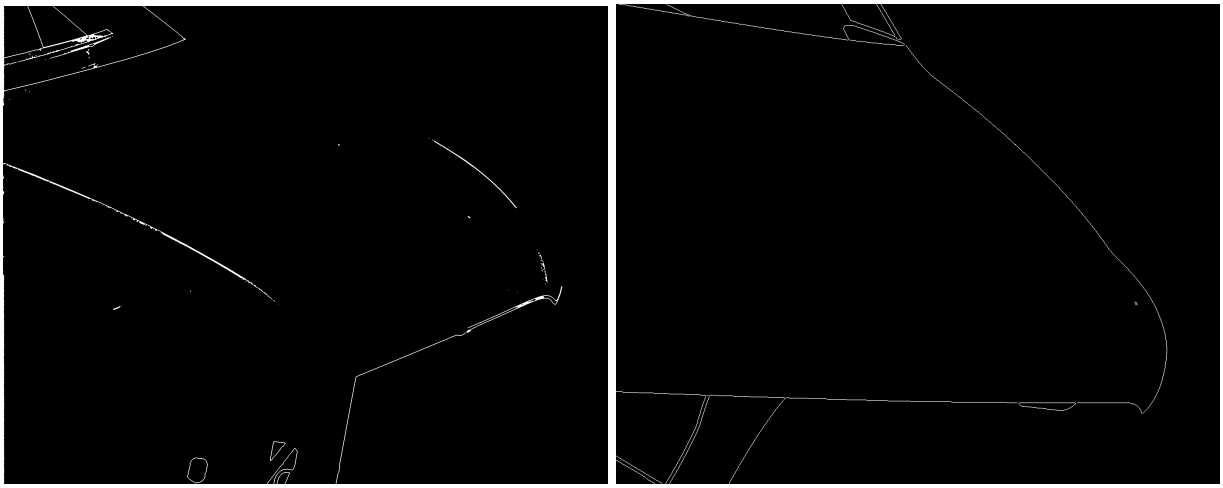
Camera 1. **Left**: Edges from real image. **Right**: Edges from Blender image.



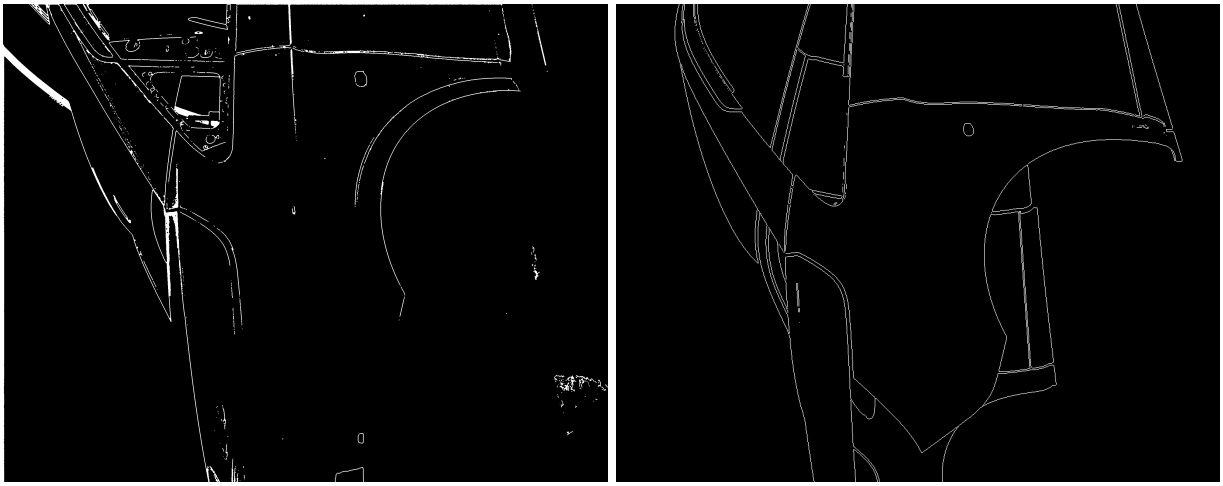
Camera 2. **Left**: Edges from real image. **Right**: Edges from Blender image.



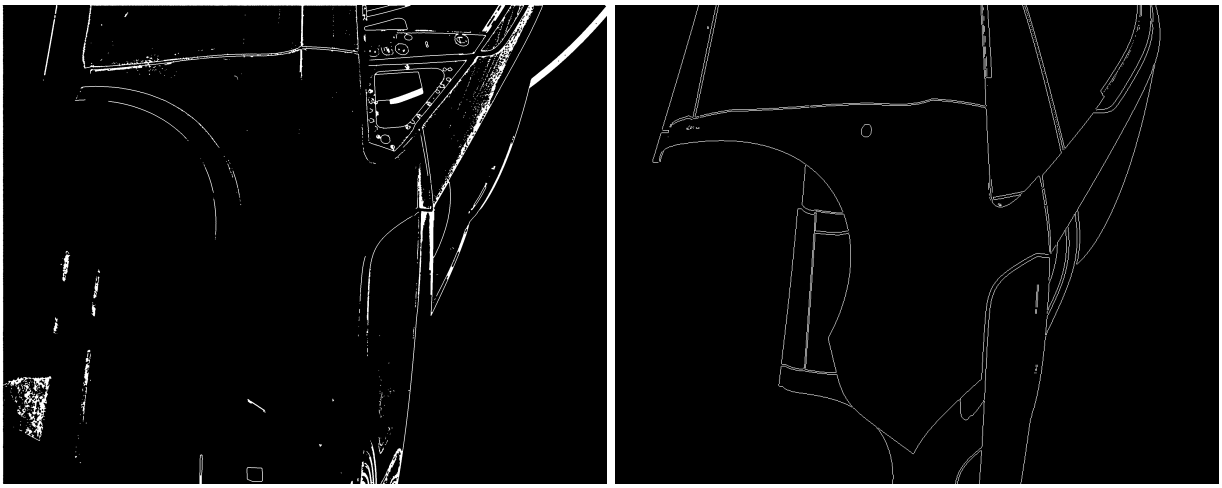
Camera 3. **Left**: Edges from real image. **Right**: Edges from Blender image.



Camera 4. **Left**: Edges from real image. **Right**: Edges from Blender image.



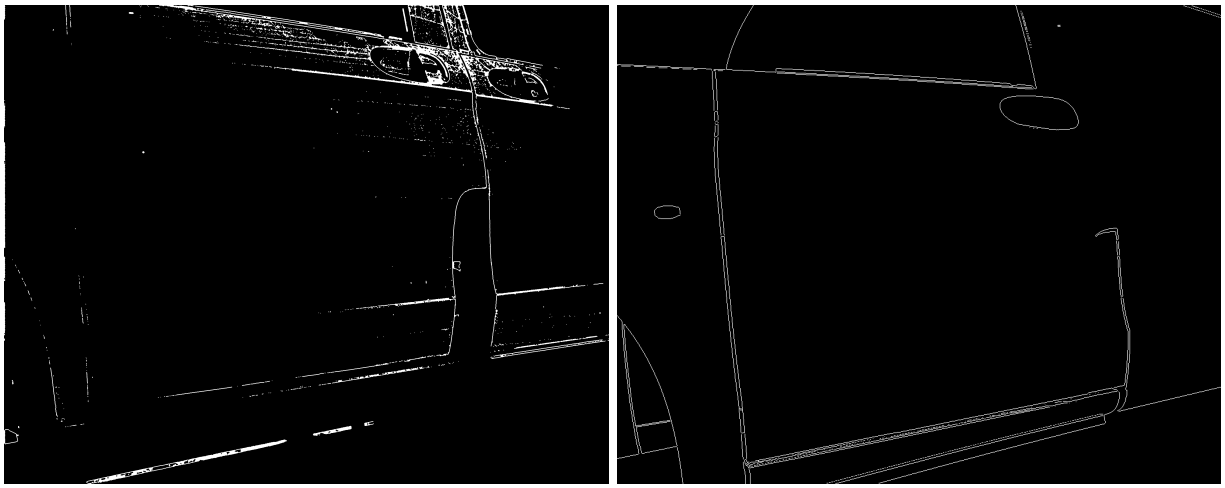
Camera 5. **Left**: Edges from real image. **Right**: Edges from Blender image.



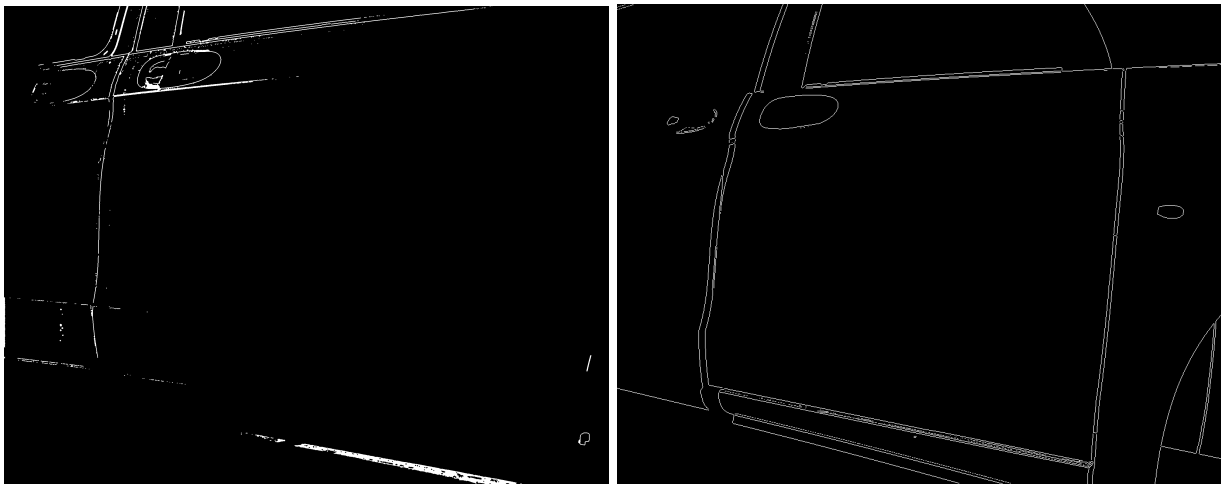
Camera 6. **Left**: Edges from real image. **Right**: Edges from Blender image.

Camera 7. **Left**: Edges from real image. **Right**: Edges from Blender image.

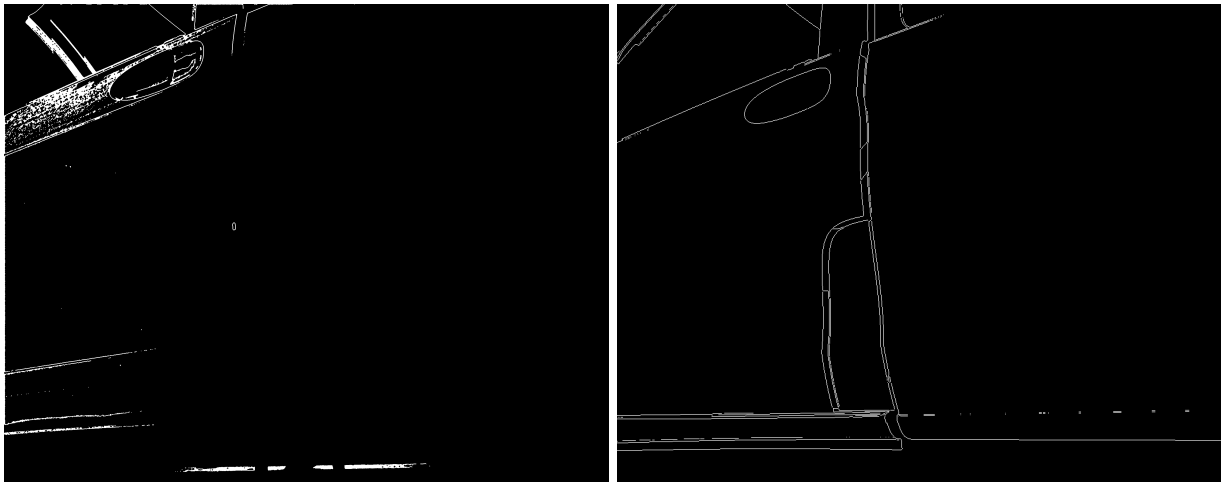
Camera 8. **Left**: Edges from real image. **Right**: Edges from Blender image.



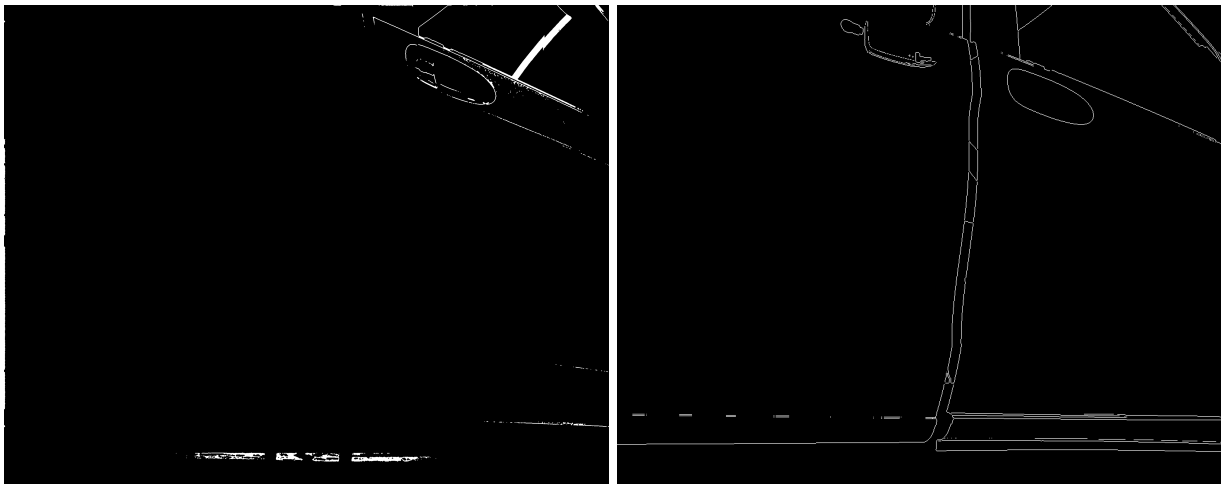
Camera 9. **Left**: Edges from real image. **Right**: Edges from Blender image.



Camera 10. **Left**: Edges from real image. **Right**: Edges from Blender image.



Camera 11. **Left**: Edges from real image. **Right**: Edges from Blender image.



Camera 12. **Left**: Edges from real image. **Right**: Edges from Blender image.

Chapter 6

Conclusion

6.1 Project Recommendation

The current system has been shown to work well using simulated data. However, the next step for the project is to evaluate the system using real data from the manufacturing plant. This requires finding the calibration parameters for the real cameras. Once we know the calibration parameters, we can replicate the real cameras in the Blender simulation so that we can find the true alignment of the real and simulated camera images.

Ground truth evaluation

It is likely that the ground truth vehicle pose will not be provided with the real camera data, so a metric for the quality of the optimisation needs to be found. One possible solution is, given a pose estimate from the optimisation, to transform and project the simulated points into each of the real camera images. We can then statistically or visually compare the edges to determine the error in the pose estimate. While this will not give us the pose error in real world units (metres and degrees), it can give us a relative measure of error between estimates which can indicate whether the algorithm is improving. The accuracy of this method will be dependent on whether we can extract matching edges in the simulated and real camera images.

Camera selection

Based off the experiments with simulated data and our current knowledge of the system, we recommend manually selecting a few cameras from the robust set identified in Sec. 4.1.1, based on the edge detection of the real camera images. These edges will differ from those extracted from the simulated data primarily due to the lighting conditions in the tunnel, therefore the camera selection must be made based on the real camera images. The choice can be made by replicating Experiment 1 with the real cameras.

Simulated edge selection

Once we have selected the cameras to use, we can identify the key edges that are detected in each image. We can then ensure that these edges are present in the simulated edge image. We should also identify the consistency of the edges of each real camera image for different pose offsets of the vehicle (i.e. if we have the same vehicle model with a different pose offset, do the same edges appear in both images). If we find the edges to be consistent, we may choose to only include the edges that we manually select. This edge selection can be done using Blender's Freestyle Toolbox.

Edge Detection Method

We should do further testing with the real camera data to find which edge detection method produces the most consistent and robust pose estimates. The edge detector choice will largely be dependent on finding the best balance between extracting more prominent edges and reducing image noise.

M-Estimator Choice

The type of M-estimators used should also be investigated more thoroughly. Currently, a combination of the Huber and bisquare estimators have produced the most accurate and robust pose estimates. However, different estimators should be tested on the real camera data. Student-t has been shown to outperform other estimators [40] and should be compared with the existing choices.

6.2 Conclusion

Finding an accurate, global estimate of a vehicle's pose is vital for an accurate projection of paint defects onto the vehicle model. This report has presented a system to estimate the vehicle pose by minimising the distance between the edges of a real camera image and the projected edges of a CAD model of the vehicle. The system has been evaluated on simulated data and has been shown to perform well past the required tolerance of the system. The key components of the system and certain methodology choices have been discussed and areas for improvement have been identified. Finally, a recommendation for the project moving forward has been made to identify the next steps for the project.

Bibliography

[1] L. Armesto, J. Tornero, A. Herraez, and J. Asensio, "Inspection system based on artificial vision for paint defects detection on cars bodies," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 1–4. DOI: 10.1109/ICRA.2011.5980570.

[2] D. Scaramuzza. (). Robust visual-inertial state estimation: From frame-based to event-based vision, [Online]. Available: https://www.dropbox.com/s/kbtjnb04n9r18ci/Scaramuzza.pdf?dl=0.

[3] J. Engel, T. Sch, and D. Cremers, "Lsd-slam: large-scale direct monocular slam," pp. 834–849, 2014, ISSN: 16113349. DOI: 10.1007/978-3-319-10605-2_54.

[4] P. Larsen. (). Direct and feature based methods in slam, [Online]. Available: https://www.doc.ic.ac.uk/~mly15/c2359z/computing_topics_website/directvsfeature.html.

[5] "Improving the agility of keyframe-based slam," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5303 LNCS, no. PART 2, pp. 802–815, 2008, ISSN: 03029743. DOI: 10.1007/978-3-540-88688-4-59.

[6] E. Eade and T. Drummond, "Edge landmarks in monocular slam," *Image and Vision Computing*, vol. 27, no. 5, pp. 588–596, 2009, ISSN: 02628856. DOI: 10.1016/j.imavis.2008.04.012.

[7] A. Concha and J. Civera, "Using superpixels in monocular slam," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 365–372, 2014, ISSN: 10504729. DOI: 10.1109/ICRA.2014.6906883.

[8] B. Fan, F. Wu, and Z. Hu, "Line matching leveraged by point correspondences," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2010, pp. 390–397. DOI: 10.1109/CVPR.2010.5540186.

[9] B. Fan, F. Wu, and Z. Hu, "Robust line matching through line-point invariants," *Pattern Recogn.*, vol. 45, no. 2, pp. 794–805, Feb. 2012, ISSN: 0031-3203. DOI: 10.1016/j.patcog.2011.08.004.

[10] M. Al-Shahri and A. Yilmaz, "Line matching in wide-baseline stereo: A top-down approach," *IEEE Transactions on Image Processing*, vol. 23, no. 9, pp. 4199–4210, Sep. 2014, ISSN: 1057-7149. DOI: 10.1109/TIP.2014.2331147.

[11] K. Li, J. Yao, M. Xia, and L. Li, "Joint point and line segment matching on wide-baseline stereo images," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Mar. 2016, pp. 1–9. DOI: 10.1109/WACV.2016.7477734.

[12] A. D. Bimbo. (2009). Invariance, [Online]. Available: http://www.micc.unifi.it/delbimbo/wp-content/uploads/2011/03/slide_corso/A31%20feature%20invariance.pdf.

[13] P. Pritchett and A. Zisserman, "Wide baseline stereo matching," *Imagine*, 1999.

[14] P. H. S. Torr and A. Zisserman, "Feature based methods for structure and motion estimation," in *Vision Algorithms: Theory and Practice: International Workshop on Vision Algorithms Corfu, Greece, September 21–22, 1999 Proceedings*, B. Triggs, A. Zisserman, and R. Szeliski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 278–294, ISBN: 978-3-540-44480-0. DOI: 10.1007/3-540-44480-7_19.

[15] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 15–22. DOI: 10.1109/ICRA.2014.6906584.

[16] C. Beall, *Stereo visual odometry*, http://frc.ri.cmu.edu/~kaess/vslam_cvpr14/media/VSLAM-Tutorial-CVPR14-A12-StereoVO.pdf, 2014.

[17] A. Concha and J. Civera, "RGBDTAM: A cost-effective and accurate RGB-D tracking and mapping system," *CoRR*, vol. abs/1703.00754, 2017.

[18] M. Irani and P. Anandan, "All about direct methods," in *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, ser. ICCV '99, London, UK, UK: Springer-Verlag, 2000, pp. 267–277, ISBN: 3-540-67973-1.

[19] C. Gava. (2011). Dense 3d reconstruction, [Online]. Available: https://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws11-12/3DCV_WS11-12_lec10.pdf.

[20] P. Mlsna and J. Rodríguez, "Gradient and laplacian edge detection," English (US), in *The Essential Guide to Image Processing*. Elsevier Inc., 2009, pp. 495–524, ISBN: 9780123744579. DOI: 10.1016/B978-0-12-374457-9.00019-6.

[21] Dharampal and V. Mutneja, "Methods of image edge detection: A review," *Journal of Electrical & Electronic Systems*, vol. 4, no. 2, pp. 1–5, 2015, ISSN: 2332-0796. DOI: 10.4172/2332-0796.1000150. [Online]. Available: https://www.omicsonline.org/open-access/methods-of-image-edge-detection-a-review-2332-0796-1000150.php?aid=57249.

[22] P. A. Mlsna and J. J. Rodríguez, "Chapter 19 - gradient and laplacian edge detection," in *The Essential Guide to Image Processing*, A. Bovik, Ed., Boston: Academic Press, 2009, pp. 495–524, ISBN: 978-0-12-374457-9. DOI: https://doi.org/10.1016/B978-0-12-374457-9.00019-6. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780123744579000196.

[23] F. Bergholm, "Edge focusing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 6, pp. 726–741, Nov. 1987, ISSN: 0162-8828. DOI: 10.1109/TPAMI.1987.4767980.

[24] A. Shweta, "A review paper of edge detection using ant colony optimization technqiues," vol. 1, 2012, pp. 120–123.

[25] M. Heath, S. Sarkar, T. Sanocki, and K. Bowyer, "Comparison of edge detectors: A methodology and initial study," in *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 1996, pp. 143–148. DOI: 10.1109/CVPR.1996.517066.

[26] K. Bowyer, C. Kranenburg, and S. Dougherty, "Edge detector evaluation using empirical roc curves," in *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, vol. 1, 1999, 359 Vol. 1. DOI: 10.1109/CVPR.1999.786963.

[27] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 2, pp. 239–256, Feb. 1992, ISSN: 0162-8828. DOI: 10.1109/34.121791. [Online]. Available: http://dx.doi.org/10.1109/34.121791.

[28] T. Yang, *Summary of a method for registration of 3-d shapes*, Presentation Slides.

[29] M. Alshawa, "Icl: Iterative closest line a novel point cloud registration algorithm based on linear features," vol. 10, Jul. 2007.

[30] A. Censi, "An icp variant using a point-to-line metric," in *2008 IEEE International Conference on Robotics and Automation*, May 2008, pp. 19–25. DOI: 10.1109/ROBOT.2008.4543181.

[31] B. O. Community, *Freestyle*, Blender Foundation. [Online]. Available: https://docs.blender.org/manual/en/dev/render/freestyle/index.html.

[32] ——, *Blender - a 3d modelling and rendering package*, Blender Institute, Amsterdam: Blender Foundation. [Online]. Available: http://www.blender.org.

[33] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, pp. 509–517, 9 Sep. 1975, ISSN: 0001-0782. DOI: 10.1145/361002.361007. [Online]. Available: http://doi.acm.org/10.1145/361002.361007.

[34] P. Huber, J. Wiley, and W. InterScience, *Robust statistics*. Wiley New York, 1981.

[35] Z. Zhang, "Parameter estimation techniques: A tutorial with application to conic fitting," *Image Vision Comput.*, vol. 15, pp. 59–76, 1997.

[36] J. Fox and S. Weisberg, *Robust regression**, 2013. [Online]. Available: `http://users.stat.umn.edu/~sandy/courses/8053/handouts/robust.pdf`.

[37] T. Binder and E. Kostina, *Gauss–newton methods for robust parameter estimation*, Aug. 2013.

[38] W. J. Rey, *Introduction to Robust and Quasi-Robust Statistical Methods*. Springer, 1983.

[39] Z. Zhang, "Parameter estimation techniques: A tutorial with application to conic fitting.," *Image Vision Comput.*, vol. 15, pp. 59–76, Mar. 23, 2004. [Online]. Available: `http://dblp.uni-trier.de/db/journals/ivc/ivc15.html#Zhang97`.

[40] G. Agamennoni, P. Furgale, and R. Siegwart, "Self-tuning m-estimators," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 4628–4635.

[41] D. Chetverikov, D. Svirko, D. Stepanov, and P. Krsek, "The trimmed iterative closest point algorithm," in *Object recognition supported by user interaction for service robots*, vol. 3, 2002, 545–548 vol.3.

[42] P. Huber and E. Ronchetti, *Robust statistics - Second Edition*. John Wiley & Sons, 2009.

[43] S. Katiyar and A. P V, "Comparative analysis of common edge detection techniques in context of object extraction," vol. 50, pp. 68–78, Nov. 2012.