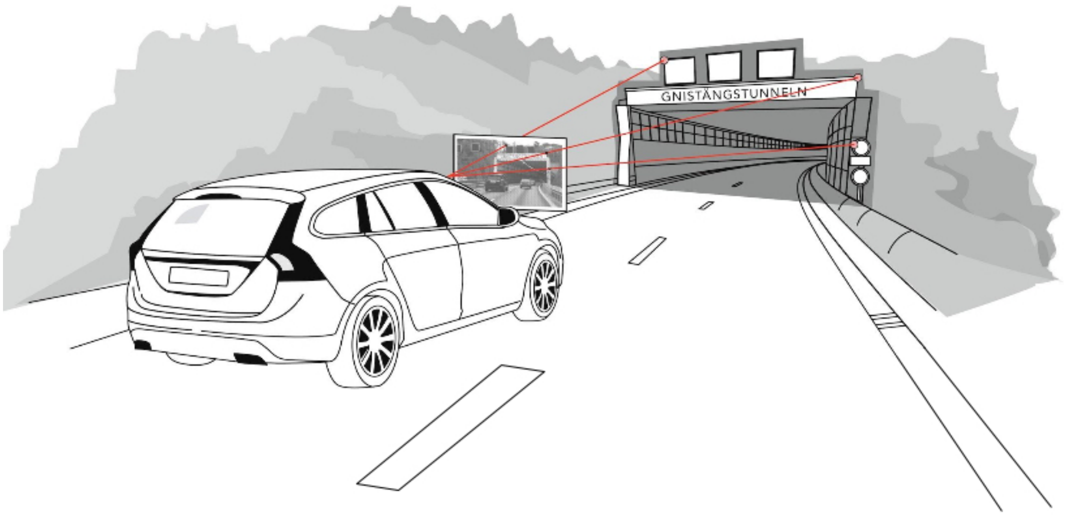


Global Positioning inside Tunnels

Using Camera Pose Estimation and Point Clouds

Master's thesis in Computer Systems and Networks

David Bennehag
Yanuar T. Aditya Nugraha

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016:XX

Global Positioning inside Tunnels

Using Camera Pose Estimation and Point Clouds

DAVID BENNEHAG
YANUAR TRI ADITYA NUGRAHA

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2016

Global Positioning inside Tunnnels Using Camera Pose Estimation and Point Clouds
DAVID BENNEHAG
YANUAR TRI ADITYA NUGRAHA

Supervisor: Fredrik Kahl, Department of Signals and Systems
Examiner: Elad Schiller, Department of Computer Science and Engineering

Master's Thesis 2016:XX
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration of an entrance to the Gnistängtunneln in Gothenburg as seen from the camera image with keypoints are being shown.

Illustrated by: Nesia Anindita (nesianindita@gmail.com)

Typeset in LaTeX
Printed by PRISMA Chalmers
Gothenburg, Sweden 2016

Abstract

This master thesis aims to solve the positioning problem in the absence of an accurate GPS solution. We will show how high density tunnel model in 3D point clouds, together with only a single vehicle-mounted camera, can estimate the vehicle's position throughout tunnels without the assistance of other positioning systems. We developed a solution by analysing an image sequence where feature detection and image to point cloud backprojection were used to generate an accurate positioning system. The feature detection was performed using the well-known SIFT algorithm and back-projection onto the point cloud is done through repeated closest-neighbour search along a ray we build from the feature's coordinates. Multiple post-processed GPS runs within the tunnel are used to generate a ground truth which is used as the reference for evaluating the solution. For this end, we developed a new software architecture where the software could ultimately be placed on top of any existing positioning method and then provide an easy comparison between the two. An experimental result is provided to show that accurate positioning can be achieved under the right circumstances, reaching a drift distance of below ten centimeters.

Keywords: camera positioning, visual localization, autonomous vehicles, offline positioning, point clouds

Acknowledgements

The authors would like to thank Volvo Cars, especially our supervisors Jonathan Ahlstedt-Gyllbrandt and Nasser Mohammadiha for their support. Also, Fredrik Kahl for sharing his expertise within the area of computer vision.

David Bennehag
Yanuar Tri Aditya Nugraha

Gothenburg, June 2016

Contents

List of Figures

List of Tables

1
Introduction

Human drivers are dangerous. The human factor is today a major cause of traffic related accidents, where three out of five road traffic crashes were caused by driver-related behavioural factors [1]. It will likely remain as such until the roads are inhabited primarily by robotic drivers, or autonomous vehicles. The road to getting there is still long, but a not insignificant stepping stone towards that end is to verify and assert that the autonomous driving works as expected. One nontrivial part of this is making sure that the vehicle's information of where it is currently positioned is accurate down to the most extreme levels, even when moving into confined areas such as tunnels.

Computer vision can be used for a multitude of purposes, such as scene reconstruction, video tracking and object pose estimation. It revolves around acquiring and analysing images for the purpose of generating some kind of information needed, for example when making other decisions. For the safety systems of an autonomous vehicle, it can give information such as where pedestrians are walking, or what road signs are being passed. For the purpose of this project, what will instead be the focus is the properties of the general scene around the vehicle. The surroundings will be scanned for unique features, with which we can uniquely identify very specific points in the world, which can then be passed to algorithms with the purpose of accurately positioning the vehicle in global coordinates. We will be comparing the images from a vehicle-mounted camera with a high density point cloud to find these unique features and find global coordinates for the vehicle as it moves through a tunnel.

1.1 Background of the Problem

One of the main reasons for deploying autonomous vehicles today is attributed to the fact that a majority of traffic related accidents happens because of human error [2]. The advent of such technology is predicted to have a paramount effect in reducing the number of casualties in traffic situations. One of the key problems when designing the active safety systems in a self-driving car is how to accurately position the vehicle based on its surroundings.

Global Navigation Satellite Systems (GNSS) are often used for vehicle positioning. However, whenever there is not a clear line of sight to the sky, the accuracy of these systems will decrease. The signals coming from the satellite are obstructed by foliage, walls and structures [3]. For this reason, whenever the vehicle travels into a tunnel, we need to use other sensors to position the vehicle. A popular positioning method involves building a map of the vehicle's surroundings based on the camera images while driving. Another method, and the method we chose to use, is by analysing the data gathered from a camera, by extracting the required information and then comparing this to a high-accuracy reference, in our case a high-density point cloud.

1.2 Statement of the Problem

The current gap in the knowledge is how to utilise a very high density point cloud in order to achieve very accurate positioning in areas such as tunnels, where there is no possibility of using ordinary positioning systems like the GPS.

The problem addressed in this report is how to use the vehicle-mounted camera instead, to find a robust position which can be used for the verification. Other solutions build point clouds from the images gathered while driving, but these are significantly more sparse and should thus also lead to more inaccurate positioning results. We hope to use the already constructed high density point cloud in order to improve the results.

There are a number of more or less obvious challenges when working with computer vision inside tunnels. The most obvious challenge is the lack of good lighting conditions resulting in dark images, possibly so dark that no unique features can be found. It is of course possible to increase the brightness, but this may instead lead to the inverse problem in areas that are already lit up; they may become too bright and lose the unique features that would have otherwise been found. Another challenge that may not actually be realised until one travels through the tunnels, is the repetitive physical patterns. Since we need a distinct matching between two subsequent images, this repetetiveness leads to problems when one tries to match features between them. It is hard to match the feature you find with the correct match in another image if you have tens, hundreds, or even thousands, of them looking very similar.

1.3 Thesis Contribution

We propose, implement and evaluate via experimentation, an approach for validating a vehicle's position inside the tunnel using a camera-based method that can serve as an alternative to the GPS when it is unavailable. We combined a widely used camera-based feature matching with an accurate 3D model of the tunnel to provide a positioning system comparable to its GPS counterpart. Through that, we are addressing the absence of a ground truth by deriving it ourselves. Using two different methods, statistically and empirically, we obtain a greater degree of robustness for the ground truth, as described in Section 1.5. Another contribution is related to the algorithms used. While many of the algorithms are provided by the third-party libraries, our back-projection algorithm, shown in Section 5.7, is a novel approach and specifically designed to fit in our solution. For this we have implemented a search algorithm for finding specific points in the cloud using a nearest-neighbour method.

We also adapted a feature matching algorithm (Section 3.4) to fulfill our requirements of handling SIFT features inside the tunnels by applying region based feature tracking, as shown in Section 5.3, in addition to the standard matching procedure as proposed and demonstrated in the original published SIFT paper [20]. The remaining implemented algorithms are explained in Section 5.

Lastly, we also propose a software architecture that we believe to be suitable for future works using similar validation techniques, as shown in Section 4.1. Since our solution is independent to the input, different scenarios could be simulated as long as the correct inputs are available. This has been an important benefit of using a lookup table as implemented in Section 5.1.

1.4 Project Setting

This master thesis is carried out within Volvo Cars to solve the positioning problem inside tunnels, using one monocular camera. Given the camera's intrinsic and calibration parameters, along with the camera images, we will find the camera position in global coordinates. A point cloud of the tunnel augmented with global coordinates of the *SWEREF 99* system will be used to estimate the global coordinates of the vehicle-mounted camera within the tunnel. The thesis is done together with Volvo Cars as a part of the *Drive Me* project [4], in which 100 self-driving cars will use public roads around Göteborg, including the four main tunnels; Göta-, Gnistängs, Tingstads- and Lundbytunneln.

1.5 Verification and Evaluation

As there is no way of verifying the correctness to 100% certainty, we will evaluate the method's performance by comparing the resulting position with the other positioning methods already available from the car. The simplest method available is the dead-reckoning that is performed when the regular GPS can no longer position the vehicle, for example inside a tunnel. The less sensors we use for our method, such as the IMU or radar without GPS assistance, could be used to independently verify the solution. Other ways of verifying the results could be to manually compare the resulting position to what you experience when driving the car, however this is not very precise. The results could also be compared to those of the regular GPS when driving outdoors. A good environment very comparable to a tunnel is an area of tall buildings or objects where the camera still detects walls on both sides and the GPS still receives signals from satellites.

Evaluation consists of determining the accuracy of the positioning results, and under what conditions we can achieve this. One part of this thesis contribution is to build a measurable, usable and practical reference which can be used as the reference for our solution. We compare our positioning method to the multiple post-processed GPS runs where the virtual ground truth is based on the mean value of the trajectories. We also provide an estimation of the vehicle trajectory using two steps. First is by manually selecting carefully a point in every lane marking in the middle of the road and that gave us a ground truth for the middle lane. The second step is to estimate the vehicle true trajectory by visually marking vehicle's whereabout from the middle lane and estimate its trajectory relative to the middle lane. This method will be used to visually verify the camera trajectory within the road lane, therefore metric comparison is not available for this and instead, the statistical approach ground truth from the post-processed GPS runs is used for metric (lateral and altitude) comparison.

1.6 Related Work

Much research has already been done in the area of using image processing for the sake of positioning a robot, or vehicle, in an environment. However, a majority of this research was performed in outdoor, or indoor, environments where the environment itself provides a rich amount of unique landmarks. Our work aims at applying this research in an environment with an almost extreme lack of such unique features, namely inside road tunnels. Similar work related to ours has been done, where camera images are analysed to detect, for example, lane markings. However, this was performed just in order for the vehicle to stay in lane, i.e. departure warning [5]. Metric positioning, where you try to find the vehicle's position within a certain distance of the true position, has not been investigated much in the literature for environments as challenging as tunnels with low light conditions and no GPS. This is what constitutes the advancement in the state-of-the-art techniques. In [6] it is shown how a single monocular camera combined with a prebuilt point cloud can be used for estimating the camera pose. Their settings, for example for RANSAC, gave us a good starting point in our own solution. The difference is again the environment in which it is used; the authors don't run their solution in a tunnel. Their 3D map is also built using another technique, through an RGB-D sensors, and they are using another technique involving virtual images created from the point cloud to run their algorithm. Another notable work on using monocular camera to track vehicle position relative to the point clouds is visual odometry [7]. Visual odometry is widely used in the field of robotics, UAV and automotive. Visual odometry operates by incrementally estimating the pose of the vehicle through examination of changes between overlapping scenes between frames. Scaramuzza et al [8], explained that there are three ways on how visual odometry could be implemented, 2D-to-2D, 3D-to-3D and 3D-to-2D. This thesis project is heavily influenced with the latter visual odometry, as the point cloud is used to represent the 3D world environment.

We have not managed to find any comparable off-the-shelves products exist for this project. This is likely due to the fact that any similar projects would be performed only by other car companies, are proprietary and not available for scrutiny.

1.7 Limitations

This thesis is limited by using prebuilt 3D model of the tunnel in a very dense point cloud. It is courtesy of Volvo Cars, along with the post-processed runs within tunnel for verification and comparison. Therefore, we do not need to perform SLAM or some other technique for building the maps ourselves.

For a more complete and robust product, one would likely have to implement a Kalman filter to assist in the visual odometry. This will however not be included in this project as we are focusing on the image analysis aspect of the solution.

Our work aims only at the Gnistängstunnel in Gothenburg. This means that the resulting method could be more specialised at solving the problem only in this well-known environment, even though a more general solution would be preferred. For the solution to function, we need to first build manual correspondences between the images and the physical world. We also limit ourselves to an offline solution, as the results are to be used for verification purposes, rather than real-time positioning.

1.8 Structure of the report

The remainder of the report is divided into the following chapters; Camera Geometry, Theory, Method & Architecture, Results, Discussion and finally Conclusions. In Theory, all technical terms will be defined, the primary problems will be explained and the required theory will be discussed. In Method & Architecture we will be exploring the general approach to solving the given problem, define our work flow, as well as reason about the third party software used. After giving the general approach, we will in Implementation dive into our solution by exploring the software architecture and design with all the important algorithms explained. Finally, in Results and Discussion, the accuracy of our solution an d its important properties are illustrated and explained.

2

Camera Geometry

As the camera plays the leading role in our project, this chapter will be discussing the camera model we are considering, the properties that must be known and general theory about the most commonly used camera model, the pinhole model. Also finding the camera position relative to the world is introduced and at the end of this chapter, an approach on solving the camera pose is explained brief and in a concise manner.

2.1 The Pinhole Camera Model

The camera model we are considering in our work is the pinhole camera model, as depicted in Figure 2.1. It is a relatively simple model where light passes through the camera aperture, here called the pinhole, before hitting the image plane, our camera sensor. It is assumed that the pinhole, or the optical centre, is infinitely small, represented by a point.

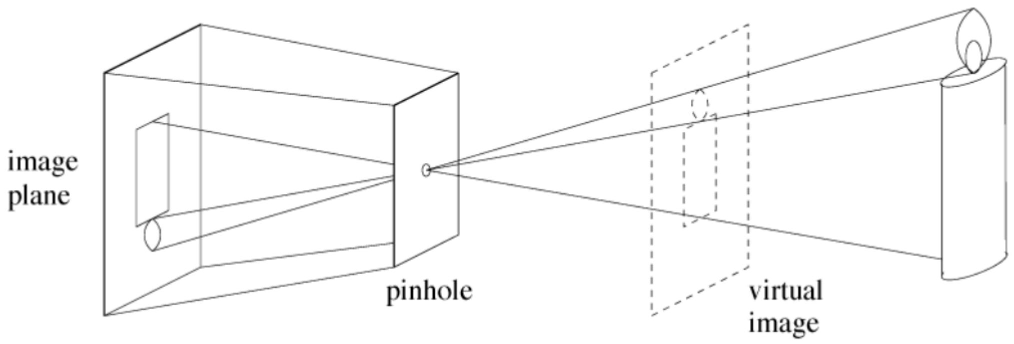


Figure 2.1: The pinhole camera model, from [9].

There are a number of basic properties to this camera model:

1. The further away the object is in the world space, the smaller the object representation is on the image plane.
2. We lose both length and angles of lines, but straight lines will still be straight.
3. Parallel lines will intersect in the image at a so called vanishing point.

When representing the camera model like this, the image appears on the image plane as upside down, as can be seen in Figure 2.1. To overcome this nuisance and the problems that stem from it, such as having negative values in our calculations, we instead place the image plane in front of the camera, as can be seen in Figure 2.2.

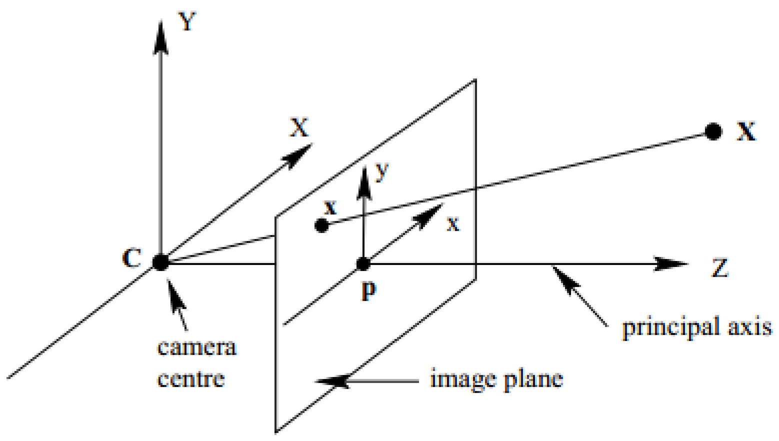


Figure 2.2: The modified pinhole camera model, as described by Hartley & Zisserman in [10].

C in the above figure is the camera center, or optical centre which, in our previous figure, lay in front of the image plane. The center of the image plane is p, the principal point. The camera center C represents the centre of projection, and is placed at the origin of the Euclidian coordinate system. The straight line in Figure 2.2 above, which starts from the camera centre and passes through the principal point and onwards, is known as the principal axis, or ray.

In Figure 2.3 it is shown how all this is related to the physical properties of a camera.

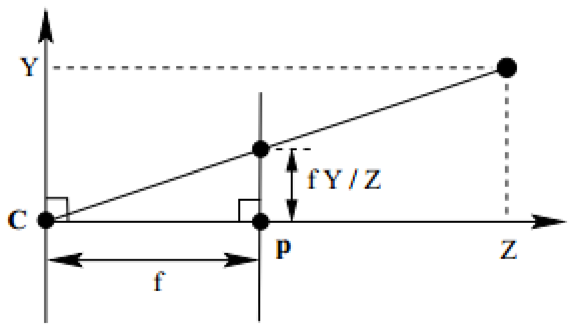


Figure 2.3: The modified pinhole camera model, as described in [10].

The distance between the camera center C and the principal point p is known as the focal length, an important aspect in the coming algorithms, as it is part of the transformation between image (2D) and world coordinates (3D) frames.

2.2 Camera Parameters

When talking about camera parameters, we make the distinction between internal, hereafter intrinsic, and external, hereafter extrinsic, parameters.

2.2.1 Extrinsic

In theory, the extrinsic camera parameters defines the position and rotation of the camera, relative to a known world reference. In practice, finding the extrinsic parameters consists of finding:

- The rotation matrix R, describing how to align the axes of the world coordinate frame and the image coordinate frame.
- The translation vector t, describing the relative positions of the two coordinate frames.

Having both R and t, it is possible to combine the two, in order to create the transformation matrix T:

$$\left[\begin{array}{c|c} R & t \\ \hline 0 & 1 \end{array}\right]$$

Combining R and t gives us a 3x4 matrix. Thus, to simplify the matrix operations we need to perform in our algorithms we add a fourth row, as can be seen above. This gives us a 4x4 matrix instead, without affecting it's correctness. With this transformation matrix, we can transform points in world coordinates to points in camera coordinates. These different coordinate spaces will be explained in further detail in Section 2.3. For this project we are more interested in converting in the opposite direction, namely from camera to world coordinates, as will be shown in Chapter 5.

2.2.2 Intrinsic

The intrinsic parameters define, in contrast, the internal properties of the camera, and the given matrix is often called the *camera matrix.* These properties are specific for each camera unit and includes both optical and geometric properties of the camera. They are represented by the camera matrix K:

$$\begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where f defines the optical length, c defines the principal point and γ defines the skew coefficient between x and y axis. Both of f and c can have separate values for the X and Y axes. The camera matrix K is very useful for converting between points in the camera coordinate system and the image coordinate system. The information given by the camera matrix does not depend on the scene around the camera, so we only need to estimate it once and it can then be re-used as long as we have a fixed focal length. Camera calibration is a step which is needed to find all the intrinsic values. Once all the values are known, the camera system is considered *well-calibrated.* An internal property of the camera that is not described by the intrinsic matrix, but is instead represented by its own vector, is the camera distortion coeffcients. Typically there is some degree of optical distortion from a camera's lens, resulting in the warping of the images. A very clear example of this, where the distortion is on purpose, is a fisheye lens.

2.3 Coordinate Spaces and Transformations

When using cameras for the computer vision algorithms, there are three coordinate spaces, or systems, involved. There are the two obvious ones, image and world spaces, but we must also consider the camera coordinate system. The image and camera coordinate systems are easily confused, but in order for our transformation algorithms to work, we need to clearly define them.

Starting with the camera coordinate system, there may be some confusion why this is needed when we already have the world coordinate system. If we align the camera in such a way that it is looking down the Z axis and stays still at the origin of the world coordinate system, as depicted in Figure 2.4, then indeed we do not require another coordinate system, we can just use the one already defined.

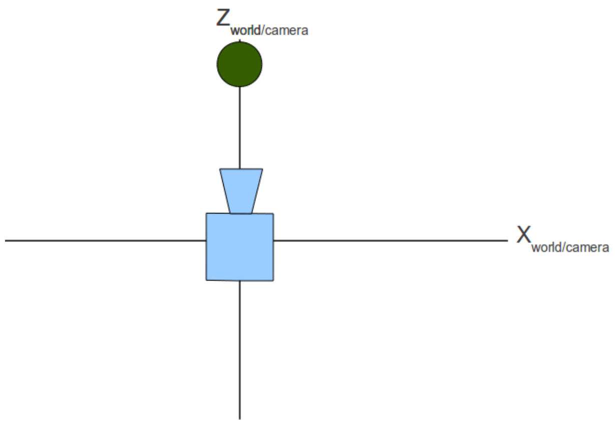


Figure 2.4: The camera coordinate system when the camera is aligned with the world coordinate system [11].

However, this is rarely the case. Most of the time, we want to be able to move and rotate our camera. To do this, another coordinate system for the camera is defined, as is shown below in Figure 2.5

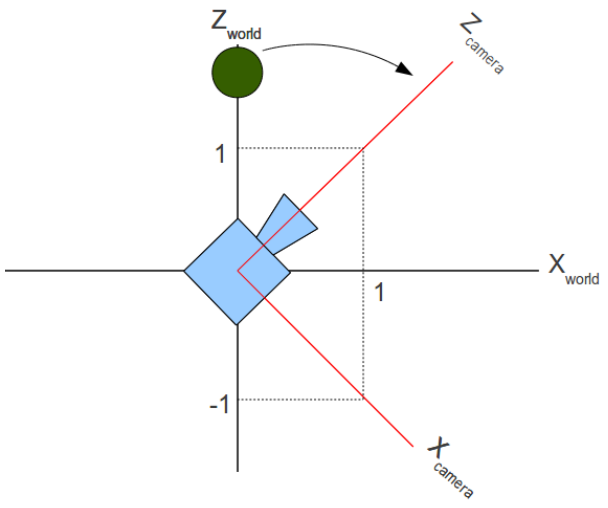


Figure 2.5: The camera coordinate system when the camera is rotated [11].

The origin for the camera coordinate system is at the center of the sensor inside the camera, and the focal length describes the distance to the image plane, as described previously in Figure 2.2.

When talking about the camera coordinate system, we need to introduce the perspective projection, which describes how to transform 3D camera coordinates to 2D camera coordinates. This mapping between 3D and 2D is described by the camera projection matrix, P, which includes the camera's focal length. We represent our 2D point as a homogeneous coordinate by adding a third coordinate. This is called having homogeneous coordinates and the matrix representation of this can be seen below.

$$\begin{bmatrix} x_{image} \\ y_{image} \\ z_{image} \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_{camera} \\ Y_{camera} \\ Z_{camera} \\ 1 \end{bmatrix} \tag{2.1}$$

If we instead represent this through regular equations, we can find the non-homogeneous 2D coordinates, hereafter denoted u and v for easier separation, by utilizing the focal length as shown in Figure 2.3 and the following perspective projection equations:

$$u = \frac{x_{image}}{z_{image}} = f\frac{X_{camera}}{Z_{camera}} \tag{2.2}$$

$$v = \frac{y_{image}}{z_{image}} = f\frac{Y_{camera}}{Z_{camera}} \tag{2.3}$$

If we want to move our point's position, currently located in the camera coordinate system, into the world coordinate system, we need to use the transformation matrix T, defined in Section 2.2.1. When doing this, we are performing a so called Euclidian transformation, which is defined as follows:

$$p_{camera} = R \cdot p_{world} + t \tag{2.4}$$

or directly with the transformation matrix, which gives us homogeneous coordinates p'_{camera}:

$$p'_{camera} = T \cdot \begin{bmatrix} p_{world} \\ 1 \end{bmatrix} \tag{2.5}$$

Notice that the we have to add one more coordinate into the p_{world} to make it homogeneous and conform with the T.

Basically what happens through these equations is, we use our rotation matrix R to align the axes of the camera and the world coordinate systems, and then use our translation vector t to align the origins of both coordinate systems. This effectively creates the same situation as in Figure 2.4.

We have shown how to handle and translate between the world and camera coordinate systems, but since we are dealing with images from a camera, we also need a way of translating between the pixel coordinates and the camera coordinates.

To do this, we utilize the intrinsic parameters of the camera through the camera matrix K defined in Section 2.2.2. Again we add a coordinate to make our coordinates homogeneous, and then use the following equation for the translation:

$$\begin{bmatrix} x_{image} \\ y_{image} \\ z_{image} \end{bmatrix} = \begin{bmatrix} f_x & \gamma & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_{camera} \\ Y_{camera} \\ Z_{camera} \\ 1 \end{bmatrix} \tag{2.6}$$

An extra column is concatenated with the intrinsic matrix to make the computation conform to the homogeneous p'_{camera}.

Now that we have all the puzzle pieces, it is trivial to put the puzzle together and translate between image pixel coordinates and world coordinates through one concatenated matrix equation. Remember, K is the intrinsic parameters, and T is the extrinsic parameters of the camera.

$$\begin{bmatrix} x_{image} \\ y_{image} \\ z_{image} \end{bmatrix} = K \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} T \begin{bmatrix} X_{world} \\ Y_{world} \\ Z_{world} \\ 1 \end{bmatrix} \tag{2.7}$$

Thus we have found a 3x4 camera projection matrix P' for projecting world points into the image plane [37]. Note that it is not the same as the camera projection matrix P which was described earlier in Equation 2.1.

$$P' = K[R|t] \tag{2.8}$$

2.4 Camera Pose Problem

The camera pose problem in our case revolves around describing the camera motion around a static scene, the scene being the world around the camera, through a mathematical model that can be used in the algorithms. If we are given a number of correspondences between points in a 3D space and points in 2D space, the pose estimation problem in every step of our sequence consists of estimating the rotation and position of the camera in the current image [12].

In order to find how the world is rotated around the camera, or basically how the camera is pointing, we need to solve the camera pose problem. This problem is known as *Perspective-n-Point* problem which originated from camera calibration [13, 14]. This is an already solved problem and in OpenCV we can find the *SolvePnP* algorithm. There is also *SolvePnPRansac*, which uses RANSAC [14] in order to make the function resistant to outliers, which is why we choose to use this one instead. What it needs as input is a number of object points and their corresponding image points. For the first frame, these are the points we manually select and feed to the function. It needs the camera matrix, describing the camera's focal length and principal point, as well as the camera's distortion coefficients. What we get as output is the rotation matrix R and translation vector t. Together they form the extrinsic matrix which can be used to bring points from the world coordinate system to the camera coordinate system. With the following Equation 2.9 we can find the position of the camera in world coordinates.

$$C = R^t \cdot (K^{-1} * (u, v, 1)) - t) \tag{2.9}$$

Where C is the camera center point's position in world, or 3D, coordinates. K is the camera matrix and $(u, v, 1)$ is a vector of homogeneous image coordinates for the camera center, assumed to be $(0, 0, 1)$. Important to note is that the image, or 2D, coordinates vector is represented by three elements and not two as in Euclidean space.

One particular solution and one of the classical methods for solving this problem is *Perspective Three Problem* (P3P). P3P uses the smallest possible subset of points that generates a finite number of solutions. Assuming that the camera parameters are known and we have at least four points, the solution is generally unique [15]. The point C world coordinate in Equation 2.9 can easily be imagined by looking at Figure 2.6 below. By using the P3P algorithm for solving the camera pose problem, the problem becomes the following:

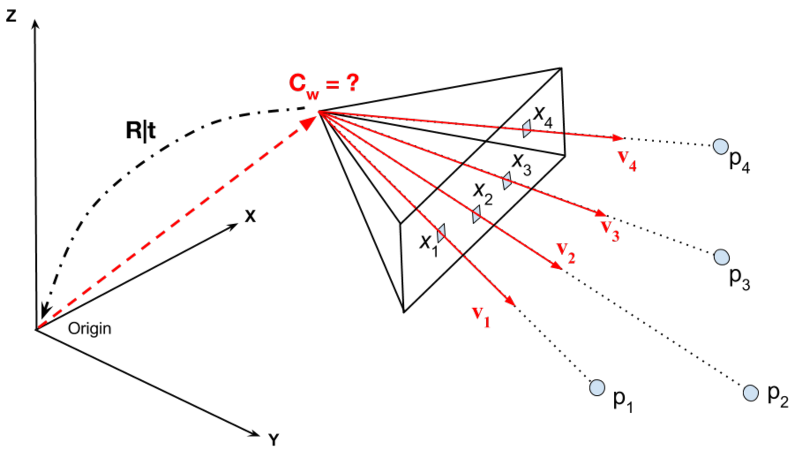


Figure 2.6: Illustration of the P3P problem

Like before, C_w represents the camera position in world coordinate and p_n are 3D landmarks located also in the world coordinate system with x_n are the corresponding 2D projections in the image plane. $R|t$ describes the rotation and translation that has been performed to move the camera to its current position from the origin. The Figure 2.6 gives us four 3D-2D correspondences, $(C_w \Rightarrow p_1, C_w \Rightarrow p_2, C_w \Rightarrow p_3, C_w \Rightarrow p_4)$. Only three are used to determine the distance estimation $|C_w p_1|$, $|C_w p_2|$ and $|C_w p_3|$ with the fourth point, p_4 is used to select the best pose estimation parameters[14]. When the number of 3D-2D correspondences is greater than four, using RANSAC to select the best set is done by OpenCV's *SolvePnPRansac*.

3
Theory

Estimating the vehicle position is done by finding the camera pose relevant to the world. Point clouds represent the surrounding environment and is considered to be the world reference. Accuracy of the project depends greatly on the quality of the point cloud as it is the physical representation of the scene surrounding the camera. The approach of implementing point clouds to be used as the reference for vehicle positioning requires several steps. This chapter will focus on explaining the theory required regarding point clouds, necessary image preprocessing, how features are deteced and keypoints are extracted from the video feed, selecting good keypoints to be projected onto the point cloud, RANSAC, and bundle adjustment to minimize the reprojection error and refine the camera pose estimation.

3.1 Point Clouds

This project used dense and accurate 3D model in a real-world scale as a dense point cloud. It can be generated using Lidar and multiple laser scanner. Lidar is an active-resolving optical remote measurement technique which allow the system to measure every property of the atmosphere [16]. For Lidar systems, they are expensive and are therefore currently not suitable for mass production vehicles [17]. Thus, its accuracy is still used for, e.g., offline positioning as we proposed using camera pose estimation within point point cloud.

The point clouds contain more than 30 millions of points in 3D coordinates relative to SWEREF99 metric standard and using meter for the distance unit. It is the closest thing to the ground truth and used as reference for the metric positioning in this project.

3.2 Image Preprocessing

Preprocessing is performed to achieve better accuracy from the features we detect. When we perform feature detection, as described in Section 3.3, some of the features are located in parts of the image that does not have a good representation in the point cloud, or are not stable enough for future feature matching. The process of ignoring these unwanted areas is called creating a Region of Interest (ROI) for our feature detection. In Figure 3.2 it is shown how this process can be done.

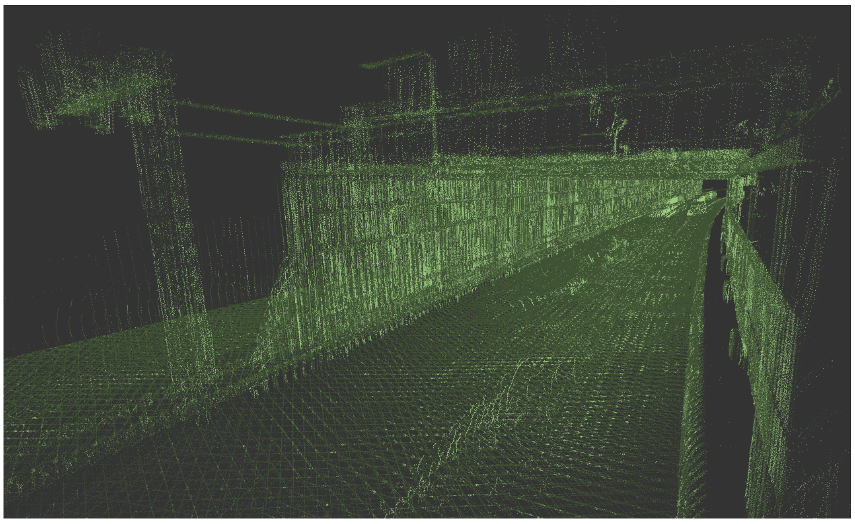
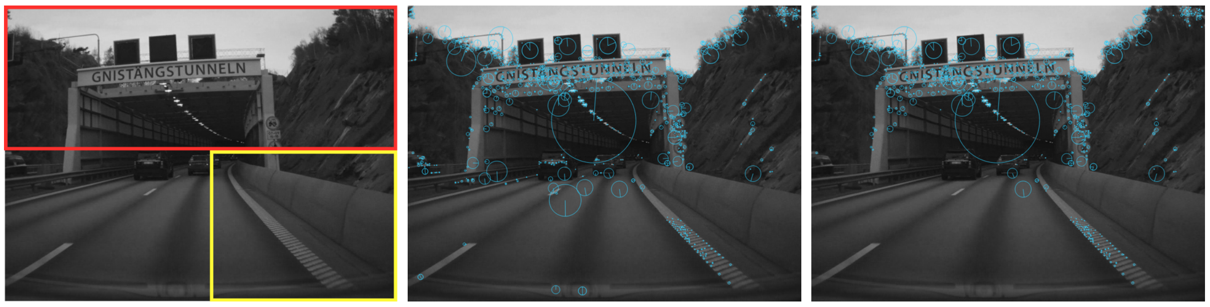


Figure 3.1: The point cloud of the tunnel entrance



(a) **(b)** **(c)**

Figure 3.2: Our implementation of the ROI to remove the unwanted features. (a) The red and the yellow boxes cover the areas in which we want to search. (b) Before the ROI implementation, features are found in the unwanted areas. (c) After the ROI implementation

As can be seen in Figure 3.2b, the features we find when not implementing an ROI are spread all over the frame. Many of these are in undesirable areas, such as the vehicle's dashboard and other vehicles. Those are not usable features due to the inavailability of a real world representation of them within the point cloud.

3.3 SIFT Feature Detection & Extraction

An interesting point or keypoint in an image is a pixel which has well defined position and can be robustly detected. Keypoints have high local information content and they should be ideally repeatable between different images. Thus it is usually implemented on object recognition, tracking and image matching.

Amongst all of the feature or keypoint detection techniques, the most commonly used in this field are FAST [18], SURF [19] and SIFT [20]. For this thesis, we decided to use SIFT based on the findings in [21], which explained how SIFT topped the chart in terms of precision. While SIFT features are not completely invariant to all affine transformations, a different approach is implemented to allow relative feature positions to shift significantly with only small changes in the descriptor [20]. For future reference, the terms feature and keypoint will be used interchangably in the text.

SIFT was developed by David Lowe and presented as a method for detecting distinctive invariant features from images that can later be used to perform reliable matching between different views of an object or scene. The approach can be broken down into four computational stages: *scale-space* extrema detection, keypoint localization, orientation assignment, and descriptor extraction.

Since the tunnel, and tunnels in general, contains less features than the typical outdoor scenario with plenty of patterns and recognizable features, our algorithm will inherently detect fewer keypoints. This is a challenge indeed on tuning the SIFT parameters to obtained more keypoints and extracted robust and stable descriptors.

All the explanations about each stages are explained in detail in the original paper by David Lowe [20] and some are found from his earlier work [22].

3.3.1 Keypoint Detection

The first step in keypoint detection is to find the local extrema (maxima or minima) at several different spatial scales of the input image. Gaussian blur is applied repeatedly to the input image to produce a set of images, termed an octave. The scale space of the image is defined as a function, $L(x, y, \sigma)$, as a result of a convolution process incorporating input image $I(x, y)$ and a variable-scale Gaussian $G(x, y, \sigma)$.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{3.1}$$

where $*$ is the convolution operation in x and y. The two dimensional Gaussian can be obtained by the following equation:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \tag{3.2}$$

In [22], stable keypoints are efficiently detected by computing scale-space extrema in the Difference-of-Gaussian (DoG) function $G(x, y, k\sigma) - G(x, y, \sigma)$ which then convoluted by the input image.

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned} \tag{3.3}$$

In Figure 3.3 below, on the left-hand side, neighboring images in same octave are used to compute DoG. To compute the next octave as shown on the right-hand side of the same figure, each Gaussian blurred image is downsampled by a factor of two and the process is repeated.

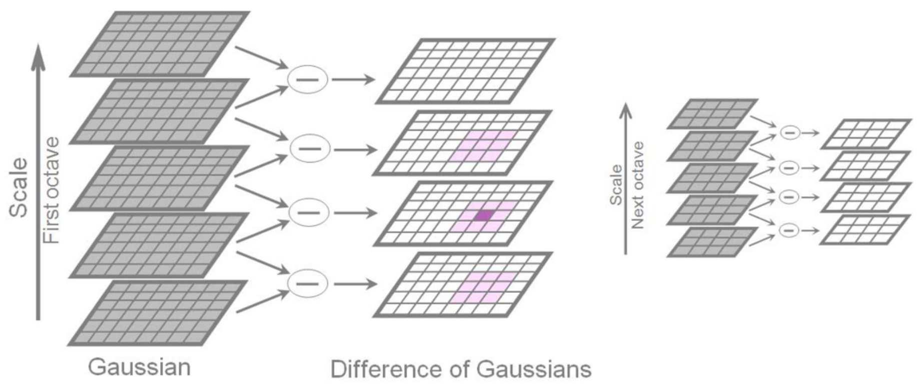


Figure 3.3: Gaussian filtered image are used to create the DoG and downsampling is done upward. The image is adapted from [20].

The last step is to extract the extreme from the DoG images. It is done by comparing each point to the surrounding 8 neighbouring points in the same scale and 3x3 neighbours from adjacent scales, shown as purple points in the figure. At this stage, there are three main control parameters: the number of octaves, the initial Gaussian smoothing value, σ, and the number of scales sampled in each octave.

3.3.2 Keypoint Selection

In the second stage, the keypoint candidates are located by applying a Taylor Expansion series function to the local pixel values. This enables keypoints to be located with sub-pixel accuracy. Weak keypoints with low contrast are then removed to refine the keypoint locations previously identified. For keypoints located along edges, the weak keypoints are eliminated by defining an edge threshold.

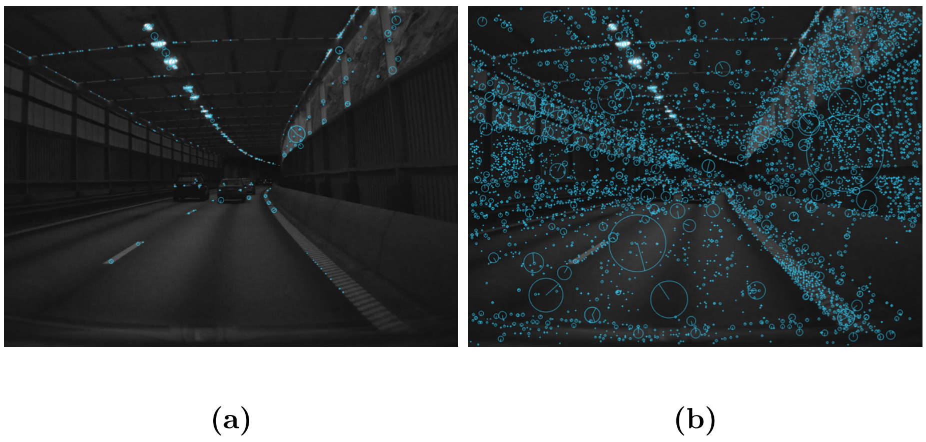


Figure 3.4: Detected keypoints with contrast threshold set to (a) 0.04 and (b) 0.004

3.3.3 Orientation Assignment

After the second step, keypoints with a poor spatial localization have been discarded. In this stage, keypoints will be assigned a principal orientation relative to their rotation to achieve rotational invariant features. It is assigned to each keypoint based on local gradient directions. The magnitude, or weight, of the orientation is heavier the closer they are to the keypoint center. Orientations are shown in 36 bins with 10 degrees each. It is possible for a keypoint to have multiple orientations to increase the stability on matching [20].

3.3.4 Descriptor Extraction

Once keypoints are found, localized and described invariant to both magnitude (scale) and orientations, a descriptor is needed to uniquely identify each keypoint. The descriptor is created from 16 concatenated histograms with each consisting of 8 bins, all computed from the 16x16 pixels around each keypoint. In *OpenCV* a SIFT descriptor is implemented as a 128 long non-negative sequence of digits.

3.4 Feature Matching

Matching feature descriptors between frames is commonly done to find geometric relations between two camera poses. In the visual odometry from [7], tracking features is carried out by performing feature detection in a certain frame then matching them to the following frames. Then, corresponding world points are obtained by triangulating the matched feature points from two different views of a world space. The camera pose is then computed relative to these world points.

The vehicle position, which is derived from the camera pose as explained in 3.1, requires the point cloud as a reference. Feature descriptors that are extracted from the current frame's keypoints are then matched with the lookup table's prior descriptors. In Lowe's SIFT paper [20], comparing the similarities between two descriptors is done using the sum of the squared distances of each descriptor's 128 values. This distance is also known as Euclidean distance or straight distance between two points in Euclidean space. This distance is obtained by performing squared distance between two values from each descriptor.

$$d(p_1, p_2) = \sum_{n=1}^{n=128} \sqrt{x_n^1 + x_n^2} \tag{3.4}$$

where p denotes a feature point in the image, and x its corresponding bytes of information, ranging from 1 to 128. In terms of matching accuracy, no algorithm is known that can identify the exact nearest neighbour in high dimensional spaces other than an exhaustive search [20]. It has very straight forward procedure and is commonly known as a brute-force approach. It compares descriptors from the query set with all the descriptors in the train set and returns a distance describing how resemblant the observed descriptor against all train set descriptors. Here, the *query* set refers to the feature descriptors from the current image while *train* set refers to subset from all descriptors gathered from prior images that have 3D world points correspondences.

When there are descriptors that are located on relatively close landmark when we need only one unique, more effective measure is obtained by comparing the distance to the closest neighbour with the second closest neighbour. If the first distance ratio is less than 0.8 of the second distance, the query point is taken as a matched point, otherwise it will not be taken. This method eliminates 90% of the false matches while discarding less than only 5% of the correct matches [20].

3.5 RANSAC

Random Sample Consensus or RANSAC algorithm proposed by Fischler and Bolles [14] is an approach to estimate general parameter to cope with presence of a large proportion of outliers in the input data. Outliers are elements that do not fit the fitting model within some deviation threshold (error). Unlike many of the common robust estimation techniques which were adopted into computer vision, RANSAC was developed from within the computer vision community [23].

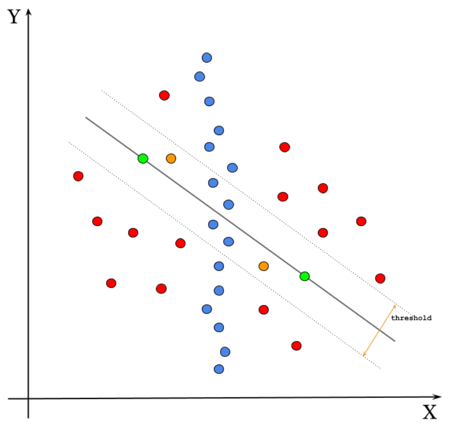


Figure 3.5: An illustration of RANSAC. Red points are the outliers, blue points are expected inliers, green and orange points are guesses.

RANSAC is a resampling technique that discards outliers and generates consensus of inliers by using minimum data points required to estimate the model parameters. As pointed out by Fischler and Bolles [14], RANSAC starts by using the smallest set possible and proceeds to enlarge this set with consistent data points.

The required steps for RANSAC cover several stages. The first stage is to select randomly number of points to determine the model parameters. It is obvious by examining Figure 3.5, that expected outputs are blue points. However, these points are not exactly intersected with the straight line, hence a threshold is needed to allow these points to deviate from a line to some extent. For the second stage, a number of hypothetical guessed inliers are selected and the model is adjusted to conform with these samples. Then all points are tested to this model and all correspond points within the threshold are considered inliers and the rest of outliers. This procedure is repeated a number of times and the correct model is the one with the largest consensus set, or a refined model together with a corresponding consensus set size. For this, the refined model only if its consensus set is larger than the previously saved model.

This master thesis problem lies on solving camera pose problem. Section 2.4 explained how *PnP* can be used to solve this. RANSAC is usually used in combination to pick good correspondences between image points and world points [24]. The sample data would be the pair between image point and its world representation in point clouds. RANSAC will try to fit the model of reprojecting the world points into image coordinates and check the reprojection errors between the reprojected image points and the input image points. Outliers are detected when the error is considered greater than the threshold. Implementing RANSAC on solving the camera pose makes the function resistant to outliers.

3.6 Reprojection Error

A measured point p and its correspondingly reprojected image point p' from a world point X, both in pixel coordinate, are assumed to have some inconsistencies in terms of pixel drift distance, which conveys information about the error of the back-projection algorithm. This potential decrease in precision contributes to the camera pose drifting in accuracy. The accuracy can be numerically described by a so called reprojection error. The reprojection error is a geometric error corresponding to the euclidean distance between a keypoint and its 3D-to-2D projection [10].

Here given two points, p and p', the reprojection error $d(p, p')$ denotes the Euclidian distance between the corresponding points with $p' = P'X$, where P' is the 3x4 camera projection matrix obtained from Equation 2.8, which projects world points into the image plane.

$$d(p, p') = \sqrt{p^2 + p'^2} \tag{3.5}$$

Since reprojection error quantifies the imperfection between a world point with its projection in image plane, it is an important to keep the value as low as possible. Camera calibration accuracy is one of the significant factor on determining this reprojection error and as well as 3D point cloud accuracy.

3.7 Bundle Adjustment

Just like we have image preprocessing before running our algorithm, we are doing postprocessing of the results from the algorithm in order to decrease the error. Bundle adjustment can be applied to a sequence of images, consisting of a subset of all the images we are given, which all show a number of world points from different viewpoints. By using bundle adjustment, we can refine those world coordinates through a process of estimating projection matrices and world points, while also minimizing the distance in image coordinates between our detected points and the corresponding reprojected points.

Bundle adjustment basically solve this problem: Given a set of reconstructed world points X_j, as captured and seen from a set of camera with camera matrices P_i. Each camera projects X_j to $x^i_j = P_i X_j$, such that x^i_j are the projection pixel of X_j on camera P_i. The optimal solution would be the scenario where we have minimum summed square of reprojection error.

$$min \sum_{ij} d(\widehat{P}^i \widehat{X}_j, x^i_j)^2 \tag{3.6}$$

where $d(x, y)$ is the Euclidian distance between image points x and y. This problem is commonly solved using iterative non-linear least square methods. One of the iterative method is Levenberg-Marquardt algorithm. One of the very good solution of the problem has been proposed in [10] and implemented in [25].

4 Method & Architecture

Image-processing has been used mainly on object detection, 3D reconstruction and vehicle positioning. Real-time indoor image-based simultaneous localization and mapping (SLAM) using a monocular camera has been done in [26]. A monocular camera, along with a vehicle model, can be used for visual odometry [27]. Then by combining the camera, the vehicle model and state estimation using Kalman Filtering, in [28], resulted in an estimated vehicle motion for a self-driving vehicle.

In this particular project, Volvo will provide us with raw video data and the logged output from the vehicle's sensors, collected throughout several runs through Gothenburg, including tunnels. We can use the output from a calibrated camera to extract the required information. These images from the camera will be processed to get enough feature descriptors to match them with the point cloud maps.

The 3D point cloud itself is a fixed map, where each point has a fixed coordinates relative to a reference point, or origin. Then, given a calibrated camera with a known initial pose relative to the world map, the detected feature points on the extracted image can be used to augment the point cloud data with relevant descriptors. Therefore, when all the necessary points in the point cloud has been populated with the descriptors, it is possible to estimate the camera pose through the tunnel using feature matching, as has been proven in [6].

Possibly there will be more than one feature point that is matched from the database and there could be outliers that need to be rejected [29]. An additional technique, bundle adjustment, could be implemented to refine the 3D coordinates of the scene's geometry.

4.1 Software Architecture

Below is the high-level architecture our software will be adhering to, as represented by the activity diagram in Figure 4.1. It shows the flow of the application without diving into each part.

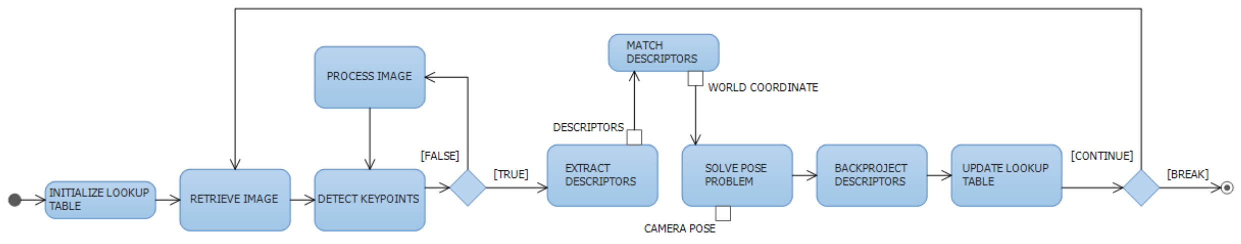


Figure 4.1: The activity diagram describing the high-level architecture of our software.

As can be seen, the first thing we do is to initialize the state of the application and load the point cloud. As the point clouds are very big and can take a lot of time to load, we only want to do this once. By following the top path, we begin our primary loop by retrieving an image. Using this image, we run the SIFT detector in order to detect as many keypoints as possible. Next, a decision has to be made, regarding whether we found enough keypoints to get a good result in the next steps, or do we have to process the image in order to try and find more keypoints. This processing can consist of making the image brighter, for example. Once we have detected enough keypoints, we run the SIFT extractor to extract the descriptors for each of the keypoints. The descriptors we find with SIFT in the image can then be matched against the descriptors we have stored in our lookup table together with the corresponding world coordinates. By taking this corresponding world coordinate, together with the image coordinate for the SIFT descriptor, we can use the correlation between these coordinates to solve the camera pose problem. In the following steps we use this camera pose in order to backproject the descriptors we find onto the point cloud, basically augmenting the world points with a SIFT descriptor for the matching step in the next iteration of the loop. When we say augment the point cloud, in practice we will be updating the lookup table we have been mentioning, where each entry contains a world coordinate and its SIFT descriptor. At this stage, we have to decide whether we continue with another iteration of the loop, or if we have reached the end and should stop. As we are dealing with image sequences, when we decide to stop is whenever we run out of images, or at some specified endpoint.

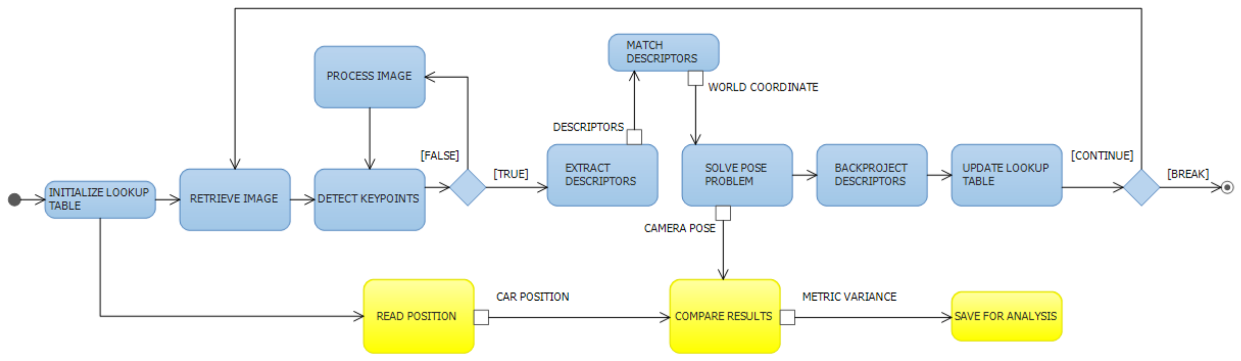


Figure 4.2: The activity diagram describing the high-level architecture of a whole solution, comparison included.

In each iteration of the loop, when the camera pose problem has been solved, we do the computation of where the camera is positioned. When we have this information, we can compare it to whatever positioning information is available, for example GPS data from the logs. As can be seen from the bottom part of our activity diagram in Figure 4.2, we can read the position as calculated in real-time by the car in the logs, and then compare this to the results from our PnP solver. The metric variance between these two would arguably be the most important evaluation criteria for the end product.

For the interested reader, we have also chosen to describe the software more in-depth, showing how objects interact with each other through function calls and requests, in the sequence diagram below in Figure 4.3. The architecture is built mainly from the five classes besides the main function, as seen in the Figure below. The primary part of the software is the image sequence loop, from which we find features, solve the camera pose problem and also backproject features onto the point cloud. Before this, we mainly do initialization and preparations. This includes loading our point cloud which, because of it's size, is quite slow. From this, we can build the kdtree which will be used whenever we need to search for a point in the point cloud. This is handled by the 3rd party library PCL. We also prepare our Lookup Table with the manual correspondences, which are loaded from files in local storage before the loop begins.

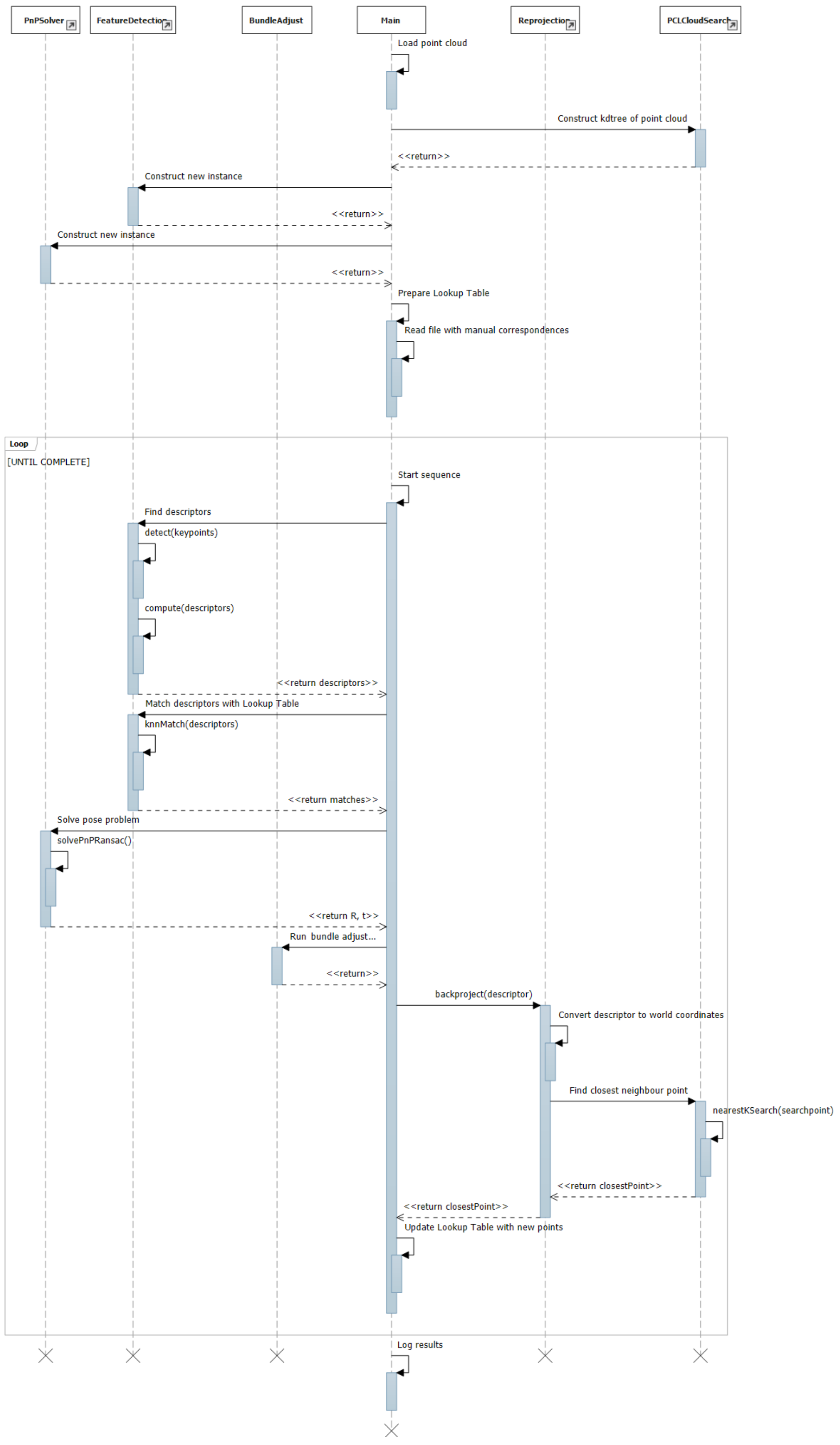


Figure 4.3: The sequence diagram describing the flow of the software throughout function calls and objects.

We have already discussed most of what happens in the image sequence loop. However, here we see that bundle adjustment is also applied before we run the back-projection procedure. The purpose of this is basically to refine the rotation and translation of our camera, before we use these properties to build the ray that is used for finding a point in the point cloud.

4.1.1 Third-Party Software

For the project, a number of open-source libraries have been utilized. There is no reason to reinvent the wheel and as such, we make use of the image processing algorithms and point cloud processing already developed in OpenCV and PCL.

OpenCV

OpenCV is a library that already contains a vast amount of computer vision algorithms, originally developed by researchers from Intel, but now maintained by Itseez [30]. For SIFT, it lets us detect the keypoints in the images that will be used when comparing and finding matches in the point cloud. The code has also already been optimised, is well known and documented, and is free for use under the open-source BSD license.

PCL

PCL, or Point Cloud Library, contains algorithms for processing point clouds and was originally developed by Willow Garage. When doing our reprojection from image to world coordinates, PCL can be used for finding the node closest to the projection ray. Just like OpenCV, it is also released under the BSD license.

Matlab

MATLAB is mostly used in our project for camera pose verification within the point cloud. The point cloud is derived from a polygon type and PCL does not support this right out of the box. Conversion, and sampling of the point cloud such as finding the k number of nearest point neighbours, is done through MATLAB functions.

4.1.2 Result Verification

Verifying the motion of a camera through image sequences in a forward motion of vehicle is done by comparing the camera position with a known ground truth. This ground truth however, mostly acquired through an accurate GPS device on a vehicle that produced exactly the same run while recording the frames. The metric comparison is done by finding an initial pair of camera position and GPS sample and then sample both data accordingly if the frames are not time synced with the GPS samples.

For this particular thesis, the result is done by comparing the pose for each frames with a sample of preprocessed GPS result taken by the same vehicle. The GPS data are provided by Volvo Cars, as well as with other data. One may ask about the accuracy of the GPS as there no such thing as an accurate GPS within confined area like tunnel. Hence, a scenario outside the tunnel is created and used to verify the solution with the GPS data. Then we could argue given the solution that is equally better with the GPS for that specific scenario therefore for the scenario inside the tunnels one could expect that our solution should be as good as the preprocessed GPS data.

5 Implementation

Following will be a description of each important step in the flow of the program. Key algorithms will be pointed out and described, and limitations will be made clear. As we are considering offline positioning, there are almost no constraints on time or computation power. Because of this, we will be able to use very complex, or even slow, methods and algorithms for the sake of accuracy. Instead of trying to minimise the computation time, we focus on achieving as good positioning results as possible.

5.1 Building Initial Correspondences

As explained in the Chapter 3, given correspondences between world coordinates and a number of known points in the image, we could find the camera pose relative to the world coordinates. In short, by finding a geometric relation between a pixel and its position in the world, we can estimate the world position of the camera center relative to the same origin, which in this case is represented by the point cloud.

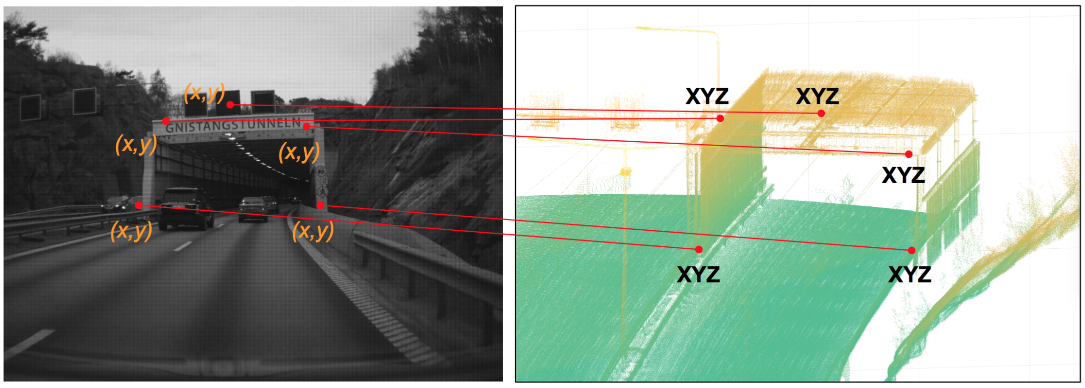


Figure 5.1: An illustration of image to point cloud correspondences at the entrance of the tunnel

Thus, we represent this 3D-to-2D relation as a look-up table which consist of the feature descriptor information and coupled with 3D world coordinates. Algorithm 2 illustrate a mock up sequences for building manual 3D-to-2D correspondent.

Before the algorithms begin, an initialisation of the lookup table is done by finding 2D-3D correspondences. First, we detect feature points on the initial image. These features are used to manually handpick the 3D points within the cloud. Based on the work by Lepetit et al in [31], at least four correspondences are needed to estimate the camera pose. In our initialisation, 10 image to world point correspondences are used to solve the camera pose problem. It satisfies the minimum requirement for both the native perspective-n-point and *ePnP* algorithms in OpenCV.

Algorithm 1 Manual 3D-to-2D Correspondences

```
Input: Initial image, point cloud
Output: Pairlist of $m$ SIFT descriptors and world coordinates

1: Detect $n$ features
2: Extract the $n$ descriptors
3: Manually find $m$ correspondences in the point cloud
4: for each corresponding 3D coordinate do
5:     Augment the coordinate with the descriptor
6:     Build the lookup table
7: end for
```

The lookup table consist of information that will be used on finding the initial vehicle position close to the whereabout of the initial image. To avoid the lack of accurate positioning, the initial image is captured right before the tunnel entrance as shown in Figure 5.1. This lets the vehicle still have a good GPS measurement, which can be used as a comparison to the estimated camera pose. However, this needs all images to be timesynced with the GPS samples. Nevertheless, the lookup table can be created once and could later on be used for subsequent runs throughout the tunnel.

In the subsections below, feature detection, descriptor extraction, how the lookup table can be built and how to use the correspondences to get camera pose will be explained further.

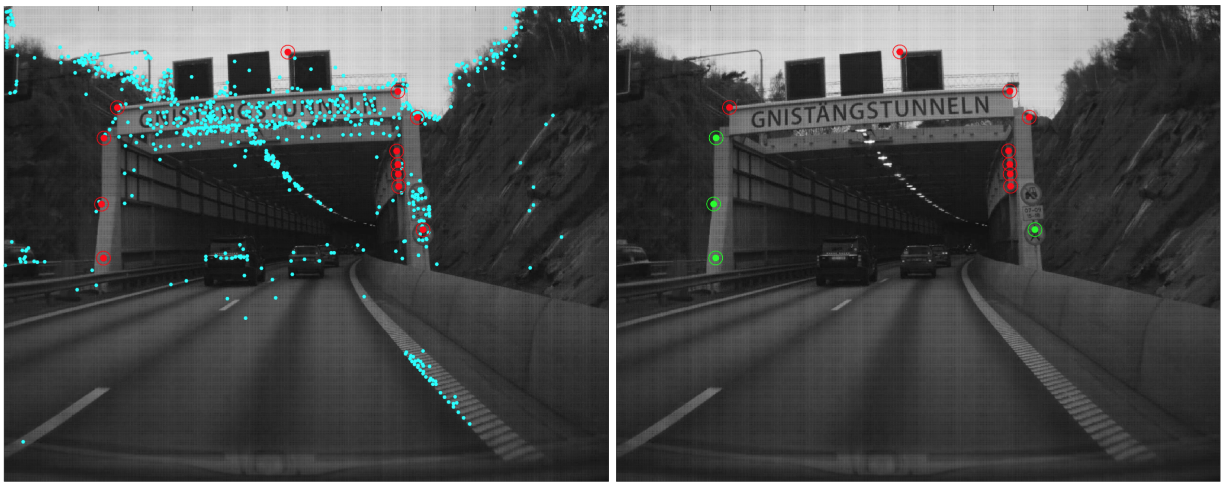


Figure 5.2: Red marked are manual correspondences stored in the lookup table. Four green points on the right image are the best points amongst manual correspondences to minimize the initial reprojection error.

After we detect the features, descriptor for each features are extracted using SIFT. And at the end of the algorithm sequences, we initiate the lookup table which consists of a pair of descriptor and 3D points $<Descriptor_i, X_i>$. The index i represents the key index for the lookup mechanism. The lookup table is staticly defined initially and will dynamically updated every new camera pose C_k has been obtained.

The location for the manual correspondences could be located on any place that also happened to be within point clouds. However, quality of image needs to be considered as bad image with bad features are prone to the noise.

5.2 Feature Detection

An interesting point or feature in an image is a pixel which has a well-defined position and can be robustly detected. Features have unique, recognizable content and should ideally be repeatable between different images. Thus, it is usually implemented in object recognition, tracking and image matching.

Since the tunnel contains less features than the typical outdoor scenario where there might be for example brick walls or other easily recognizable areas, this will result in a lower number of detected features. In Figure 5.3 we can also see how different the lighting conditions are, comparing between just outside the tunnel and when driving inside. The amount of features that can be found is drastically reduced due to this. Thus, finding a good feature detector is a crucial step for the whole solution.

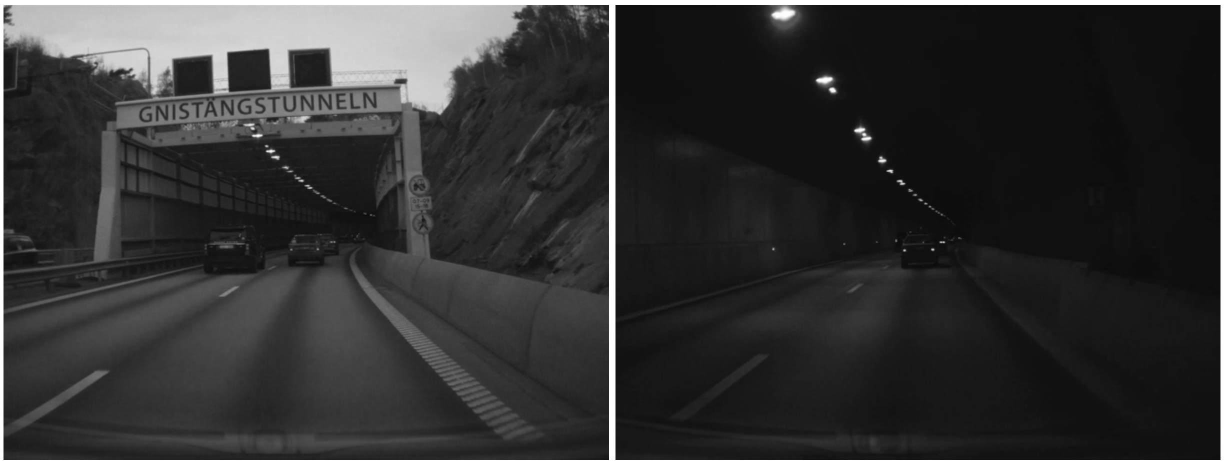


Figure 5.3: Sample images representing two of the sections of the *Gnistangstunneln*, the entrance (a) and the inside (b)

The parameters we used for all the feature detectors were the default ones, as given by the OpenCV. Further tuning and refinements are done, especially when the number of features drastically reduce as the vehicle drives deeper inside the tunnel. Moreover, most of the detected features are located on the lamps throughout the tunnel, since it is practically the only area which has enough brightness and contrast. This is a situation that was expected. In the Figure 5.4, the results of detecting features along the road are presented.

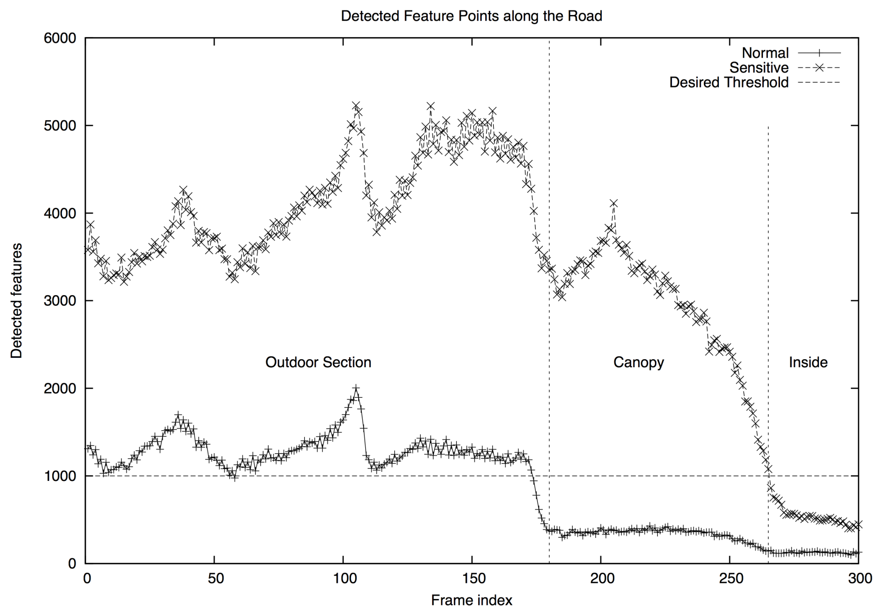


Figure 5.4: Normal (default SIFT parameter) and sensitive (tuned SIFT) are presented for all 3 different areas, outside, canopy and inside. The simulation starts from frame index 150 to 300 which gives us 150 frame sequences on all scenarios, i.e, outdoor, canopy and inside.

However, the amount of detected features does not directly measure how accurate the algorithm is. It is practically better to detect n number of unique and well spread out features, instead of m more unstable and clumped up features, where $n << m$. The number of features depends on several factors. First and foremost, the quality of the input images. Every image parameter, including for example resolution, sharpness and contrast will determine the number of detected features and also the quality of each feature. Second is the SIFT parameters, such as its sensitivity, which needs to be tuned to get a sufficient number of features.

5.3 Feature Matching and RANSAC

When features for one image are obtained, the descriptors are computed and used for matching on two occasions. First is to match those descriptors with the lookup table. This matching is specifically used to retrieve the camera pose for one particular image. As the initial correspondences are considered accurate, then if a sequence of images, I_{i-2}, I_{i-1} and I_i contained enough matches according to a given threshold, then the camera poses C_{i-2}, C_{i-1} and C_i can be derived.

Another condition for two features to be considered matched, other than the one explained in Section 3.4, is to track the image coordinate translation on both images. Given a k-th feature point F_i^k found in image I_i with a pixel coordinate of (u_i^k, v_i^k), when a feature F_{i+1}^k on the next image I_{i+1}, is match according to the Lowe's ratio test [20], the Euclidean translation distance between those two points should not bigger than some radius of squared distance. In this project, we decided to only consider matches that are located within a 25% diagonal of the image compared to the previous image.

5.4 Perspective n-Point

Once the manual correspondences are stored in the table and the feature handlers are ready, an image sequence is processed by detecting the keypoints, extracting the descriptors and then matching these with the lookup table entries. Once the number of matches is sufficient, the camera pose for the specific image is computed using a *perspective n-point* method. The P3P method takes a minimum of four 2D-3D correspondences and if we have more than the minimum number, RANSAC is applied to take the best fitting four correspondences that yield the minimum average reprojection error for all the points.

The idea is to take just four correspondences as an initial input to the *SolvePnP* function from OpenCV, then once we obtained the R and t matrices from the function, we project all the rest 3D points using equations 2.4 to 2.6. Then we obtained the average reprojection error $\hat{d_i}$ for the particular i-th attempt for corresponding four points. We do this for a number of iterations as many as 1000. This is a recommended default value from OpenCV and it should be sufficient to converge best result as such as $\hat{d_k}$ after $k - th$ attempt is the minimum using the four best corresponding points. These four best 2D-3D points are then used to calculate camera position in the world space or coordinate.

5.5 Image Tracking

In visual odometry, the motion is obtained by comparing two subsequence images, e.g., I_k and I_{k+1}, such that the camera pose C_{k+1} depends on the previous camera pose. In this thesis, we tracked several images I_k to $I_{k+(N-1)}$, with N is the window size.

In Figure 5.5, it is shown how the last three images are tracked within a window of size $N = 3$. X_j represents the set of 3D points that are matched between subsequent images within the window. One of the challenges here is to pick the size of the window since it affects the optimization phase in bundle adjustment. Reliable initial camera poses for the first N poses are important as it affects the following image's computation. The idea is not to chain two images, or picking window size of two, since the error would be bigger than window size three or four [32]. Thus, we opted for a window size equal to five for the whole sequence of images.

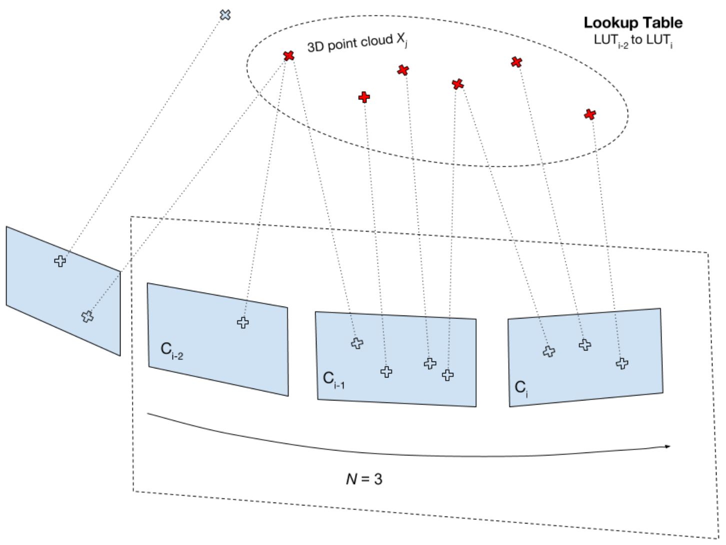


Figure 5.5: The $i-th$ stage of camera motion within sliding window with $N = 3$. Only N number of images and corresponding 3D points are considered for the next image I_{i+1}

5.6 Position Estimation

The position estimation is based on some prior works on visual odometry [8][32], Visual SLAM [6][33]. The approach we are using in this particular project share some key technologies, e.g., the approach of finding 2D-3D correspondences between features and point clouds and then used to estimate the camera pose. Some works on *Structure from Motion* (*SfM*) [29] for estimating a camera pose also rely on the 3D structures built prior to the start of the process. Triangulation is one of the common methods to form 3D point arrays from 2D images. The 3D point is determined by intersecting back-projected rays from 2D image correspondences of at least two images. In perfect conditions, these rays would intersect in a single 3D point. However, because of the presence of image noise, camera calibration errors and feature matching uncertainty, they are never guaranteed to intersect.

Algorithm 2 Position Estimation

Input: Image at time k, Point cloud $PCloud$
Output: Camera pose C_k, Reprojected 3D points at image-k X_k^j

```
 1: Load $PCloud$
 2: Build correspondences and prepare the initial lookup table
 3: Clear window
 4: for all $M$ images do
 5:     Capture new image $I_k$ and preprocess the image with ROI
 6:     Extract features and descriptors for image $I_k$
 7:     if Window is empty then
 8:         Feature matching $I_k$ and lookup table and retrieve 3D-to-2D corresp.
 9:     else
10:         Feature matching $I_k$ and image within window and retrieve 3D-to-2D corresp.
11:         Perform RANSAC to reject outliers between each matching
12:     end if
13:     Solve the camera pose using $PnP$
14:     Backproject all features into point cloud
15:     Add image-$k$ to the window
16:     if Window is full then
17:         Bundle Adjust $N$ images within the window
18:         Remove the first images inside the window
19:     end if
20:     Increase timestep $k = k + 1$
21: end for
```

This thesis however, has 3D point clouds representing the scene right from the start. Thus, rather than building the 3D point representation, we just need to find the corresponding world points within the point cloud using back-projection. However, this could not be done without a known camera pose. In other words, finding the 3D points for all the 2D feature points in an image k relies on the camera pose C_k, since the back-projection rays originating from the camera center on world coordinate $center_k^w$ can only be derived from the camera's pose.

Algorithm 2 presents the needed steps for solving the positioning problem for this project, excluding the back-projection that will be presented in the following section. The routine starts by loading the point cloud before representing it as a $kd - tree$ by PCL library. Then the correspondences for the initial lookup table is loaded and the first image can be processed.

This first image could be anywhere as long as the number of matched points between its features and the lookup table is enough to derive a good initial camera pose $C_{t=init}$. Then it marks the starting index for the following images that lead into the positioning sequence. Backprojecting all 2D points of detected features into the point cloud is the last step before the bundle adjustment starts, once the window is full. Bundle adjustment will refine the rotation and translation matrices, and the positions of the successfully back-projected 3D points.

5.7 Back-projection and Updating the Lookup Table

When the camera pose problem has been solved and we know how the camera is pointing, back-projection can be performed by theoretically shooting a ray originating from the camera center, through the image plane and onto the point cloud. It lets us find the most representative point in the point cloud. This is very useful as it lets us augment the points in the point cloud with the SIFT descriptors that we extract from the camera images. By augmenting the points with the same features as were found in the image, we can match 3D points with image pixels and thus automatically find the correspondences needed for solving the camera pose problem. One important detail to note is that we have assumed that the principal point of the camera unit, as specified by the intrinsic camera matrix and explained in Chapter 2.2.2, lies in the exact center of the image plane. Realistically, this is rarely true, and is only done here because the exact coordinates were not given in the supplied camera calibration information.

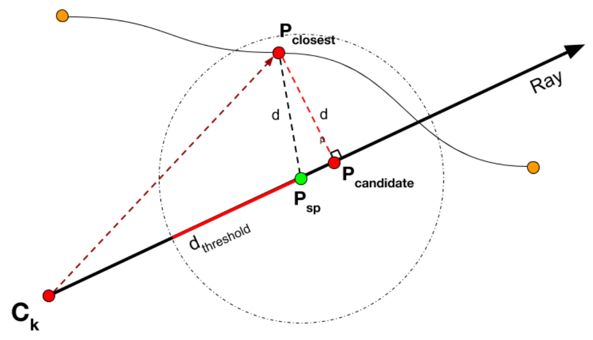
Algorithm 3 Back-projection & Lookup Table

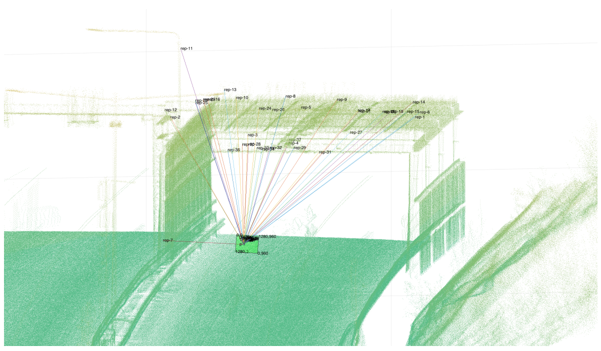
Input: Camera pose C_k, SIFT descriptors from n features, Point cloud
Output: Up to n number of 3D-to-2D correspondences

```
1:  Build $n$ lines, originating from camera center $center_w^k$ to all of the $n$ 2D points in the
    image
2:  Transform the line into a 3D ray coordinates
3:  for all $n$ rays do
4:      Find an intersection with the point clouds
5:      if intersection is found then
6:          Get the XYZ coordinates from the point clouds
7:      else
8:          Find the nearest point $X_{nearest}$.
9:          If the distance is less than $d_{threshold}$, orthogonal project the $X_{nearest}$ to the ray
    to interpolate the intersecting point
10:     end if
11:     Extract the 3D point coordinate $X_{intersect}$
12:     Build a pair of $<X_{intersect}, Descriptor_{intersect}>$
13:     Add the pair into the lookup table
14: end for
```

The algorithm for back-projection requires the camera pose matrix to be known for the current frame, since transformation from image to world coordinate requires the camera pose problem to be solved. The back-projection is performed by constructing n number of rays, starting from the camera center in world coordinates, traversing through the image plane, intersecting the n number of image features.

From Figure 5.6a, for each searching point P_{sp}, nearest-neighbor searching algorithm is performed to find the closest world point in the point cloud. From the Algorithm 3, the closest distance between the ray and the world point candidate should be lower than $d_{threshold}$, which should be equal to the half of the minimum distance between two





(a) The closest point $P_{closest}$ is picked if the squared distance d is equal to or less than $d_{threshold}$.

(b) The result of succesful back-projection on the first frame after succesful attempt on solving the camera pose.

points in the point cloud. Once we find the point with the closest squared distance to the line, $P_{closest}$, an orthogonal projection to get an approximation point $P_{candidate}$ is applied following algorithm as explained in [34]. Finally the $P_{candidate}$ is augmented with the corresponding 128-byte SIFT descriptor obtained from the feature detection and descriptor extraction.

$$< P_{candidate}, Descriptor_i^{128} >.$$

The list of succesful back-projected points containing 3D coordinate and their respective SIFT descriptor then are pushed to the lookup table for future usage. Obviously, there are missed back-projection caused by several factors, e.g., difference in point-to-point distance in the cloud and the unavailability of specific areas of the image which contained 2D features in the 3D model. Result of the back-projection at camera initial pose is shown in Figure 5.6b.

5.8 Sliding Window Bundle Adjustment

This thesis implemented bundle adjustment using a third-party library, SBA [25]. Once the sliding window is populated with the last N images, bundle adjustment is performed. SBA requires data to be prepared in such a way that all reprojected 3D points from cameras C_k are visible in one or more cameras.

As we explained in the Section 5.5, the five first camera poses build the initial parameters for a *bundle*, and these poses have to be considered accurate. Therefore, the first five images that are still located outside the tunnel with enough brightness and contrast are used to guarantee the optimal initial poses for bundle adjustment.

Assuming that we have five images, I_i to $I_{i+(N-1)}$, stored in $\widehat{I}_{window}$, which contribute to all the 3D points that will be used to construct the structure. For all these 3D points, it is most likely that not every point is visible in all the images. Thus, a binary variable

Algorithm 4 Preparing Bunde Adjustment

Input: N images
Output: N refined camera parameters and M_i 3D points from each camera C_i

```
1: Concatenate $M_i$ points from each camera and store it in $\widehat{M}_{window}$
2: Remove the duplicate points from $\widehat{M}_{window}$
3: Initiate $visibility[N][\widehat{M}]$ equals zero
4: for all $\widehat{M}_{window}$ 3D points do
5:     for all cameras $N$ do
6:         Set visibility 1 if point $X_j$ is visible to the current camera
7:     end for
8: end for
9: Run bundle adjustment for $N$ images
```

visibility is needed to determine whether the corresponding image could project the 3D point into its image plane or not. Thus, given M number of 3D points from N cameras, visibility is a 2D vector $visibility[N][M]$, such that if $visibility[i][j]$ is 1, then point X_j is visible on image I_i.

5.9 Defining the Ground Truth

When evaluating the results of an implementation, you would want an as accurate ground truth as possible to compare against. GPS measurements from inside the tunnel are not good enough to be used as the ground truth, since the blocking of its signals will stop it from working properly. We will provide two approaches on defining the ground truth. The first one is based on statistical approach since post-processed GPS measurements from several runs throughout the tunnel are available. From these runs, ten runs were carefully chosen for building the ground truth considering the lane and the consistency in the trajectory. The second method is based on the manual observation of the center lane markings which are apparently visible on the point cloud.

For the statistical approach, the first step is to choose one run as the first reference and extract all the vehicle position data (X_n^k, Y_n^k, Z_n^k) from within the tunnel. XYZ is the vehicle's metric position and n is it's number in the sample sequence. For the next step, nearest-neighbour searching was performed to find the starting point for the other runs. These points are basically the closest points to the starting point of the first run we chose, (X_1^k, Y_1^k, Z_1^k). Since all the runs have exactly the same sampling rate, further rate matching or synchronization is not needed. Each of the n samples for all the runs are extracted, started from each run's corresponding starting point.

These 10 set of post processed GPS samples which started at coordinate that relatively close to each other, may ended in different coordinates with bigger distance. This is caused by the different vehicle speed when recording each GPS runs. Thus, in order to synchronize all the data, a new reference $(X_n^{ref}, Y_n^{ref}, Z_n^{ref})$ is built using the mean value from all the runs for each adjacent sample. For each sample of the reference, in Euclidean

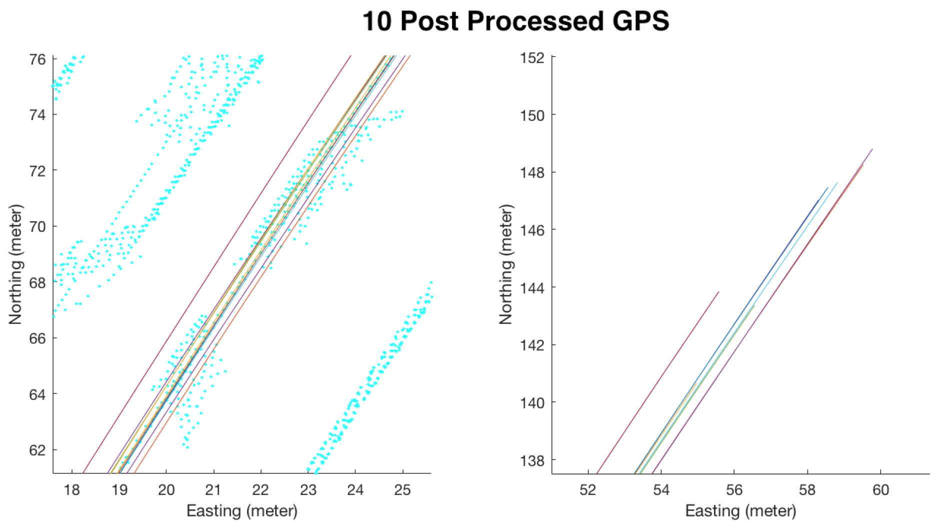


Figure 5.7: Plot of all GPS samples within tunnnel.

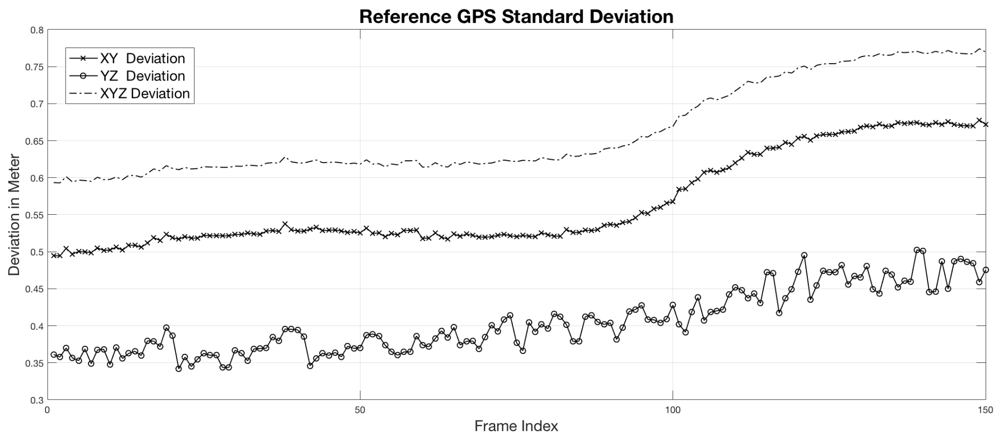


Figure 5.8: The deviation for the reference ground truth is shown in XY (lateral), YZ (altitude) and XYZ (combined).

XYZ position, searching for the closest point of other runs were performed. That will give us new ten GPS trajectories synchronized with each other.

The standard deviation for new trajectories is defined in such a way that the mean trajectory is safely estimated within a measurable distance interval. This interval is the two sided deviation which basically explain how the spread of the GPS position at particular time based on the distribution of all the available samples. The confidence level in used is 95%, which explain that 95% distribution will be located within the two sided range of (+/-) $1.96 * deviation$. For example, based on Figure 5.8, at frame 100, the XYZ deviation is 0.6558 meter from the mean. The value is 1.96x of the standard deviation which represent 95% of the distribution data based on known standard deviation of 0.3346 meter. More on this figure, as the vehicles are travelling inside the tunnel, reduced reliability of GPS measurement is represented by the increasing value of the deviation.

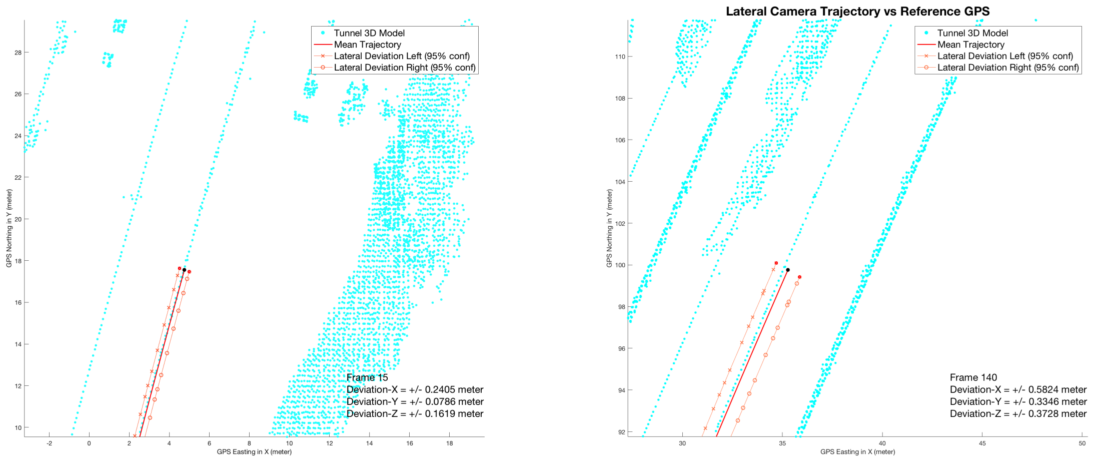


Figure 5.9: The reference vehicle trajectory based on the 10 post processed GPS runs. Leftmost figure represents the outdoor scenario with small deviation, and the deviation has increased (wider) as the vehicle went deeper into the tunnel (figure on the right.

To visualise the our reference trajectory with its deviation, Figure 5.9 shows how two boundary lines for both minimum and maximum deviation limit the vehicle true position based on our estimation and statistical approach. The deviation value would be bigger as we increased the confidence level and vice versa.

With this we have built our ground truth using combinations of post-processed measurement and statistical approach. Another method to build golden standard ground truth is to filter the middle lane of the 3D cloud model of the tunnel. Since the point cloud has been augmented with laser scanner data, certain textures such as lane markings are visible and could serve as good points to track the middle lane. Estimation of the vehicle trajectory can then be calculated by using an area estimation relative to the middle of the lane.

The second method is derived from an alternative method which required manual onsite

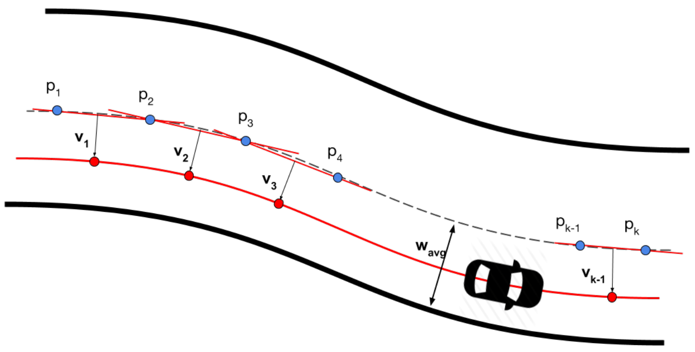


Figure 5.10: The vehicle trajectory (red continuous line) is obtained by calculating vector normal v_i for every two center lane markings. The magnitude of each vector is uniform as it is computed from half value of the average road width w_{avg}.

survey which is not in the scope of this thesis due to the time this would require. However, it was possible to detect the center line of the road due to visible lane markings. The idea is to estimate the vehicle trajectory by calculating vector normal of every two subsequence lane markings as shown in Figure 5.10. For every two markings, we computed the gradient and calculate the vector normal to the right side of the lane with magnitude equals to some values. Since from the video observation, vehicle was driven steadily in the middle of the right lane, it provided an estimation of vehicle whereabouts. Since it consistently stay in the middle of the right side lane, the magnitude is equal to the half of w_{avg}, or the average of road distance from the center to the right side border which consistent around 3.5 meters. Then the computed new points (red dot) from the figure are used to estimate the new vehicle trajectory.

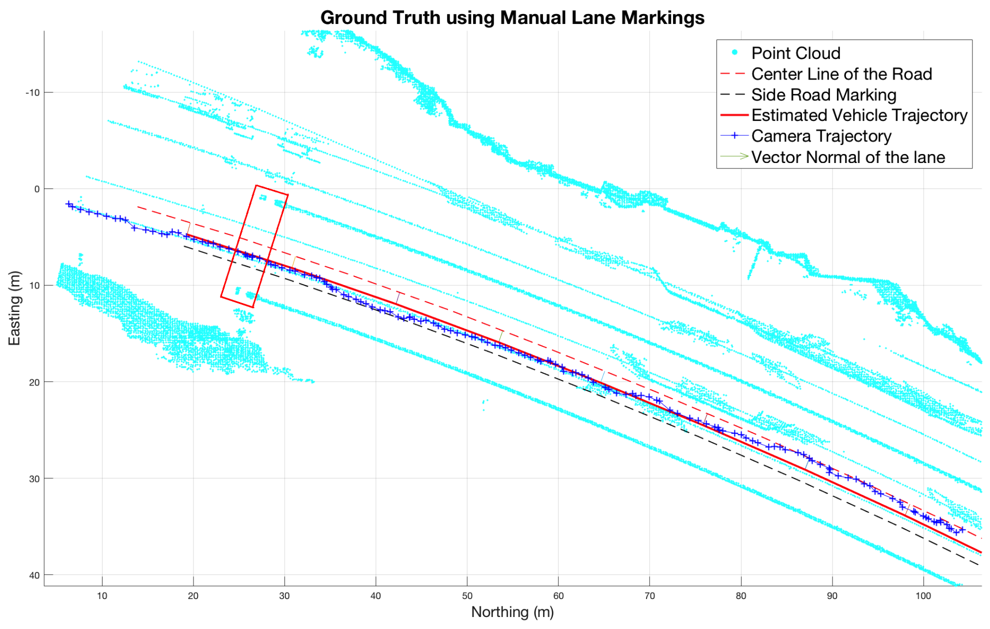


Figure 5.11: The camera trajectory as compared to the reference GPS based on the center lane markings. The entrance of the tunnel is marked by the red box.

Figure 5.11 gives an illustration of the vehicle trajectory as opposed to this reference trajectory. While this reference is considered to be appropriate and suitable for the camera positioning as they are using the same point cloud, but since the number of lane markings is much more less than the camera position samples, it is non comparable in any metric comparison until line interpolation is done to increase the number of road markings to a sufficient number. Therefore, the metric comparison to provide, e.g. drift distance, is done using the statistical reference instead.

6
Results

In this chapter, results from implementation on the input frames will be presented. The number of 3D points per frame, reprojection and then final positioning of the vehicle using camera pose will be shown and described. The observed 3D points and vehicle positions are derived from a sequence of 150 frames that cover the entrance part and the canopy part of the tunnel. The sequence starts within index 150*th* to 300*th* referring to the whole observed frame from Figure 5.4. Additional results from lateral and altitude translation between camera result and the ground truth, which represented by the preprocessed GPS will be added as a source for discussion.

6.1 Initial Camera Pose

In this first part of the result, we would like to show the accuracy of the initial correspondences in terms of reprojection error. The number of correspondences, as we explained in Subchapter 5.1, is an arbitrary value as long as it satisfies the criteria of the implemented *PnP* algorithm. The accuracy of each 2D-to-3D correspondence in this initial correspondences is important for the whole solution as it makes up the primary input for solving the first camera pose. Later, it affects the subsequent frames since the back-projection of the following frames depends on the previous pose.

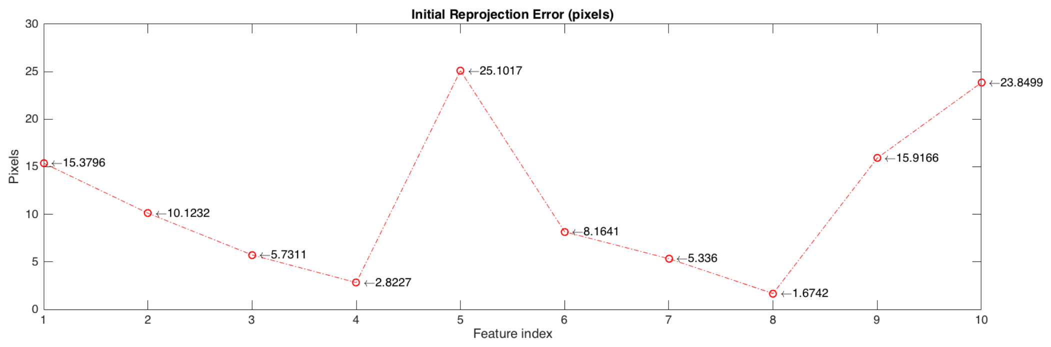


Figure 6.1: Reprojection error of initial world points in the first frame

Given the 2D-to-3D correspondences, the *PnP* solver provides two camera parameter matrices: the rotation (R) and the translation (t) matrices. To calculate the reprojection error, we use Equation 3.5 and the newly found R and t to transform all points from the world coordinate space into the camera coordinate space. Further multiplication with the intrinsic camera matrix K gave us pixel coordinates for all the transformed points. The reprojection error is then obtained by comparing the pixel distance between the original 2D feature points with the corresponding transformed points, following Equation 3.5. Figure 6.1 shows the reprojection error in pixel distance for each of the manually extracted image points. In the graph, it is clearly shown that the error is rather substantial with the maximum reprojection error reaching 25 pixels and therefore it is one of the main areas for future improvement.

Axis	**Initial (m)**	**Nearest GPS (m)**	**Drift Distance (m)**
Easting (X)	143427.55	143427.58	-0.0296
Northing (Y)	6394339.012	6394338.92	0.0917
Altitude (Z)	32.35	32.13	0.2193

Table 6.1: Comparison of initial camera position and the nearest GPS sample, all numbers are in SWEREF99 metric before shifted in future graphs.

The camera position is obtained using Equation 2.9. The first thing to measure is how far the initial camera position drifts from the ground truth, which is represented by the post-processed GPS data from other vehicles. The closest GPS sample to our initial camera position is taken as the starting point for the comparison for the subsequent frames. Table 6.1 provides a comparison of these two reference points and how far the drift distance of the initial camera pose is in each axis. Noticeably, the biggest drift was found in the altitude, with a drift of approximately 22 centimeter, while the lateral drift is less than 10 centimeters for both Easting and Northing.

6.2 Back-projection

The back-projection algorithm is designed to interpolate the calculated world coordinates for every feature point within each frame into the point cloud. From Algorithm 3, all rays originates from the camera center, which we specify as the homogeneous coordinate $[0, 0, 0, 1]$ in the camera coordinate space and is transformed into the world coordinate space. In short, the camera center notated in the world coordinate space is used to determine the vehicle position.

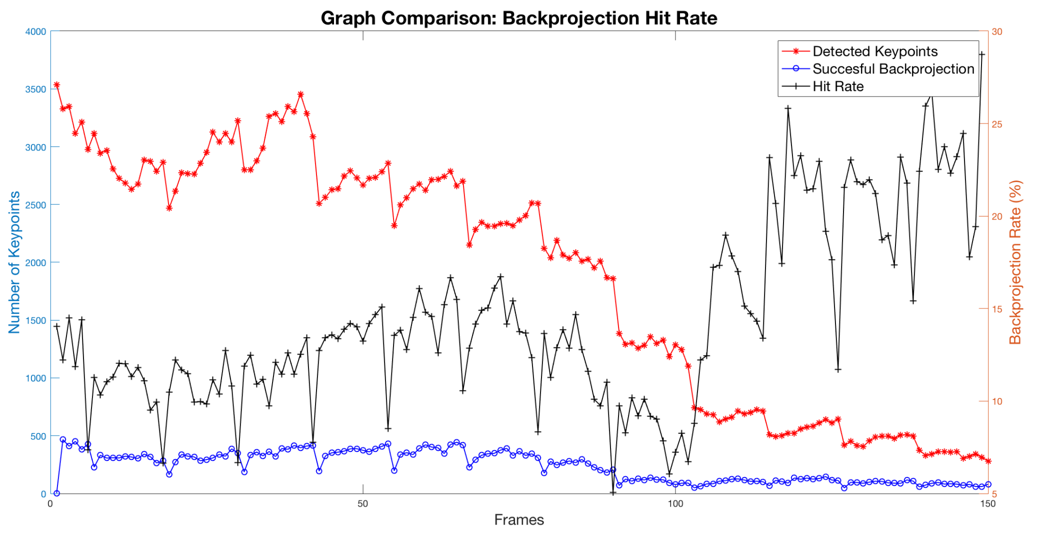


Figure 6.2: The number of back-projected feature points per frame. Depicted also is the hit rate of back-projection succession in percent.

Figure 6.2 shows the number of 3D points that were used to calculate corresponding frame's camera center. The first frame used as minimum as four 3D points from the initial manual correspondences and the rest used back-projected points using previous frame's camera center as the origin of the rays. As we can observe, the diminishing rate of the 3D points is significant as the vehicle reaches the canopy part of the tunnel, starting from frame index 10. This has been motivated by the diminishing number of good quality features due to increased in sensitivity, as explained in Figure 5.4. Frame index 50 marks the end of the frame sequence with a relatively good amount of 3D points.

6.3 Positioning

The vehicle position is derived from the camera position for the sequence of 50 frames after the initial frame. As this positioning is verified for forward motion of the vehicle, in the discussion chapter we do the backward motion of the frames to extend the comparison data to utilise outdoor scenario. This is important to verify that our solution is as good as the GPS data with regards to the accuracy limit set by the initial camera pose. The vehicle trajectories are available in 3D as the camera has 6 degree of freedom (6-DOF) as stated in the camera parameters. Thus the solution was presented in two views, lateral top down and altitude views.

The GPS dataset is given by Volvo and it is a offline preprocessed GPS. It contains several runs on both side of the road and we retrieved the samples using nearest sample points. The comparison covers both lateral (X-Y) and altitude (X-Z) of both camera and GPS position.

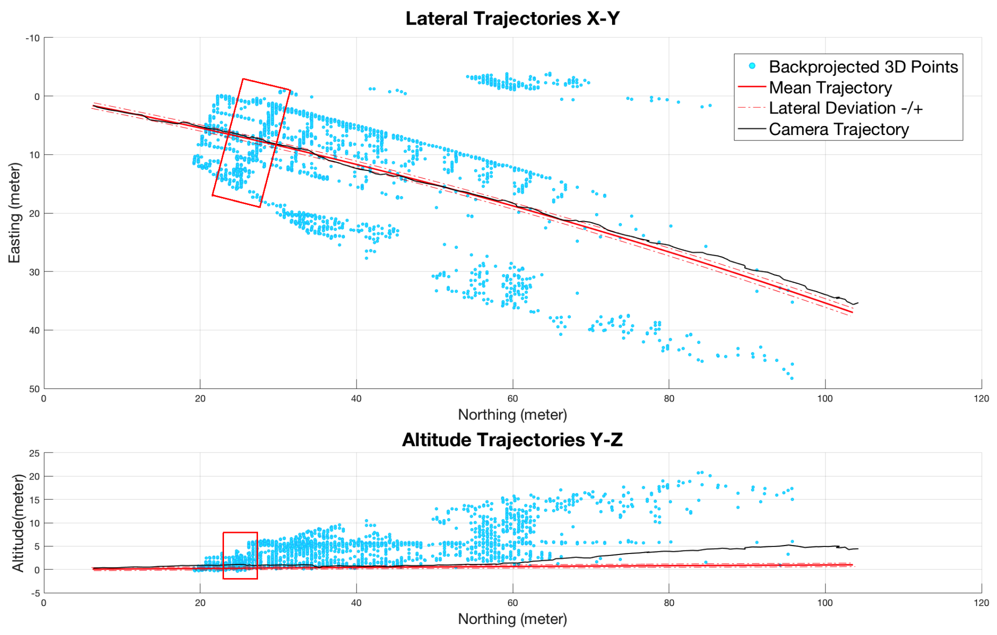


Figure 6.3: Vehicle/camera trajectory as opposed to the reference ground truth. The top side depicts the lateral (x-y) trajectory. The bottom side concerns the altitude (y-z) trajectory. The coordinate is shifted to scale down the GPS data to make it relative to the vehicle starting point at frame-0. Red boxes mark the tunnel entrance.

Figure 6.3 represents the vehicle's trajectory, from both lateral and altitude measurements. All 3D points that were used for the whole frame sequences are found in a scattered fashion to provide a visual illustration. Most of the 3D points with the highest density are located in the entrance of the tunnel, right before entering the canopy part, up to the far side of the hill. This shows how far our back-projection method finds the 3D point within the cloud. However, a lack of 3D points is found right inside the canopy part of the tunnel. This was greatly affected by the feature matching's capability which rejects low quality features and leaves us with very few. The back-projection hit rate which is relatively low also affected this even further, given us even fewer 3D points from the cloud.

6.4 Drift Distance

The lateral drift distance in the figure takes the absolute distance difference between each direction's drift, $|X - Y|$, while altitude drift distance is a combination of $|Y - Z|$ axes. Further analysis comparing both lateral and altitude drift distance refers to the Figure 6.4.

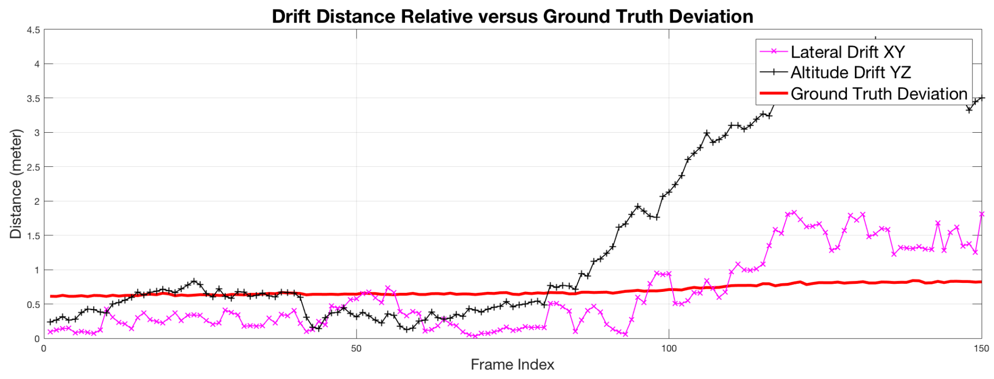


Figure 6.4: The drift distance both laterally and in the vehicle's altitude through all the frames.

Figure 6.4 describe also the quality of the experimental simulation if the solution is safely under the deviation distance. Our solution, especially in lateral trajectory has been mostly performing better than the GPS before frame 100 which marks the deeper part of the tunnel.

Overall, both lateral and altitude drift started from a well respective 30 centimeter from the reference GPS position. It is recorded right on the first frame after the camera solved its initial pose. The numbers are consistent until frame 80 when, based on Figure 6.5, the number of both feature points and succesful back-projected points dramatically decreased due to the lack of consistent landmark from the feature detection. The increase in altitude drift is greater than the lateral since the position of the available features are mostly above the camera center, for example the tunnel ceiling, and the less consistent features are found in the road.

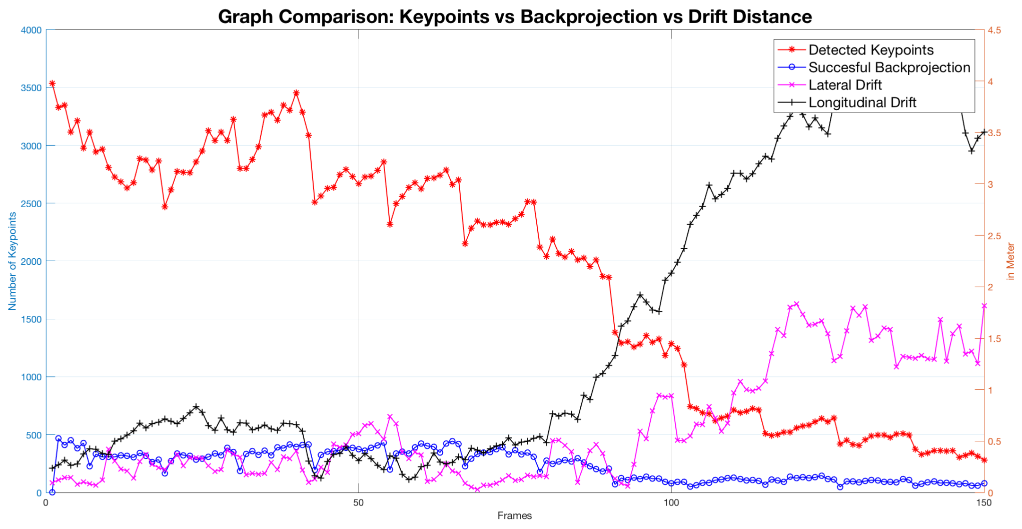


Figure 6.5: The drift distance both laterally and in the vehicle's altitude through all the frames as compared to the ground truth deviation.

The back-projection, while it builds a ray which traverse across each feature points, most of the rays did not intersect with any single 3D point inside the tunnel cloud. It gives a reasonably low succesful back-projection rate.

The drift distances depicted in Figure 6.4 are made more clear in Table 6.2 below. Most imporantly, it shows the maximum drift distance that the algorithm produced, which happens as we travel deeper into the tunnel, for the reasons explained earlier and discussed in 7.1.

Direction	**Average**	**Maximum**	**Minimum**
Lateral	0.633	1.833	0.030
Altitude	1.602	4.354	0.125

Table 6.2: Average, maximum and minimum drift distance for lateral (x,y) and altitude (z), all numbers in meters.

7 Discussion and Future Work

The project started out with the idea that autonomous cars need to be able to position themselves, even when there is no GPS signal available. Image analysis of the images from the vehicle-mounted camera quickly became the strongest candidate solution because of promising previous research and general applicability to other areas.

7.1 Discussion

There has been substantial research efforts in the area of computer vision the last decade, and not any less so with the purpose of building positioning methods on top of computer vision. It has been shown earlier that it can be done, and our results agree fully with this.

We have succeeded in finding a solution able to do this, with the restriction that it needs good images and a known starting point. These restrictions were however known beforehand, as it is clear that images without unique features are not able to help a positioning algorithm. We knew from early on that there would be certain difficulties, both because of physical reasons related to the camera and its output, but also because of logical reasons as we were both new to the area when starting the project.

As an example of the former, we found that the most prominent source of error has been the relatively low quality of the images, mainly due to degradation when converting the video, but also because of the physical properties of the camera unit itself. A concrete example of this is the lack of good calibration data for the camera. Unless you know information such as the coordinates of the principal point, which we have simplistically specified as the middle pixel of the image, there will be errors in the camera pose and thus also in the back-projection. The back-projection would be improved with better specifications of the camera's calibration data. Every keypoint in the image plane has to be transformed into world frame coordinates and incorrect camera parameters will greatly affect the accuracy of the rays. As the calibration is a crucial step for the rest of the solution, we recommend for future work that this is definitely known if the camera module is not available for a manual calibration.

The original video was known from early on to be relatively dark inside the tunnels. Also, their quality was not enough to find enough structural information in areas such as walls, road and ceiling to retrieve stable, unique features from them. The images we have been working on has been suffering from degradation in quality due to the conversion from a proprietary video format to publicly usable image format. With this in mind, we believe our achieved results and the corresponding error as given in Table 6.2 is reasonable. It has been shown that the technique works but still has much room for improvements. For the product to be used in production, improvements would indeed be needed as the accuracy is not quite good enough yet.

Another topic worth discussing is how we validate the result. The GPS is the most common technology used to generate a metric position for vehicles and is very often used as the main reference, or ground truth, for other solutions in related research. For this thesis however, the lack of an accurate GPS position means that finding a solid ground truth was not feasible. As a result of this, it is not possible to perfectly validate our solution. We had available a number of post-processed GPS coordinates for vehicles inside the tunnel, but these were unfortunately not the GPS coordinates of the vehicles capturing the video we used which led to an obvious limitation in our validation. Thus, finding the GPS position for each frame is not possible and our validation required an alternate method.

7.2 Future work

From an implementation point of view, there are several obvious future improvements. One of those is to remove the requirement that there are manual correspondences already existing before running the algorithm. While it is considered as a very impractical task, the manual correspondences is one way to associate the 3D points of the tunnel with the corresponding 2D points. It is not a trivial task to to find an alternate solution however, as the solution depends on an initial camera pose calculated from corresponding 2D and 3D points. One trivial way to make the manual correspondences better is to simply increase the vocabulary size containing the features of the tunnel, which can then be reused every time a vehicle is driving towards the tunnel and recognize the saved features in the vocabulary. Object categorization [35] and annotation [36] are examples of current research on classifying an object, which in this case an entrance to the tunnel, using feature selection.

More thorough validation of the results is another important future work, and can be done simply by running our solution on a new video feed which was taken while also recording the GPS coordinates, or any other positioning method considered robust enough. A better camera with a better sensor that works in relatively dark areas could provide substantial improvements in providing reliable features. However, a more robust way to detect features that are consistent throughout image sequences is also an important aspect to consider, since inside tunnels it is a completely different scenario. SIFT, while it is proven to work well, performance might be an issue for anyone who wanting to work with online and real-time positioning. Changing the feature detection and extraction algorithms also requires changing how we match images with the manual correspondences, since the previous correspondences would become obsolete.

Lastly, combining sensors such as the wheel sensors and the IMU to estimate the vehicle position based on previously known GPS coordinates, with our solution as a visual assistance to verify the estimation using non-linear state estimations, for example using a Kalman Filter, could be a potential future work to explore.

8
Conclusions

In this thesis we have presented an alternative to GPS for solving the vehicle positioning inside tunnels given the camera output and 3D model of the tunnel. The solution does not depend on other sensors, e.g., IMU or LIDAR, hence the scope of the project is entirely based on 2D-to-3D correspondences between images and the point cloud. While the ambition of this thesis was to produce an accurate vehicle positioning for a given scenario, multiple challenges and limitations had emerged over time. Thus, as the thesis concluded, it had changed to being able to solve the problem under certain conditions.

One of the substantial steps of this project was to incorporate the point cloud into the camera pose estimation problem. The manual correspondences method has proved to be one way to determine the critical initial camera pose. Picking world points in the point cloud for a chosen 2D feature is doable for a set of correspondences, but requires fine-tuning. A fine-tuning when picking combinations of stable and well spread out initial correspondences led to a significant increase in accuracy, not only for the initial camera pose, but for the whole sequence of processed images.

The general finding concerning the limitations of the solution, as described in Results, is the difficulty of sustaining a consistently high enough amount of back-projected features once the vehicle moves inside the darker areas of the tunnel. Increasing the sensitivity of the feature detector, or even changing the method to use another feature detector algorithm could be potential workarounds for this. Also, further work could potentially improve the whole solution by exploiting different parameters of feature detection, extraction and also, feature matching.

In the end, utilising the images from a camera for an alternative positioning method inside tunnels where we lack GPS data is a potential solution with some caveats. This thesis partially provided accurate vehicle positioning, competing with the preprocessed data given an accurate point cloud that uses a metric world GPS reference. On average, our solution gives lateral drift distances below one meter and, while there is good lighting, a minimum of below one decimeter can be achieved.

Bibliography

[1] Petridou, Eleni, and Moustaki Maria. "Human Factors in the Causation of Road Traffic Crashes." European Journal of Epidemiology 16.9 (2000): 819-26. Web.

[2] National Highway Traffic Safety Administration. (2015). "Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey". http://www-nrd.nhtsa.dot.gov

[3] P. Puricer and P. Kovar, "Technical Limitations of GNSS Receivers in Indoor Positioning", 2007 17th International Conference Radioelektronika, Brno, 2007, pp. 1-5.

[4] The Drive Me Project. Retrieved from http://www.volvocars.com/intl/about/our-innovation-brands/ intellisafe/intellisafe-autopilot/drive-me

[5] Lopez, A.; Canero, C.; Serrat, J.; Saludes, J.; Lumbreras, F.; Graf, T., "Detection of lane markings based on ridgeness and RANSAC," in Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE , vol., no., pp.254-259, 13-15 Sept. 2005

[6] Jaramillo, C., Dryanovski, I., Valenti, R. G., & Xiao, J. (2013). 6-DoF pose localization in 3D point-cloud dense maps using a monocular camera. Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on. IEEE

[7] D. Nister, O. Naroditsky and J. Bergen, "Visual odometry," Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on, 2004, pp. I-652-I-659 Vol.1.

[8] Scaramuzza, D and Fraundorfer, F. "Visual Odometry [Tutorial]," in IEEE Robotics & Automation Magazine, vol. 18, no. 4, pp. 80-92, Dec. 2011.

[9] Mat Zucker, "Cameras & Camera Geometry." http://www.swarthmore.edu/NatSci/mzucker1/e27/camera-slides.pdf

[10] R. Hartley, A. Zisserman, "Multiple view geometry in computer vision.", Cambridge University press, 2003.

[11] Etay Meiri, "OGL dev Modern OpenGL Tutorials." http://ogldev.atspace.co.uk/www/tutorial13/tutorial13.html

[12] C. Olsson, F. Kahl and M. Oskarsson, "Optimal Estimation of Perspective Camera Pose," Pattern Recognition, 2006. ICPR 2006. 18th International Conference on, Hong Kong, 2006, pp. 5-8. doi: 10.1109/ICPR.2006.909, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1699135&isnumber=35818

[13] Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang and Hang-Fei Cheng, "Complete solution classification for the perspective-three-point problem," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 8, pp. 930-943, Aug. 2003.

[14] Martin A. Fischler and Robert C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". Commun. 1981. ACM 24, 6 (June 1981), 381-395.

[15] Kneip, Laurent, Davide Scaramuzza, and Roland Siegwart. "A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation." Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on. IEEE, 2011.

[16] Wandinger, U. "Introduction to lidar, in LIDAR—Range-Resolved Optical Remote Sensing of the Atmosphere". Edited by C. Weitkamp. pp. 1–18, Springer, New York, 2005.

[17] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," Robotics and Automation (ICRA), 2011 IEEE International Conference on, Shanghai, 2011, pp. 1-4.

[18] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In Proceedings of the 9th European conference on Computer Vision - Volume Part I (ECCV'06), Aleš Leonardis, Horst Bischof, and Axel Pinz (Eds.), 2006, Vol. Part I. Springer-Verlag, Berlin, Heidelberg, 430-443.

[19] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. 2008. Speeded-Up Robust Features (SURF). Comput. Vis. Image Underst. 110, 3 (June 2008), 346-359.

[20] D. G. Lowe. "Distinctive image features from scale-invariant keypoints". International journal of computer vision, 60(2), 2004, pp 91-110.

[21] O. Miksik and K. Mikolajczyk, "Evaluation of local detectors and descriptors for fast feature matching," Pattern Recognition (ICPR), 2012 21st International Conference on, Tsukuba, 2012, pp. 2681-2684.

[22] D. G. Lowe, "Object recognition from local scale-invariant features," Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on, Kerkyra, 1999, pp. 1150-1157 vol.2.

[23] Derpanis, Konstantinos G. "Overview of the RANSAC Algorithm." Image Rochester NY 4 (2010): 2-3.

[24] Y. Zheng, Y. Kuang, S. Sugimoto, K. Åström and M. Okutomi, "Revisiting the PnP Problem: A Fast, General and Optimal Solution," 2013 IEEE International Conference on Computer Vision, Sydney, NSW, 2013, pp. 2344-2351.

[25] Manolis I. A. Lourakis and Antonis A. Argyros. 2009. "SBA: A software package for generic Sparse Bundle Adjustment". ACM Trans. Math. Softw. 36, 1, Article 2 (March 2009), 30 pages. http://users.ics.forth.gr/ lourakis/sba/

[26] Davison, Andrew J. 2003. Real-time simultaneous localisation and mapping with a single camera. Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on. IEEE, October 13

[27] Scaramuzza, D., Fraundorfer, F., & Siegwart, R. (2009). Real-time monocular visual odometry for on-road vehicles with 1-point ransac. Robotics and Automation, 2009. ICRA'09. IEEE International Conference on. IEEE

[28] Lee, G., Faundorfer, F., & Pollefeys, M. (2013). Motion estimation for self-driving cars with a generalized camera. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition

[29] Svarm, Linus, Olof Enqvist, Magnus Oskarsson, and Fredrik Kahl. 2014. Accurate localization and pose estimation for large 3d models. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

[30] Itseez, "Itseez - Vision that works," http://itseez.com/

[31] V. Lepetit, F. Moreno-Noguer and P. Fua. EPnP: An Accurate O(n) Solution to the PnP Problem, in International Journal Of Computer Vision, vol. 81, p. 155-166, 2009.

[32] N. Sünderhauf, K. Konolige, S. Lacroix, P. Protzel. "Visual Odometry Using Sparse Bundle Adjustment on an Autonomous Outdoor Vehicle". Autonome Mobile Systeme: Fachgespräch Stuttgart. p157-163. December 2005.

[33] H. Lim, J. Lim and H. J. Kim, "Real-time 6-DOF monocular visual SLAM in a large-scale environment," 2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, 2014, pp. 1532-1539

[34] Weisstein, Eric W. "Point-Line Distance–3-Dimensional." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Point-LineDistance3-Dimensional.html

[35] Pinz, Axel. "Object categorization." Foundations and Trends in Computer Graphics and Vision 1.4 (2005): 255-353.

[36] Hanbury, Allan. "A survey of methods for image annotation." Journal of Visual Languages & Computing 19.5 (2008): 617-627.

[37] Bob Fisher, "3x4 Projection Matrix" From Geometric Framework for Vision I. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/EPSRC_SSAZ/epsrc_ssaz.html