Computer Vision-based Navigation for an Autonomous Utility Vehicle

Evaluation of the feasibility in a computer vision-based visual SLAM method for autonomous utility vehicle navigation in urban environments

Master's thesis in System Control and Mechatronics

NIPUN C. GAMMANAGE
FANGZE XU

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Computer Vision-based Navigation for an Autonomous Utility Vehicle

Evaluation of the feasibility in a computer vision-based visual SLAM method for autonomous utility vehicle navigation in urban environments

NIPUN C. GAMMANAGE
FANGZE XU

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Automation research group
Chalmers University of Technology
Gothenburg, Sweden 2020

Applied Computer Vision-based Navigation for Utility Vehicles in an Urban Environment
Evaluation of the feasibility in a computer vision-based visual SLAM method for autonomous utility vehicle navigation in urban environments

NIPUN C. GAMMANAGE
FANGZE XU

Supervisor: Damir Džebo, Semcon Sweden AB
Supervisor: Anton Sediako, Semcon Sweden AB
Examiner: Martin Fabian, Chalmers University of Technology

Master's Thesis 2020
Department of Electrical Engineering
Division of Systems and Control
Automation research group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A 3D Point cloud visualization which was reconstructed using stereo vision geometric techniques.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Applied Computer Vision-based Navigation for Utility Vehicles in an Urban Environment
Evaluation of the feasibility in implementing OpenVSLAM for urban navigation purposes
Nipun C. Gammanage
Fangze Xu
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Autonomous vehicle technology has grown and developed rapidly in recent decades. One of the challenges in autonomous navigation is estimating the current location in terms of an absolute position and a heading direction. A widely used method for vehicular position estimation is GNSS (Global Navigation Satellite System). However, since the GNSS signal's accuracy depends heavily on the environment's structural behavior and the weather conditions, new research is looking for alternative methods to support GNSS-based navigation. The goal of this thesis was to implement and evaluate the feasibility of using a computer vision-based navigation method as a substitute for GNSS-based navigation. The vehicle's position and direction were estimated using a state-of-the-art visual SLAM method called OpenVSALM based on stereo vision. Furthermore, the system was evaluated against two datasets: the benchmark KITTI odometry dataset and a real scene dataset captured by the on-board stereo camera rig of the project vehicle addressed in this thesis. Feature extraction and tracking are the main components in any visual SLAM method. Hence, a comprehensive study was also conducted to evaluate the quality of feature detection and tracking based on the camera's field of view (FoV). Experiments show that the FoV reduction exponentially increases the error in pose estimation (relative translation and rotation). Errors were evaluated using two error criteria, the Root Mean Square Error, and the Absolute Trajectory Error.

Keywords: Autonomous navigation, Computer vision, Pose estimation, Scene Mapping, Visual Odometry, SLAM, Visual SLAM, OpenVSLAM, Free space detection

Acknowledgements

First and foremost, we thank Semcon AB for selecting us for this thesis work and providing all the necessary resources required to complete this thesis work. It gave us an opportunity to apply the knowledge we learned in our master's and prepared ourselves for real-world engineering applications, and for that, we are truly grateful. We have received enormous support from many people for the past eight months, especially our supervisors Damir Džebo and Anton Sediako and our team manager Mats Larsson at Semcon AB. We are very grateful for all technical supports as well as insightful guidance towards our work. We also express our gratitude to our examiner at the Chalmers University of Technology, Professor Martin Fabian, who provided enormous academic support for the past months, which greatly enlightened us to complete this thesis work. Finally, we thank our families and friends who encouraged us to overcome all difficulties we encountered in this thesis journey.

Fangze Xu & Nipun Gammanage, Gothenburg, September 2020

Contents

List of Figures

List of Tables

1

Introduction

Autonomous vehicles (AV) are one of the most discussed topics in the engineering community [1]. Many different approaches have been investigated to achieve autonomous navigation [2], which can be considered one of the largest subsets in AV technology. Computer vision is one of those approaches that tend to have more popularity over the other methods in recent years. In this thesis, one such computer vision-based navigation support function will be evaluated. This introductory chapter contains a background, which describes the project more broadly. The project's objective is then presented in which the primary purpose of this project is discussed, followed by a section on related work. Next, the research questions that are addressed in this thesis work are discussed along with the limitations. The thesis contributions are followed by a brief introduction to the system settings. The chapter ends with a section about the thesis outline, which provides a summary of the remaining chapters and their content.

1.1 Background

According to recent statistics on urban transport [3], there is a huge trend in using either cycling or walking as the mode of transportation in urban areas. As a result, cyclists and pedestrians today make up the largest proportion of traffic injuries. Recent research [4] published by the European Transport Safety Council (ETSC) shows that in 2019, the number of fatalities due to traffic accidents was 22659 in Europe. One significant factor in these accidents is bad road conditions. Traffic accidents occur more frequently in the winter months. Therefore, it is vital to keep good road maintenance during winter. Especially in the Scandinavian countries, since the weather condition is harsh in winter. Semcon Sweden AB, in collaboration with the Research Institutes of Sweden (RISE), develops a concept solution of an autonomous utility vehicle for road maintenance, specifically for urban cycle/pedestrian paths. This thesis is conducted as a part of the collaboration, and the project is named as, *Barmark*.

Autonomous vehicle technology has grown and developed rapidly in recent decades. However, research regarding the viability of vehicles that can operate autonomously in complex environments such as highways, roads, and crowded urban streets is not an entirely new topic [5]. One main concern in AV is the safety factor since people's lives are one of the highest priorities, no pedestrians or cyclists should be harmed by these vehicles, thus autonomous navigation needs to be done not only in a precise and accurate manner but should also be able to respond to events and obstructions that may be encountered. A comprehensive review from 2010 [6] claims that ensuring the vehicle, passengers, and

the pedestrian's safety at all instances regardless of having several sensor technologies, will be a key challenge, especially in urban environments. After one decade of exponential growth in the field of AVs, one can see that this issue is yet to be solved. In the field of AVs, "safety" generally refers to the likelihood of being involved in a crash, vehicular, or otherwise [7].

In terms of safety in the autonomous navigation context, it is important to know the vehicle's current state, which contains the vehicle's position and heading. This thesis focuses on addressing the issue of accurately estimating the current position as well as the heading direction of the vehicle using a computer vision-based approach.

1.2 Objective

As mentioned in the previous section, safety being one of the main concerns in AVs, it is important to accurately estimate the vehicle position and heading in the real world. In this thesis, the vehicular heading is defined using the orientation of the vehicle. This estimation of the translational movements and orientations are often called pose estimation in literature. Similarly, in this thesis, when referred to the term *pose* it would mean both the translational movement and the vehicle's orientation in a relative coordinate system. The translational estimation is given as X, Y, and Z coordinates in meters. The orientation terms include rotations around each of those X, Y, Z axis.

Different sensor techniques have been used to estimate the pose of a vehicle, namely GNSS, LiDAR, Radar, or ultrasonic sensors. Despite the popularity of these techniques, they are not infallible, and each has their specific limitations. For example, GNSS (Global Navigation Satellite System), is widely used in the engineering community for navigation tasks, however, as shown in Figure 1.1, the accuracy of the GPS signal depends heavily on the environment and the weather conditions, which means that the system is sensitive to structural changes in the environment, therefore, it cannot guarantee good accuracy especially when navigating in an urban environment. In the context of structural behaviors in the environment, estimations from ultrasonic sensors and Radars have the same effects when dealing with reflective surfaces. Apart from that, Radars and ultrasonic sensors need extra sensor values, like the wheel odometry from the vehicle, to get accurate estimations for the long-range. LiDAR, on the other hand, provides very accurate 3D information of the environment but is slow to get accurate measurements and still relies on very simple scan-matching approaches that are not very robust.

Therefore, to find methods that have fewer effects on the accuracy (in terms of pose estimation) from the environment's structural behavior is essential. Thus, in this thesis, an objective was set to investigate the feasibility of a computer vision-based approach for pose estimation. A state-of-the-art visual SLAM method was evaluated in this thesis. In research conducted around visual SLAM, researchers typically use high-end hardware or near-ideal datasets, implying that the absolute efficiency that researchers claim their models achieve is tied to the high-end hardware and ideal datasets. In this thesis, however, the project is conducted with real-world data with moderate hardware specifications.

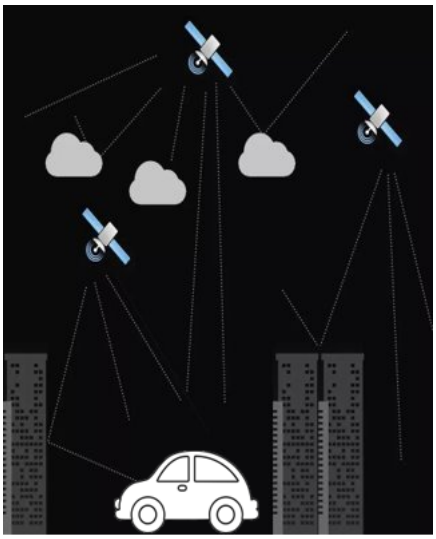


Figure 1.1: The structural effect from the surroundings on GNSS estimations. The GNSS signal might reflect when it comes to different structures in the environment. Weather conditions could also have impact on the reflections.

1.3 Related work

As mentioned in the introductory chapter, the main task of this thesis is pose estimation, since we have depth data, free space detection is also implemented as an extra function. An extensive literature review regarding these tasks was conducted and an overview of the related work that has been done in recent years is presented in this section.

1.3.1 Vehicle Navigation and SLAM

A study of the stereo vision-based navigation and monocular vision-based navigation is conducted, as the vehicle is equipped with stereo cameras, the study will mainly focus on the stereo vision-based navigation methods.

1.3.1.1 Monocular vision-based navigation

A pose estimation method based on road signs using only a monocular camera fused with GPS data was proposed by [8]. The authors of this study claim this method can achieve good accuracy, within the range of 100 meters distance to the road signs, the pose error is limited within 2 degrees, and the position error is less than 1 meter.

Unlike typical vision-based methods that rely on the integration of different sensors, which would result in unstability of the implementation when the baseline is small, [9] provides a novel visual localization method. A visual street map and extracted *SURF* (Speeded Up Robust Features) image features are used in this method, by monitoring the difference in scale of features matched between input images and the visual street map within a Dynamic Time Warping framework, accurate vehicle positioning is then achieved.

1.3.1.2 Stereo vision-based navigation

A method for real-time localization using a stereo vision system along with inexpensive GPS data was suggested by [10]. Their system uses inertial measurements to fill in motion estimates when visual odometry fails. This incremental motion is then fused with a low-cost GPS sensor using a Kalman Filter to prevent long-term drifts. The system shows great robustness and high accuracy in real time pose estimation.

A Graph-based SLAM tutorial is introduced in [11], in which a method is able to create a map of the unknown environment is proposed, this state-of-the -art solution is based on least-squares error optimization. A graph is used in which vehicle poses during the movement at different time instances are represented by the nodes, and constraints between the poses are represented by the edges. A map can hence be obtained once the graph is built, which can be achieved by finding the spatial configuration of the nodes that is mostly consistent with the measurements modeled by the edges.

The paper [12] provides a solution to the SLAM problem. A proof is given that the estimated map converges monotonically to a relative map with zero uncertainty, combined with the result that the absolute accuracy of the map and the vehicle location reach a lower bound only defined by the initial vehicle uncertainty. This shows the possibility to build a perfect map in an unknown environment to solve the SLAM problem by only using relative observations. A number of key issues appeared in the SLAM problem such as suboptimal map-building algorithms and map management are also discussed in this paper.

In [13], the SLAM problem is seen as least-squares optimization of an error function. A general structure of SLAM and an open-source C++ framework called g^2o are presented in this paper, which aims at optimizing graph-based nonlinear error functions. It is said that the proposed system can be easily extended to a wide range of problems and can achieve good performance.

Both of the papers from M. W.M. Gamini Dissanayake et. al [12], and Hugh Durrant-Whyte et. al [14] provide a probabilistic form of the SLAM problem as well as essential solutions and significant implementations. The key to solve the probabilistic SLAM problem is finding a good representation of the observation and motion models that allow to efficiently calculate the prior and posterior distributions. Among various representations, an Extended Kalman filter is used to solve the SLAM problem in a form of a state space model with additive noise that follow Gaussian distribution, while a Rao-Blackwellised particle filter, also called Fast SLAM algorithm, is often used when the vehicle motion model is described as a set of samples of a more general non Gaussian probability distribution.

1.3.2 Free space detection

As free space detection is treated as a low priority task, the method used in this thesis is inspired by [15], where salient object detection algorithm is developed to detect objects by using a depth map. However, the implementation of this algorithm in this thesis work is modified, unlike the implementation of traditional object detection algorithm, generating bounding boxes of the detected objects is not required in this thesis.

1.4 Thesis Contribution

In this thesis, the authors will try to answer the following research question by implementing a computer vision-based pose estimation system and evaluating the system with *image data* captured through a utility vehicle.

1. Is it feasible to use a stand-alone computer vision-based pose estimation method in a complex urban environment?
2. Given different methods to estimate the vehicle direction, what would be the recommended method in terms of accuracy?
3. What aspects of the hardware plays the most critical role in pose estimation accuracy?
4. Why are certain hardware requirements critical in computer vision-based pose estimation?
5. Will sudden scenery changes affect the pose estimation?
6. What would be an ideal setup for better pose estimations?

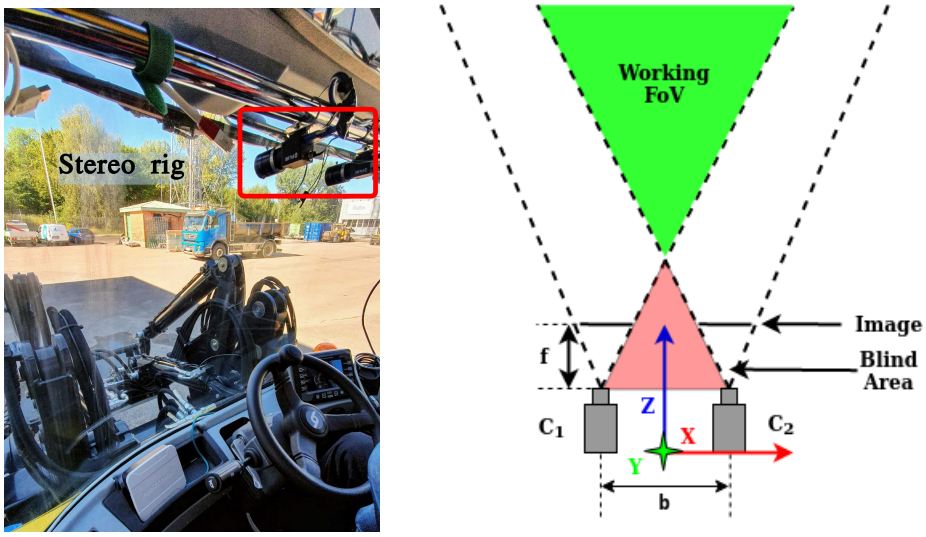
1.5 Limitations

The project scope will be restricted to the following limitations.

- The dataset that contains the images from the *Barmark* project is collected in real-time. However, the data are recorded as compressed images, and the implemented system will only be tested in an off-line separate system.
- The two cameras in the stereo rig are approximately parallel. In other words, the 3 axes of the two cameras have a negligible amount of relative rotations with respect to each other.
- Weather condition effects will not be addressed in this thesis.
- The project will not include sensor fusion with either Lidar, radar, infrared or ultrasonic devices. However, the conversion between GPS coordinates and Cartesian coordinates will be implemented.
- Vehicle dynamics will be ignored. The vehicle used in this project is an articulated vehicle. Therefore, the position of a sensor estimating the pose (regardless if it is a GNSS unit or an IMU) would need extra mathematical adjustments to obtain the vehicle's position. However, this mathematical adjustment for vehicle dynamics is already implemented and tested by the Semcon team. Hence, in this thesis, such mathematical adjustments will not be performed on the estimated vision pose; instead, the Semcon team's method will be used directly on the vison pose to adjust the pose accordingly.
- In this thesis, the main concern is to detect the objects on the driving path, no matter whether the detected objects are stationary or moving, potential collision should be prevented. Classification of the detected objects on the path will not be carried out, as any detected objects, whether animals, cars, people, should not be facing collision with the vehicle.

1.6 System configuration

This thesis used two different datasets to evaluate the feasibility of a vision system being used in an urban navigation system. The dataset from this project is called *BARMARK* dataset. It contains compressed images from the left and right cameras, captured in real-time over a period of approximately 120 s. These images have a resolution of 960×600 and field of view (FoV) of around 23°. The total area covered in this dataset is approximately around $30\,m^2$. A standard benchmark dataset *KITTI* [16] was also used to evaluate a hypothesis in this thesis. The KITTI dataset contains images with a resolution of 1220×370 and has an FoV close to 82°. The total area covered in this dataset is close to $250m^2$. The main hardware setup that is used in this thesis is shown in Figure 1.2.



(a) **(b)**

Figure 1.2: System configuration. Figure (a) illustrates how the two cameras are attached in the vehicle, and figure (b) illustrates a 2D view of how the two cameras see the scene and its camera coordinate system.

1.7 Thesis outline

This thesis is divided into five chapters. It starts with this introductory chapter, followed by Chapter 2, where fundamental theories involved in this thesis work are presented. Next, Chapter 3 discusses the system implementation and the evaluation methods related to this thesis work. In Chapter 4, the analysis of the outcome of the implementations is presented. Finally, in Chapter 5, a summary of the study is presented on the project outcomes based on the research questions presented in Section 1.4.

2

Theoretical Background

Estimating an agent's current location, given camera input, fundamentally requires estimating the camera's pose. The estimated camera pose will then have to be updated over time to create an accurate vehicle movement trajectory. In this thesis, an open-source state-of-the-art visual SLAM algorithm called *OpenVSLAM* [17] is used to achieve the above task. This process, starting from obtaining the images until estimating the *GPS* location, relies on four main parts, namely accurate estimation of the depth of the scene, feature extraction (and tracking), final pose estimation, and the optimization of the estimation. This chapter illustrates the theoretical fundamentals of obtaining reliable location estimations. In the first section, the geometrical overview of stereo vision will be discussed. Next, essential feature extraction and tracking factors will be presented, followed by an overview of *Visual odometry* and *SLAM*.

2.1 Stereo vision

The concept of *stereo vision* is widely adopted in computer vision applications, especially when the depth perspective of a scene is required in an application. The idea suggests to use two cameras to observe a scene and estimate the depth of that scene using only the geometric characteristics in the two images, which is also known as the *epipolar geomatry* (see Section 2.1.3).

This estimation starts with computing *correspondences* between two image planes. These correspondences along with the distance between the two image centers (also known as the *base-line* of the *stereo rig*) one would be able to calculate the *3D* location or the depth of a point in the real world. The entire depth estimation method of a scene, also known as the *Stereo imaging*, mainly consists of four different steps [18].

1. Lens distortion (radial and tangential) removal. As described in Section 2.1.1, the pinhole camera model does not work in the real world as the light exposure process is slow. To avoid this, one would need to add external lenses. This, however, introduces distortions in the resulting image.
2. The *rectification* process, described in the Section 2.1.3.1 ensures that the two images are *row-aligned*.
3. Construct the *disparity map* by matching the features in the left and right images, also known as finding the *stereo correspondences* (Section 2.1.3.2).
4. Use *triangulation* to estimate the depth of the image. This step is known as the *reprojection* step and will be discussed in Section 2.1.3.3.

2.1.1 Camera model

The camera model used in this project is known as the pinhole camera model, which is also the most common used model in the computer vision field. The image points are projected on a plane through a microscopic pinhole, a camera intrinsic matrix K is defined (2.1) to record the information of focal length, aspect ratio, principal point, and skew. Theoretically, a lens can be defined without introducing distortion, however, in practice, avoiding lens distortion is impossible due to the fact that no lens could be manufactured perfectly. There are two kinds of main distortions that exist in a camera model, which are radial distortions, and tangential distortions. The radial distortion is a result of the shape of the lens, while the tangential distortion is often caused by the manufacturing process.

$$K = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

The radial distortion model can be expressed by the following equations

$$x_{corrected} = x\,(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.2}$$

$$y_{corrected} = y\,(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.3}$$

The tangential distortion model can be expressed by the following equations.

$$x_{corrected} = x + [2p_1 y + p_2(r^2 + 2x^2)] \tag{2.4}$$

$$y_{corrected} = y + [p_1(r^2 + 2y^2)2p_2 x] \tag{2.5}$$

Follow the OpenCV rountines, the distortion parameters are then introduced into one distortion vector $[k_1\ k_2\ p_1\ p_2\ k_3\ ]$, where k_1, k_2 and k_3 are terms for radial distortion, while p_1 and p_2 are terms for tangential distortion.

2.1.2 Camera calibration

Critical information that is needed in depth estimation is the cameras' focal length. Furthermore, in the VSLAM algorithm, knowing the intrinsic matrix given in (2.1) is essential. The method of estimating these parameters is often called *geometric camera calibration.* The matrix K gives the relationship between the real world and the image plane,

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \underbrace{K \begin{bmatrix} R & t \end{bmatrix}}_{:=P} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.6}$$

Here the intrinsic matrix K represents the projective transformation (like a homography matrix) between the 3D camera coordinates (X_c, Y_c, Z_c) to the image coordinates (the pixel x, y coordinates), while the extrinsic matrix $\begin{bmatrix} R & t \end{bmatrix}$ transforms the world coordinates X, Y, Z to the camera coordinates of X_c, Y_c, Z_c. In *pose estimation* algorithms, what is often estimated is the camera matrix P, and multiplying the newly estimated P with the inverse of K would give the relative pose (the $\begin{bmatrix} R & t \end{bmatrix}$ matrix) of the camera.

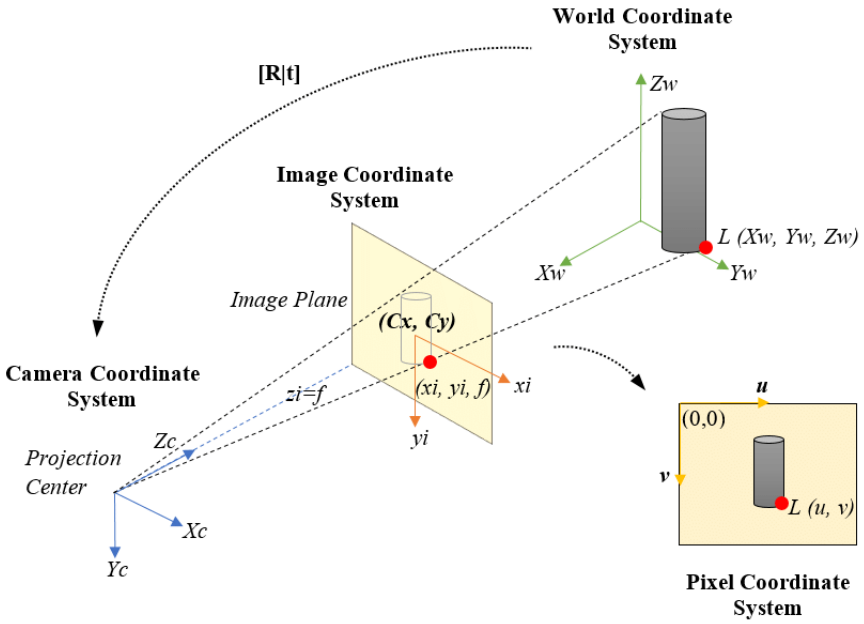


Figure 2.1: Pinhole camera model [19].

2.1.2.1 Direct Linear Transformation Method

The standard method that could be used in camera calibration is *Direct Linear Transformation (DLT)*, which estimates the camera parameters given the 3D location (also known as the world coordinate) of a point and its corresponding image location, via the following expression,

$$x_{3\times1} = \underbrace{K_{3\times3} \cdot R_{3\times3} \begin{bmatrix} I & -X_0 \end{bmatrix}_{3\times4}}_{P} \cdot X_{4\times1} \tag{2.7}$$

In (2.7), the values $x_{3\times1}$ and $X_{4\times1}$ denote the image coordinates and its corresponding world coordinates. The value of P would be the camera matrix, which needs to be estimated. One could also see that the intrinsic parameters in the matrix K have 5 degrees of freedom (DoF) and the rotational matrix and the translation vector have 3 DoF each, hence summing up to 11 DoF in total for the system. Using geometric properties in the relationship given in (2.6), and methods such as Singular value decomposition (SVD) and RQ factorization, one would be able to estimate these values. Detailed explanations and mathematical proofs are given in Appendix A.1.

2.1.2.2 Zhang's Method

Obtaining the intrinsic parameters using *DLT* as mentioned above, will not work for the problem in this thesis, since acquiring the correct world coordinates is not practical. In the more practical scenarios to obtain the camera's correct intrinsic parameters, one could use *Zhang's Method* [20].

Zhang's Method is also based on the DLT, but with some additional steps. One should also note that this method will only focus on obtaining the intrinsic parameters and will

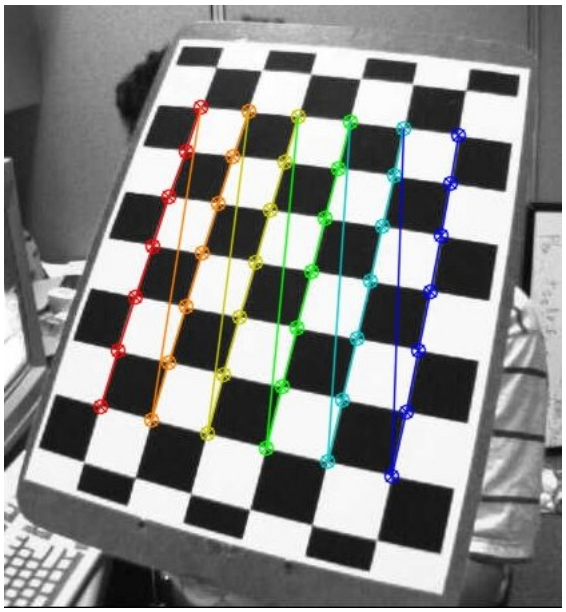


Figure 2.2: Checkerboard pattern. This unique pattern allows the feature detection algorithms to easily detect feature points since the gradient change (in intensity) from one square to another is significant. OpenCV [21] uses different colors for each row so that it would be easy to track the rotational changes efficiently. Dimensions of the checkerboard as per this example is 7×6.

not focus on obtaining the rotational or the translational matrices. The difference in this method is that instead of using known 3D locations, it uses a *checker-board* pattern under the assumption that the parameters (or the structure, width, height, and the number of squares) are known in the checkerboard, and the checkerboard is a planar surface. One main reason to use a checkerboard for this method apart from it being a planar surface is its unique pattern, which allows distinguishing feature points and tracking them in different images. An example checkerboard image is given in Figure 2.2.

The trick used in this method is that it defines its own coordinate system such that the x, y plane is defined on the checkerboard surface. The idea would be to force the feature points that are detected to have a Z coordinate value of zero, which results in a new *homography* matrix as shown in (2.8).

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{pmatrix} \left(\begin{array}{cccc} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{array} \right) \tag{2.8}$$

will be zero

Instead of directly estimating all the parameters of the camera matrix P, a series of *homography* matrices (as given in (2.8)) will be estimated. Each homography matrix will correspond to one image, and all these homography matrices will then be used to estimate the camera's intrinsic parameters. Mathematical explanation and derivation are given in Appendix A.2. Once the K matrix is estimated, the extrinsic parameters can be estimated via the method suggested in [22].

2.1.2.3 Stereo calibration

Here the term *stereo calibration* does not mean estimating intrinsic or extrinsic parameters, instead, it means estimating the transformation matrix P that describes the relationship between the two cameras. Firstly, the parameters of the left camera and the right camera need to be estimated separately via *Zhang's* method. These two matrices can be then defined as P_{left} and P_{right}. Consider a vector $X^{'} = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^T$, where a 3D location is denoted by $X = \begin{bmatrix} X & Y & Z \end{bmatrix}^T$ and its images locations (in the two image planes, left and right) as x_l and x_r.

$$\begin{aligned} x_r &= P_{right} X^{'} \\ &= \begin{bmatrix} R_{right} & t_{right} \end{bmatrix} X^{'} \\ &= R_{right} X + t_{right} \end{aligned} \tag{2.9}$$

$$\begin{aligned} x_l &= P_{left} X^{'} \\ &= \begin{bmatrix} R_{left} & t_{left} \end{bmatrix} X^{'} \\ &= R_{left} X + t_{left} \end{aligned} \tag{2.10}$$

The relationship between the two cameras can be defined as $x_r = R_{new} x_l + T_{new}$. Where R_{new} and T_{new} are the new rotational matrix and translational vector that defines the relationship between the two cameras. Combining (2.9) and (2.10) and re-arranging them,

$$R_{right}^{-1}(x_r - t_{right}) = R_{left}^{-1}(x_l - t_{left}) \tag{2.11a}$$

$$x_r = R_{right} R_{left}^{-1}(x_l - t_{left}) + t_{right} \tag{2.11b}$$

$$x_r = R_{right} R_{left}^{-1} x_l - R_{right} R_{left}^{-1} t_{left} + t_{right} \tag{2.11c}$$

$$x_r = \underbrace{R_{right} R_{left}^{-1}}_{R_{new}} x_l + \underbrace{t_{right} - R_{right} R_{left}^{-1} t_{left}}_{T_{new}} \tag{2.11d}$$

Since the inverse of a *rotational* matrix is its transpose, the matrix R_{left}^{-1} will become R_{left}^{T}. Then R_{new} and T_{new} can be re-written as,

$$R_{new} = R_{right} R_{left}^{T} \tag{2.12a}$$

$$T_{new} = t_{right} - R_{new} t_{left} \tag{2.12b}$$

2.1.3 Epipolar geometry

Depth estimation only using the image data is one of the key advantages in *stereo vision*. As explained in the previous section, depth estimation takes several steps, and one of the critical steps is finding the corresponding left and right image projection of the same 3D point. (this is often referred to as stereo correspondences, a detailed description is presented in Section 2.1.3.2). While there are various techniques to match images based on their local appearance, one could use the extra information available to search and define correspondences in stereo imaging. This geometrical technique of using additional

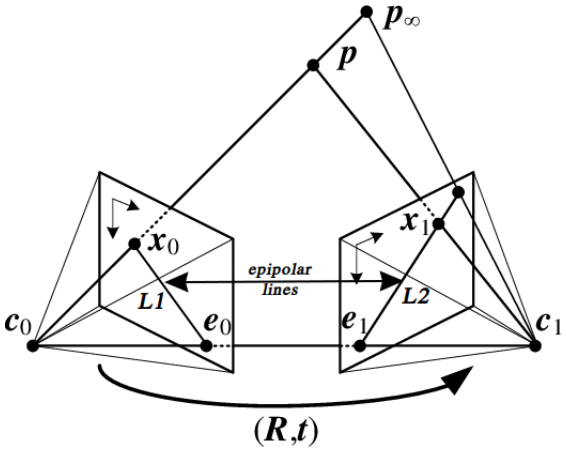


Figure 2.3: Epipolar geometry. The lines $L1$ and $L2$ represents the two epipolar lines in the images. The line $L2$ is a projection of ray c_0 to p_∞. The triangle connected by the vectors $(c_1 - c_0)$, $(p - c_0)$ and $(p - c_1)$ are co-planar and represents the epipolar plane [23].

information, the positions, and calibration data of the cameras that took the images of the same scene is called *Epipolar geometry.*

Consider two image planes capturing the same static point p, as illustrated in Figure 2.3. In this thesis, a similar geometry will be followed. The two images captured from the left and the right cameras, have a relationship between them that is given by a matrix P, where $P = \begin{bmatrix} R & t \end{bmatrix}$. This matrix P can be defined by the method given in Section 2.1.2.3. In Figure 2.3, p is a point that is located in the 3D world, and its projections to the two image planes are denoted as x_0 and x_1 (respectively on the left and right images). The two-camera centers are denoted as c_0, and c_1 and their projection on each other's image planes are denoted as e_0 and e_1 (right-center projection and left-center projection, respectively). These two special points are called *epipoles.* The co-planar surface formed by the two *epipoles* e_0, e_1 and the 3D point p is called the *epipolar plane*, and the line(s) defined by the points e_0x_0 (and e_1x_1) from the point of projection to the corresponding epipolar points are called the *epipolar lines.*

The projection of the 3D point p could lie anywhere on line $L2$ in the right image plane. An interesting feature that can be seen here is that when projecting the line $L2$ to the left image, it will become a point $(x0)$ in the left image plane. More specifically, all of the possible locations of a *point* seen in one image is the line that goes through the corresponding point and the epipolar point on the other image. This unique geometrical constraint is known as the *epipolar constraint* and reduces the traditional 2D feature search to a 1D feature search along the epipolar lines. Knowing the epipolar geometry of the stereo cameras not only saves computational power, it also allows the matching algorithms to reject points that could otherwise lead to spurious correspondences.

2.1.3.1 Stereo rectification

Once the epipolar geometry has been computed as described above, one can use the epipolar line corresponding to a pixel in one image to constrain the search for corresponding pixels in the other image, which would generate the *disparity* map (described in Sec-

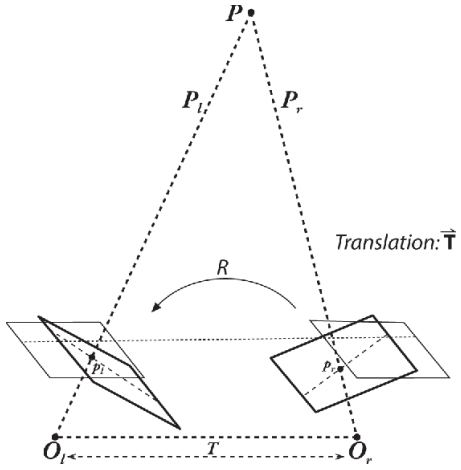


Figure 2.4: An interpretation of images plans in a real-world stereo rig. Projected image points of the 3D coordinate P are define as p_l and p_r. These two points need to be aligned to apply the *epipolar constraints*. However, this is not possible as the two image planes (illustrated as the two bold rectangles) have internal rotations [18].

tion 2.1.3.2). The given scene's depth map can then be estimated using the calculated disparity map, as described in Section 2.1.3.3. This process of epipolar geometry and searching through the point correspondence along the epipolar lines assumes that the two images are *row-aligned**. Unfortunately, in the real world one would not find perfectly aligned stereo rigs. An exaggerated but theoretically accurate stereo rig consisting of two un-aligned image planes is shown in Figure 2.4. The goal of stereo rectification is to align the two image planes mathematically so that the *epipolar geometry* can be applied during the process of *depth estimation*.

Several different algorithms are proposed in the literature to compute the rectification terms† for both calibrated [20], [24] and uncalibrated [25] cameras. However, since the calibration parameters are known in this section, only the calibrated version of stereo rectification will be discussed. This calibrated stereo rectification algorithm has often been called *Bouguet's algorithm*. It tries to estimate two rotational matrices R_l and R_r (left and the right cameras, respectively) that adjust the two cameras to be *co-planar* and *row-aligned* by minimizing the reprojection errors while maximizing the working area‡.

Recall the rotational matrix calculated in Section 2.1.2.3 that describes a rotational matrix which rotates the right camera's image plane into the left camera's image plane (ignore the translational part). Let R_{new} matrix in (2.12a) be split into two matrices r_l

*The term *row-aligned* means that the two image planes (left and right) are co-planar and that the *y-coordinates of the image* have the same values in the same direction (or all the image rows of both images are aligned)

†Here *rectification terms* means computing the distortion vector, a rotational matrix R_{rect} and the rectified and un-rectified camera matrices (M_{rect}, M). These rectification terms are unique for the OpenCV library that was used in this thesis, and a total number of 6 matrices and two vectors (for both left and right images) will be calculated in this stereo rectification process.

‡Here the term working area refers to the illustration in Figure 1.2b. In this figure, the green color zone is often in the literature called the common area between the two cameras.

and r_r, the the two matrices R_l and R_r can be define as,

$$R_l = R_{rect} r_l \tag{2.13a}$$

$$R_l = R_{rect} r_r \tag{2.13b}$$

The technique in *Bouguet's algorithm* is to first rotate each rotational part of the camera matrices (say R_{left} and R_{right}) until two principal rays (refer to Figure 2.5) of the cameras are parallel to each other. This would make the two image planes coplanar but not row-aligned. In the ideal case, as shown as the light rectangles in Figure 2.4, the two epipoles (e_0 and e_1 according to Figure 2.3) should be connected with the epipolar lines horizontally. *Bouguet's algorithm* defines a rotational matrix R_{rect} that rotates the left camera around O_l, so that the epipolar lines become horizontal and project the left camera's epipoles to infinity. The resulting R_{rect} can be defined by three mutually orthogonal unit vectors e_1, e_2 and e_3.

$$R_{rect} = \begin{pmatrix} e_1^T \\ e_2^T \\ e_3^T \end{pmatrix} \tag{2.14}$$

The first vector is defined by the left epipole (e_l), and since e_l is aligned with the image center O_l that makes e_l to coincide with the same direction as the translational vector T (shown in Figure 2.4)*. e_2 and e_3 are defined as,

$$e_1 = \frac{T}{\|T\|} \tag{2.15}$$

The vector e_2 is orthogonal to e_1 and can be defined in the principal rays' direction. Since the translational vector T, as is shown in Figure 2.4, has its origin at the center of the left image, the third value, T_z is zero since the depth at the principal point (O_l and O_r) is zero.

$$e_2 = \frac{\begin{bmatrix} -T_y & T_x & 0 \end{bmatrix}^T}{\sqrt{T_x^2 + T_y^2}} \tag{2.16}$$

Finally, since e_3 is orthogonal to both e_1 and e_2,

$$e_3 = e_1 \times e_2 \tag{2.17}$$

2.1.3.2 Stereo correspondence

The concept of matching a 3D point in two different image planes for two different cameras are known as the *stereo correspondence*. The matching can only be performed in an area where both the camera views overlap (recall the *working area* in Figure 1.2b). Once the two matching points that describes the same 3D point is found, one would be able to calculate the disparity of the scene, where disparity of point i, j, is defined by $d(i, j) = x_l - x_r$. Here x_r is the image projection of the 3D point p in the right image plane, and x_l is the same

*This translational vector T is in fact, the same vector defined in (2.12b). For easy interpretation, let T_{new} be redefined as T.

3D point's image projection in the left image. A disparity map is an image with the same size as the left (or the right) image with a disparity $d(i, j)$ value for its intensity.

A straightforward method to find correspondences between two images would be to use some kind of feature detection and matching technique (discussed in Section 3.3). Even though delivering accurate results with lesser computational effort, these feature based detection and matching methods provided disparity information only for the detected keypoints, hence generating a disparity map for the detected points. These disparity maps are often called the sparse disparity maps in the literature. However, for depth estimation, one would need the so called dense disparity map, which provides the disparity information for the entire image.

To generate a dense disparity map, one can use the unique formations of epipolar geometry. Epipolar geometry suggests that, in stereo images, the left image's corresponding point lies in the right image's epipolar line. There exist several different methods to find these correspondences and mainly consist of the following steps.

1. Define matching cost.
2. Aggregate the cost over a squared window.
3. Select the minimal aggregated value for each pixel.

In a *local* (block matching) method, the matching cost is calculated using the intensities within a finite window. Sum of squared differences (SSD) [26], Sum of absolute difference (SAD)[27], can be categorized as local methods. On the other hand, global methods would not typically perform the second step of calculating the cost over a finite window, instead, disparities are assigned by minimizing a global cost function that includes all the matching costs. In this thesis, an OpenCV implementation of a robust method called Semi Global Matching (SGM) will be used. A local window-based block matching is performed along the epipolar lines while iteratively minimizing its global cost function.

2.1.3.3 Triangulation and depth estimation

Triangulation is the process that uses the epipolar geometry and the disparity values obtained from the stereo correspondences (Section 2.1.3.2) to estimate the depth map of the scene. In triangulation, the following things will be assumed,

- A stereo rig with known camera parameters (as described in Section 2.1.2).
- The two stereo images having image planes that are coplanar with each other with parallel optical axes*
- The two optical centers are apart from a known distance and have equal focal lengths (in practice, having approximately equal focal lengths are enough $f = f_l \approx f_r$).
- Images are row-aligned as described in the stereo rectification section (2.1.3.1). Having row-aligned images in a stereo rig is often called a frontal parallel camera arrangement.

Figure 2.5 illustrates the geometric relationship of the projection scene in 2D with the assumptions that are mentioned above. By using similarity triangles, one can calculate

*Also known as the *principal rays*. This is the ray that goes through the image centers c_x^{left} (or c_x^{right}) and the optical center O_l (or O_r for the right image plane). These two principal rays will meet at the infinity or more commonly known in the literature as the vanishing point (Figure 2.5)

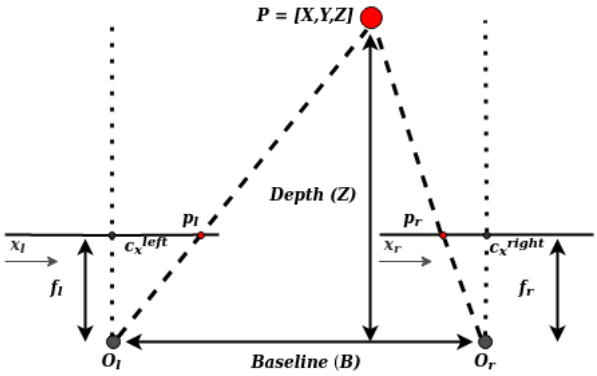


Figure 2.5: Geometric relationship of stereo vision in 2D. Point P is a 3D world coordinate that has a depth of Z. The values c_x^{left} and c_x^{right} are the principal ray intersections of the two image planes. For calculations only the x coordinate of the two image points p_l, p_r will be considered.

the depth given the information of disparity (defined as $d = p_l - p_r$) and the baseline B between the two cameras. The two similar triangles that will be considered here are the triangles connected by points P, p_l, p_r and P, O_l, O_r. When considering the smaller triangle, its height will be equal to $Z-f$, and its base can be defined as $B-((p_l-c_x^{left})+(c_x^{right}-p_r))$. After stereo rectification, c_x^{left} and c_x^{right} will have the same coordinates resulting in the base of the smaller triangle to be $B-(p_l-p_x)$. The depth Z can be calculated according to (2.18).

$$\frac{B-(p_l-p_r)}{Z-f} = \frac{B}{Z} \Rightarrow Z = \frac{fB}{(p_l-p_r)} = \frac{fB}{d} \tag{2.18}$$

2.2 Feature detection and matching

The pose estimation method investigated in this thesis is heavily relied on the vision input as it does not have any other sensory data. The movement of the vehicle is, therefore, tracked by observing the surrounding environment. In general, sensors like wheel odometry can describe a robot/vehicle's motion in a quantitative way. Similarly, to detect the motion in vision application, special features in the surrounding will be used. These features are often called as *landmark* or *feature points*. These landmarks represent special objects, points, or textures in the surrounding environment. By tracking them throughout a sequence of images one would be able to estimate a motion vector of a given camera system. Feature detection and matching consists of three different processes:

1. Prominent keypoints (landmarks) need to be detected.
2. A descriptor will be defined, which describes the surrounding area (image patch) of the detected keypoints, which can later be used to recognize the same keypoint.
3. In the matching phase, a selected keypoint will be matched between two different frames.

These keypoints, which are often call *feature points*, *interest points*, or *landmarks* in the literature, are detected using either the image intensities or an edge detection method. A *feature* should be a point that can be easily recognized in image space. For example, a

point in "a clear sky" should not be recognized as a feature, but a "corner of a building" can be identified as a feature point.

There are different methods available for feature detection and matching. Scale-Invariant Feature Transform (SIFT) [28], Speeded-up Robust Feature (SURF)[29], Binary Robust Independent Elementary Features (BRIEF) [30], Oriented Fast and rotated BRIEF (ORB) [31] are the most popular feature descriptors that can be found in the literature. Binary Robust invariant scalable keypoints (BRISK) [32] and Fast retina keypoint (FREAK) [33] are feature detectors that were introduced much later that have comparatively better computational performance compared to the methods mentioned above. The feature detection part is quite similar in all of these different methods. However, all these methods have unique ways to define the descriptor values, which comes with its advantages and downsides. Depending on the descriptor technique, the feature detector can be divided into mainly two groups: the *vector-based* features and *binary* features. Vector-based features are defined by intensity gradient changes over an image window, whereas binary features are defined as Gaussian smoothing kernels over an image patch. While vector-based features provide robust results in general cases (no rotations, no scale differences), they tend to have a much bigger computational complexity than that of binary features. In this thesis, the feature detector ORB will be used.

ORB serves as an efficient alternative method to SIFT and SURF. As [31] suggests, the algorithm shows excellent robustness for feature extraction, based on FAST and BRIEF. The descriptor is at two orders of magnitude faster than SIFT. In ORB, FAST is performed for keypoint detection, despite FAST being widely used, the drawback is also noticeable: it does not have an orientation operator. The intensity centroid presented in [34] provides a straightforward way to determine the corner orientation. With the assumption that the corner's intensity is offset from its center, the vector could impute an orientation. Given an image patch that contains pixels in large form. The moments of a patch are defined as:

$$m_{ij} = \sum_{x,y} x^i y^j I(x, y) \tag{2.19}$$

Provided the moments calculated according to (2.19), the centroid can be determined by:

$$C := (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}) \tag{2.20}$$

A vector $\overrightarrow{OC}$ is constructed, where O and C represent the corner's center and centroid, respectively. The orientation of the patch can then be expressed as:

$$\theta = atan2(m_{01}, m_{10}) \tag{2.21}$$

The BRIEF serves as the basis for descriptor calculation of ORB. Given a smoothed image patch, a binary test τ is used, which is defined as:

$$\tau(p; x, y) = \begin{cases} 1 & : p(x) < p(y) \\ 0 & : p(x) \geqslant p(y) \end{cases} \tag{2.22}$$

where $p(x)$ and $p(y)$ are the intensity of the pixel at the location x and y in the patch. The binary test is carried out in the patch where points follow a Gaussian distribution, the test is implemented n times, which yields a vector f of length n:

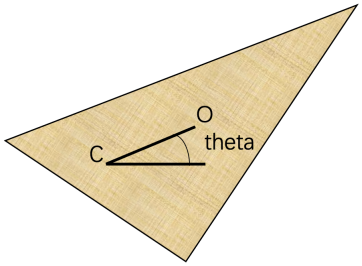


Figure 2.6: Orientation of the patch.

$$f_n(p) := \sum_{1 \leqslant i \leqslant n} 2^{i-1} \tau(p; x_i, y_i). \tag{2.23}$$

As [31] suggests, BRIEF's matching performance falls off sharply for in-plane rotation of more than a few degrees. ORB proposes a method to steer BRIEF according to the orientation of the keypoints. For any feature set of n binary tests at location (x_i, y_i), one would need a matrix of size $2\,n$. Then it discretizes the angle to increments of $2\pi/30(12^\circ)$, and construct a lookup table of precomputed BRIEF patterns. As long as the keypoint orientation Θ is consistent across views, the correct set of points will be used to compute its descriptor.

2.3 SLAM

SLAM, known as Simultaneous localization and mapping, is a process of constructing a map and simultaneously estimating the pose of a vehicle in an unknown environment. This process's difficulty is often referred to as a "chicken and egg" problem based on the fact that localization and mapping are mutually dependent. An accurate map is needed to estimate the pose of the vehicle, and at the same time, an accurate pose is also needed to construct the map. While this problem appears to be complicated, several algorithms are developed to solve this problem.

There are two main parts in the *SLAM* problem, and in the literature the two parts are called the *front-end* and the *back-end*. The front-end processes the raw sensor data from the various sensors and represents those data in an immediate representation of the environment. These representations could be the *constraints of an optimization problem* or the *probability distribution of a landmark* etc. The*back-end* takes the above-mentioned intermediate representation and solves the underlying state-estimation or the optimization problem. In a nutshell, estimating the parameters that describe where objects are in the environment with respect to the vehicle will be done in the *back-end*. Mainly three different categories of approaches can be found in the *back-end* process.

- EKF - Extended Kalman Filter
- Partical Filters
- Least squares optimization((Graph based SLAM)

In this project, the open-source SLAM algorithm *OpenVSLAM* adopts the most commonly used *graph-based SLAM* approach in solving the state estimation problem. The general idea behind all SLAM problems is illustrated in Figure 2.7.

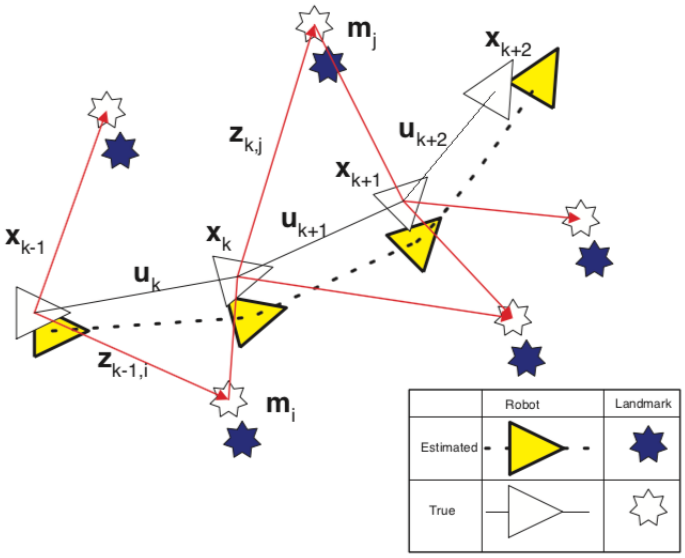


Figure 2.7: SLAM problem representation [14].

Given the situation where the vehicle is running in an unknown environment, a sequence of variables are defined to achieve localization and mapping [14].

1. x_k: A state vector, which encodes the information of the pose of the vehicle.
2. u_k: A control vector that is applied to the vehicle from time $k-1$ to time k.
3. m_i: A landmark location vector containing the information of landmarks of i.
4. z_k : A landmark observation vector which contains the observation taken at time instance k.
5. $X_{0:k} = \{x_0, x_1, ..., x_k\}$: The trajectory of the vehicle from instance 0 to k.
6. $U_{0:k} = \{u_1, u_2, ..., u_k\}$: The sequence of control signals.
7. $m = \{m_1, m_2, ..., m_n\}$: The sets of landmark locations.
8. $Z_{0:k} = \{z_1, z_2, ...z_k\} = \{Z_{0:k-1}, z_k\}$: The sets of observations.

Since there is uncertainty in the vehicle's motion as well as the observations, probability theory can be used to express the uncertainty explicitly. The SLAM problem can therefore be formulated in a probabilistic model as (2.24), where the target is to obtain the vehicles' path as well as a map given the information of controls and observations.

$$p(x_k, m | Z_{0:k}, U_{0:k}, x_0) \tag{2.24}$$

2.3.1 Overview of VSLAM

VSLAM, or the visual SLAM, is a method where a visual aid is used as input sensor data. Lately, VSLAM has shown great popularity among the engineering community for its flexibility in hardware and easy software integration. VSLAM has the same structure of

SLAM, the only difference being that in VSLAM, instead of sensor reading from sensors like odometry, GNSS, IMU, etc., VSLAM defines environmental changes by detecting feature movements in the scene for its estimation in the *front-end* part. The *back-end* part is the same as in the previous section. An overview of the *front-end* section of the general VSLAM algorithm is illustrated in the next few figures.

In Figure 2.8, the initial setup of a scene is illustrated. A typical scene often contains moving objects (like vehicles) and static scenes like buildings, road signs, etc. The camera view is shown in Figure 2.8c as the resulting view from the stereo rig, as explained in Figure 1.2. When the vehicle moves forward (or in any direction), it will start capturing different scenes at each time instance. These captured scenes could be slightly different from the previous scene or could have a considerable amount of changes compared to the previous scene. For example, when a vehicle is taking a turn, the changes will occur more frequently compared to that in a straight line motion. Hence a turn is named an eventful situation* in this thesis.

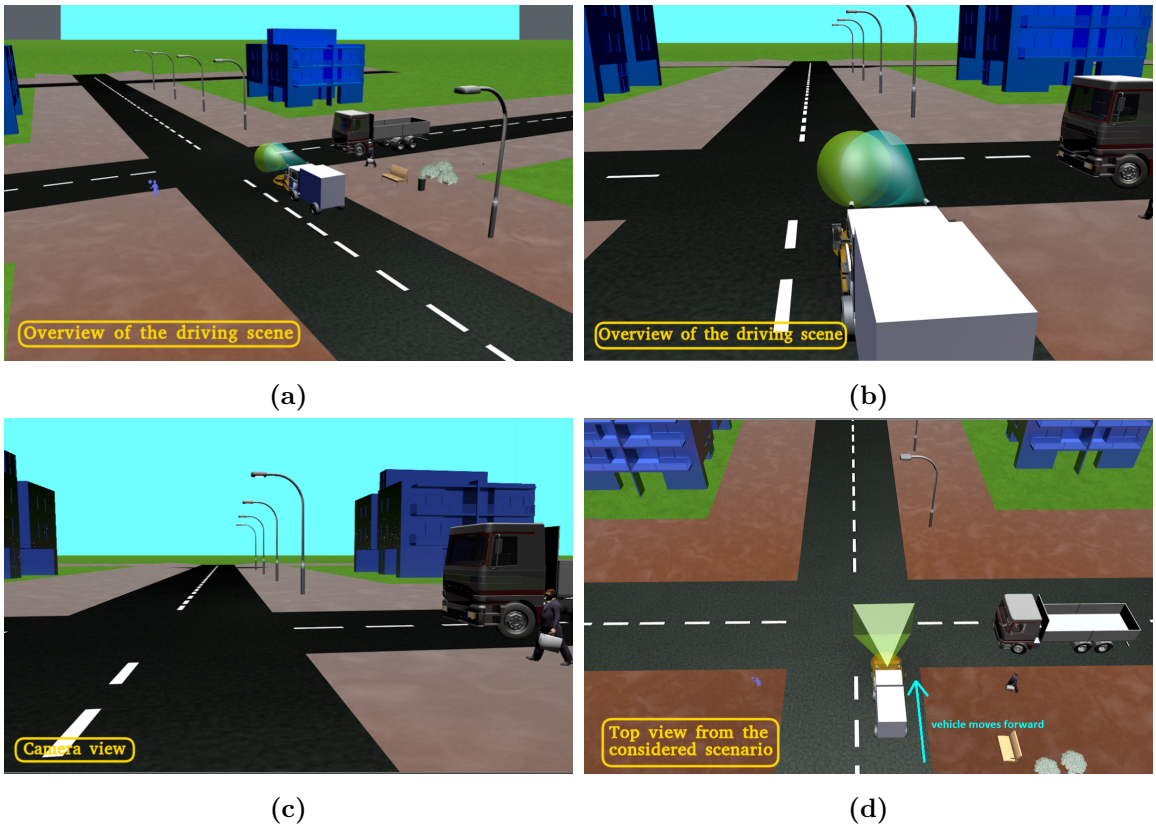


Figure 2.8: Overview of the driving scene. Figures (a) and (b) show how a typical environment would look in an urban area. Figure (c) camera view of the intersected images from left and right cameras. Finally, a scenario of a vehicle moving forward is illustrated in figure (d).

*Analyzing an eventful situation is quite important because the final pose estimation will depend not only on feature extraction but also on feature tracking. If the previous feature can not be tracked in the current frame, which often occurs in an eventful situation, the algorithm has a high chance of breaking the continuous pose estimating.

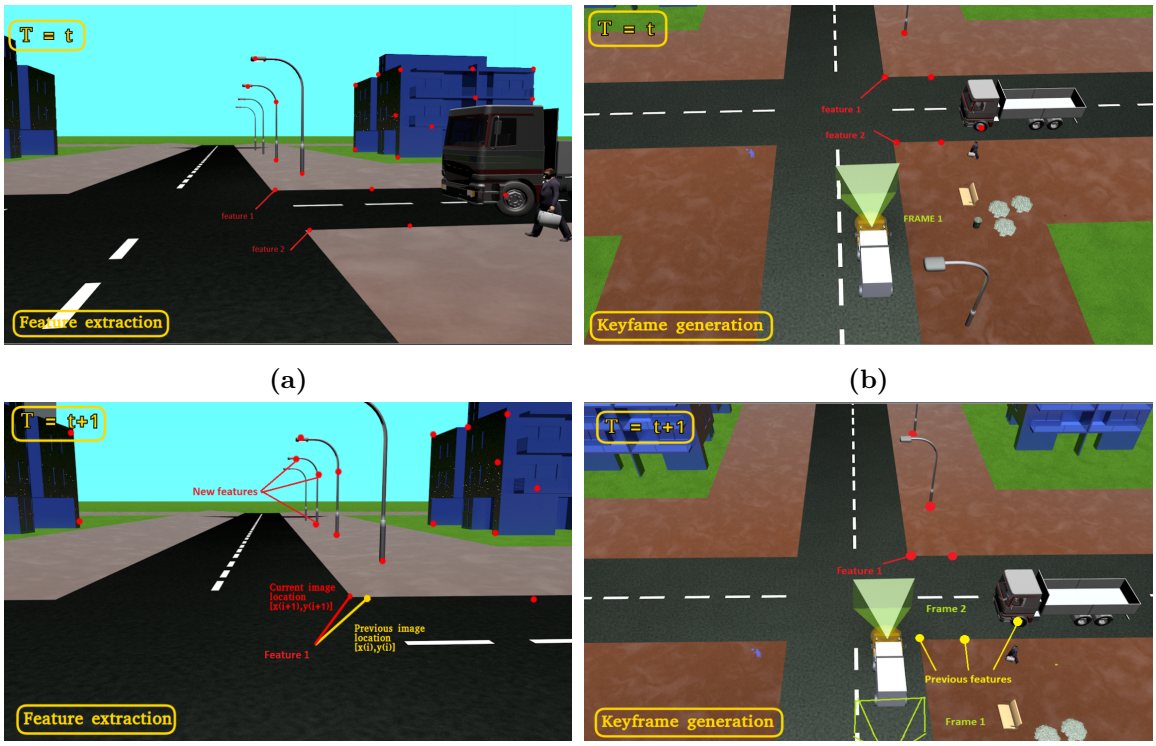


Figure 2.9: Feature extraction and key frame generation. Figure (a) is scene capture at time T, and (b) is the keyframe generation part at time instance T. Figures (c) and (d) are the respective processes at time instance $T + 1$.

Figures 2.9a and 2.9b illustrate the feature extraction and the keyframe generation at time instance T. In feature extraction, the detected features are labeled as landmarks, and each of these landmarks is given a specific id so that those can be distinguished among other landmarks. Typically a landmark apart from its id contains the following attributes,

1. kf_{initial}: This gives the initial keyframe id that was detected for this particular landmark. In other words, which keyframe detected this landmark first.

2. $pose_i$: This is often defined as a 3×1 vector, which indicates the landmark position with respect to the camera.

3. kf_i: Typically, a landmark will be visible through at least a couple of keyframes. The kf_i represents the i^{th} keyframe id that detects a particular landmark

Therefore, a standard landmark vector contains two ids and two vectors that represent its location and behavior throughout a visual movement. Meanwhile, a keyframe generated at time T will be defined as keyframe_T, which includes a unique id, a camera pose, and a vector of detected landmarks. Once the pose estimation is finished, the estimated states will be fed to the SLAM back-end, resulting in a detailed map as shown in Figure 2.10.

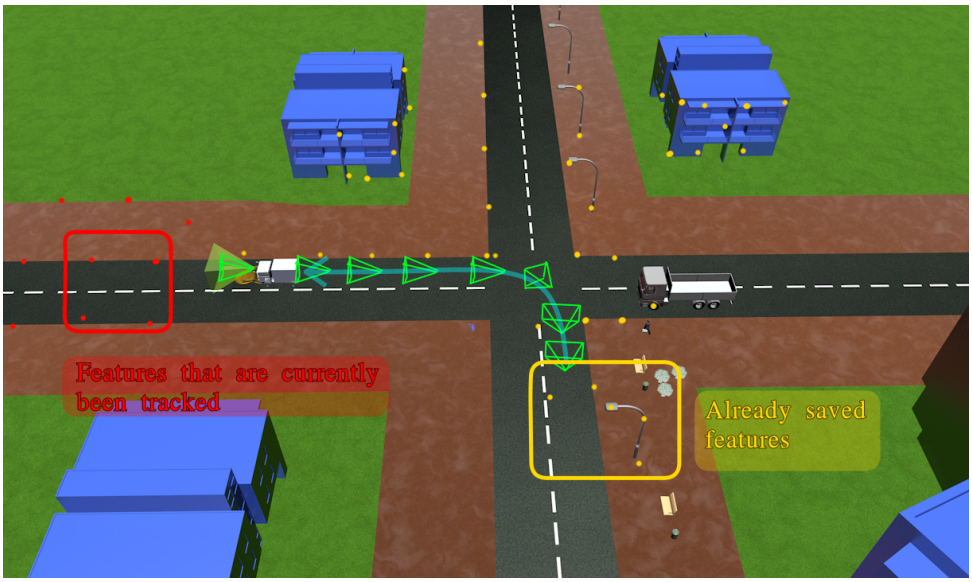


Figure 2.10: Resulting trajectory with the estimated poses. Red color points are currently tracked features. The yellow color points represent the already saved features. These features will be saved as map data, which can be used in the localization phase. Green color pyramids are the generated keyframes, while the blue color arrow shows the estimated trajectory.

2.3.2 Graph-based SLAM

Graph based SLAM, as mentioned in previous section, is a SLAM paradigm which is based on least squares error optimization methods. Among three different categories of approaches, graph based optimization has following characteristics :

1. The problem is presented by a graph.
2. The graph contains nodes, which represent poses of the vehicle.
3. The edge between two nodes refers to the spatial constraint.
4. The key idea is to construct a graph and find a best node configuration that minimizes the error caused by the constraints' effect.

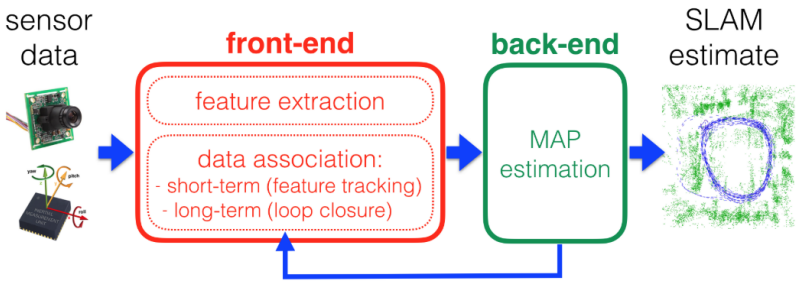


Figure 2.11: Graph-based SLAM framework [35].

A typical graph-based SLAM framework that consists of front-end and back-end parts is illustrated in Figure 2.11. The nodes represent the states of the vehicle, which could be used to generate predicted observation, the key idea of graph-based SLAM lies on

finding a best configuration of the nodes such that the error between real and predicted observations are minimized. A graph-based optimization method called *g2o* [13]is used in implementation. The details of mathematical derivation are presented in Appendix A.4.

2.3.3 OpenVSLAM

OpenVSLAM [17] is an open-source, state-of-the-art visual SLAM framework with high usability and extensibility. The software is developed to incorporate several useful features and functions, therefore, it can serve as a library that is compatible with third-party programs.

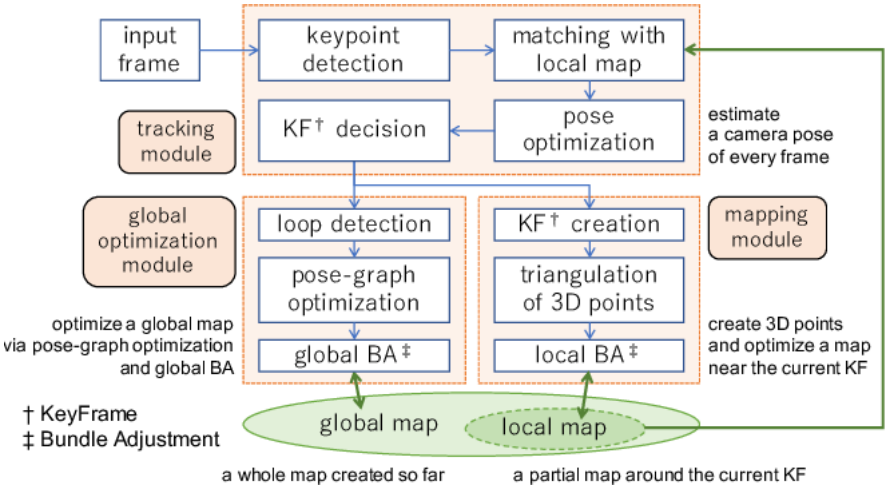


Figure 2.12: System architecture of *OpenVSLAM* [17].

As described in Section 2.3, a typical visual SLAM algorithm can be divided into mainly two processes: the front-end and back-end parts. The same architecture can be found in the OpenVSLAM library. As one can note in Figure 2.12, OpenVSLAM consists of three modules, the tracking module, the mapping module, and the global optimization module. The tracking and mapping modules fall into the front-end part, and the optimization part, which performs a graph-based optimization, falls into the back-end part. The camera pose for each frame will be calculated in the tracking module, and this calculated pose will be optimized in the local optimization in the back-end section. A decision will then be taken to save the newly calculated pose as a keyframe (KF). If it is selected as a KF, then this new KF will be saved in the map. The new KF will then be sent to the mapping module for global optimization. Here the map will be updated with the 3D points in the new KF using triangulation techniques described in Section 2.1.3.3. Furthermore, a loop-closure detection using a search window in the map will be performed. This part is defined as the global optimization, and this uses the technique of *Bundle Adjustment (BA)* [36]. The term loop closure stands for the algorithm of detecting a previously visited area/section of the map. This technique is also used in the localization stage of OpenVSLAM, where a pose will be estimated relative to a previously recorded map.

3

Methods & Evaluation Setup

This thesis focuses mainly on two objectives: evaluating the accuracy in pose estimation of an urban vehicle using a computer vision-based approach, and detecting free-space in front of a given vehicle. A visual SLAM approach is proposed in this thesis to solve the pose estimation problem, and in this project, *OpenVSLAM* will be evaluated. One should also note that the current system is implemented under the ROS environment [37], hence, a few modifications were made to the *OpenVSLAM* package to obtain the expected results. Camera calibration is a very crucial part of this thesis. The calculated parameters were evaluated using the reprojection error. One other key component in any visual SLAM algorithm is feature detection and tracking. The effects of different *field of views (FoVs)* on the estimation accuracy were also evaluated. An image stitching method to increase the image view as a solution to small FoVs was also implemented and evaluated. Finally, a free-space detection method was implemented, where the objects that are closer to the vehicle, are detected through a technique called *salient object detection*. The overview of the pose estimation system is given in Figure 3.1.

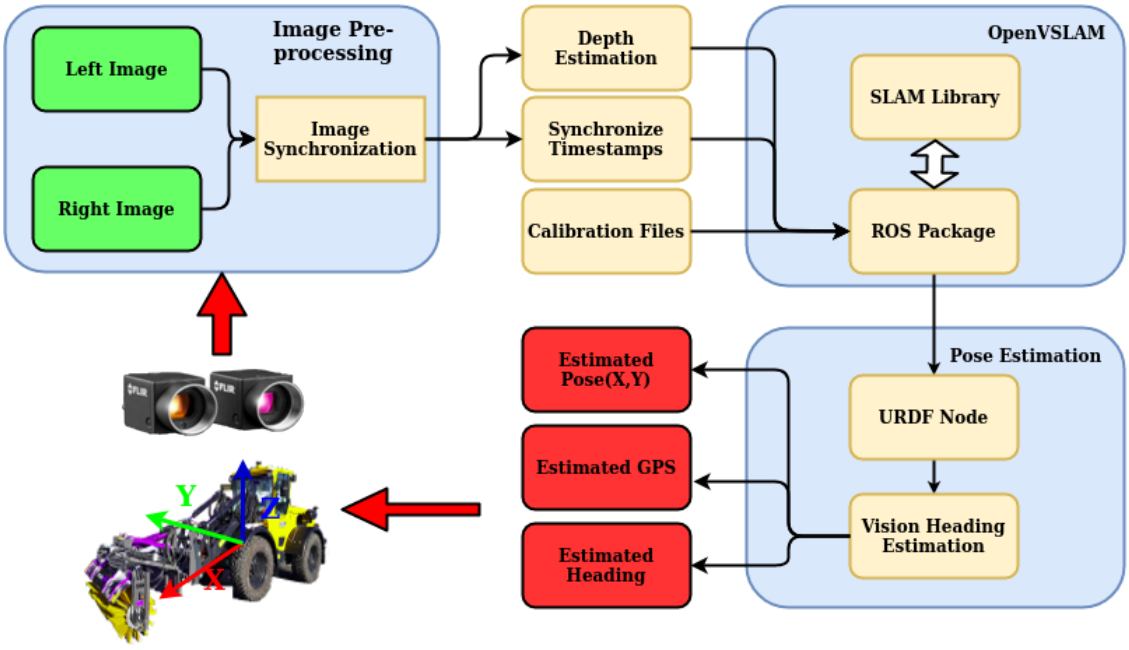


Figure 3.1: Overview of the pose estimation system.

3.1 Image Prepossessing

Image preprocessing, as shown in the first block in Figure 3.1, is a critical step before passing the image data to the *OpenVSLAM* algorithm. The images captured from the two independent cameras have two different frequencies, or in the computer vision domain, the frame rate per second (fps) of the left and the right cameras are different. The difference in terms of frequency in *Hz* is relatively small (50Hz and 55Hz for left and right, respectively*). However, this difference tends to increase over time, resulting in unsynchronized images that affect the stereo vision implementation as described in Section 2.1. The intrinsic matrix (in (2.1)) is also essential in the visual SLAM algorithm. Estimating these parameters as accurately as possible is, therefore, essential. In this section, the overall implementation of the image preprocessing algorithm will be discussed. In Section 4.1 practical challenges that one might face when implementing the following algorithms are presented.

3.1.1 Image Synchronization

The *Barmark* dataset is a *rosbag* [38] file that contains data as *compressed images*. Here, the data has to be recorded in a certain way such that it takes a minimum amount of memory in the system. One should note that one image has a resolution of 960×600 and have *RGB* information, and therefore, the amount of memory to record a few minutes of data will consume a considerable amount of memory†. To avoid this, the images are recorded as *compressed images*. The initial task would decompress the *rosbag* images, and this was done by subscribing to the two message feeds (from the respective cameras) and republishing the same message with *RGB* encoding along with a new timestamp. These new images were then recorded to another *rosbag* file for further processing.

The implementation of the *image synchronization* process is shown in Algorithm 1. The algorithm starts with selecting the message data from the *rosbag* file, which is followed by creating two lists of data vectors that contain image details and respective timestamps of the two cameras. The terms *Parent* and *Child* came from the stereo rig setup of *FLIR camera system*. The stereo rig is set up so that the left camera triggers the right camera, thus in this project, the left camera act as the parent triggering the right camera, the child.

3.1.2 Camera Calibration

The traditional method of camera calibration (described in Section 2.1.2), the DLT, is, of course, not possible in this thesis, since accurately measuring the distance to different points in the scene is impractical. Thus, *Zhang's method* in estimating the intrinsic matrix (recall the definition (2.1)) was used. A checkerboard with black and white squares was

*Theoretically, two identical cameras, even though they process images independently, should have the same frequency. However, this is not the case in the real world; apart from minor mismatches in the hardware and software, the main contribution to the frequency mismatch is that the stereo rig is set up so that one camera triggers the other.

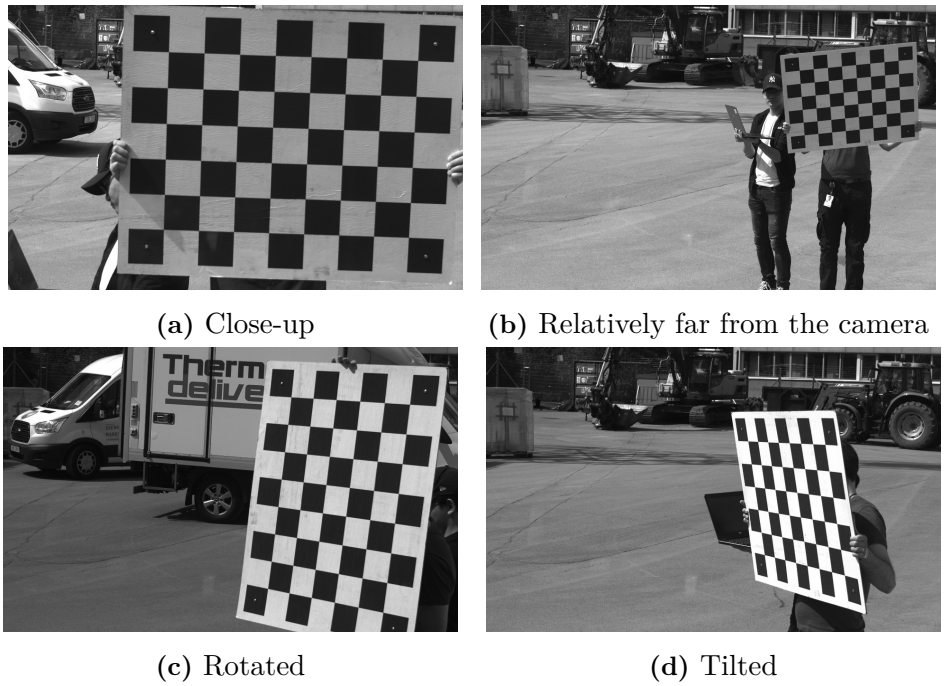
†The dataset records in this thesis took approximately two minutes. For these two minutes, with compressed images, it took a memory close to 4GB. Therefore, if one recorded the same data without decompressing, it would likely take way more than 4GB.

Algorithm 1: Image synchronization

Input Two sets of unsynced images (left and right) and their timestamps
Result: Two sets of synced images (left and right)
initialization;

1. extract image data and the timestamp values from the rosbag file
2. for each message, a data vector is generated with the image data and timestamp.
3. insert all the data vectors from the previous step to two lists *left* and *right*.
4. select the list that contains the most number of data vectors and name it *Parents* and the other vector *Children*.

define initial values;
current time = 0;
previous time = 0;
initial_image = True;
synced left images = empty list;
synced right images = empty list;
synced timestamps = empty list;
for *each child in Children* **do**
    `time_child` = timestamp value of child;
    `is_frame_synced` = False;
    `ignore_frame` = True;
    while *frames are not synced* **do**
        for *each parent in Parents* **do**
            `time_parent` = timestamp value of parent;
            `time_difference` = `time_parent` - `time_child`;
            if $|$*time_difference*$| \leq$ *pre-defined threshold* **then**
                `is_frame_synced` = True;
                `ignore_frame` = False;
                add parent image to synced left images;
                remove parent image and the timestamp from Parents;
            end
        end
    end
    if *not* *ignore_frame* **then**
        add child image to synced right images;
        if *initial_image* **then**
            initial_image = False;
            previous time = `time_child`
        else
            time difference = `time_child` - previous time;
            current time += time difference;
            add current time to synced timestamps;
        end
    end
end



(a) Close-up **(b)** Relatively far from the camera

(c) Rotated **(d)** Tilted

Figure 3.2: Different poses of the checker board. Taking images with different poses, as shown in the subfigures, are essential.

placed in front of the cameras, and several images were taken for further processing. It is vital that both cameras can capture all the small squares in the checkerboard. As described in Section 2.1.2, specifically in *Zhang's method* of camera calibration, to estimate the homography matrix is shown in (A.6) for a single image, one would need at least four points in that image plane. Then, once the homography for a single image is estimated, to estimate camera parameters (the K matrix), the B matrix shown in (A.13) needs to be formulated. To solve the values for B using (A.18), one would need at least another three images. Different poses of checkerboard images as required by the algorithm are shown in Figure 3.2.

Firstly, a new set of 3D points are defined. In (2.6), the value of Z is set to zero so that it will take the form of (2.8). Since the value of Z is zero, these sets of 3D points will be the same for all the images. The values of the X and Y will be defined according to the checkerboard's dimensions. A 3D point in space $X_k^{'}$ where Z_k is equal to zero can be expressed as,

$$X_k^{'} = \begin{pmatrix} X_k \\ Y_k \\ Z_k \end{pmatrix} = \begin{pmatrix} i \times d \\ j \times d \\ 0 \end{pmatrix} \text{ where } \begin{matrix} i = 0....W \\ j = 0....H \end{matrix} \tag{3.1}$$

where the k^{th} 3D point can be define as $k = i + j$. Here, the values W and H are the number of squares in x and y directions, respectively, in an image plane* and d is the length of one square.

*The value $W \times H$ can be defined as the dimension of the checkerboard. In this thesis, the values were 8×6. The value $W \times H$ is also equal to the number of feature points in the checkerboard.

Then to obtain the required intrinsic matrices, the two sets of images (the left and the right) were fed to Algorithm 2. Two pre-derived function from *OpenCV* was used in this algorithm: *findChessboardCorners* and *stereoCalibrate*. This method is an offline calibration method. Two sets of images fed to the algorithm are assumed to be synced. The function's output, the two intrinsic matrices, and the two distortion vectors will be obtained along with the rotational matrix and the translational vector between the two cameras (all these matrices and vectors are called the camera parameters).

Algorithm 2: Stereo calibration - Off-line method

Input Two set of synced images (left and right) with different checkerboard poses.
Result: Calibration parameters
initialization;
1. read image data and save all the images in a list named *Images*
2. create a vector of 3D points as given in (3.1)

defining initial values;
$left\_image\_point\_vector \leftarrow \emptyset$;
$right\_image\_point\_vector \leftarrow \emptyset$;
▷ *Both these vectors contain another set of vectors that contain the 2D image points per each image*
for *each image in Images* **do**
  use the function *findChessboardCorners* to detect corner features of each image (left and right);
  if *corner points are detected for both images* **then**
    add the two point vectors to $left\_image\_point\_vector$ and $right\_image\_point\_vector$
  end
end
resize the 3D point vector to the same size as the two 2D image point vectors;
apply the function *stereoCalibrate*

The next method is the *online camera calbration* method. This method is, of course, quite similar to the stereo offline calibration method. However, the algorithm is written so that it can perform calibration in real-time. This real-time camera parameter estimation is possible because of the ROS architecture. However, this online algorithm can not perform stereo calibration (in the current version), and the two cameras have to be calibrated separately. The algorithm is quite similar to Algorithm 2. Initially, the 3D points will be defined, and for each image, the OpenCV function *findChessboardCorners* will be used to detect 2D image points in the checkerboard plane. All these detected points will be saved in a separate vector, and when the vector has a size bigger than 15*, a new thread will be opened for the calibration process. Since the calibration is done in a separate thread, real-time image acquisition will not be disturbed (this would result in updating the image point vector defined by the function *findChessboardCorners*).

*In OpenCV documentation this is the recommended minimum number of iterations required to get reasonable estimations in camera calibration

Once in the calibration mode, the function *calibrateCamera* in *OpenCV* will be used to estimate the camera parameters. For this function, as input, the image coordinate vector is detected from *findChessboardCorners*, and a 3D point vector with the size of the 2D image point vector will be given. This function will output the camera matrix, distortion parameters, rotation matrix, and the translational vector. One would then calculate the K matrix (the intrinsic parameters) from the camera matrix. To evaluate the estimated matrices, the method called *reprojection error* was used. In this error evaluation, the 3D points will be projected onto the image plane using the newly acquired camera matrix. The error obtained after this operation is called the reprojection error, and pseudo-code for calculating the reprojection error is given in Algorithm 3.

Algorithm 3: Reprojection error calculation.

Input Estimated camera matrix, detected image points, and the 3D points
Result: Reprojection error for an estimated camera matrix
initialization;

1. Insert insert all the detected image points to a vector with size $1 \times n$ where n is the number of checkerboard images. This vector has sub-vectors, each having m number of points that are equal to the number of feature points in a single checkerboard
2. create a vector of 3D points as given in (3.1) (this vector should also be resized to have the same size as the image point vector defined in the previous point).

defining initial values;
Total Error = 0.0;
for $i = 0;\ i < n;\ i{+}{+}$ **do**
    error per point = 0.0;
    for $j = 0;\ j < m;\ j{+}{+}$ **do**
        use the function *projectPoints* to calculate the estimated image points ${}^{estimation}(x_j, y_j)$;
        if actual detected point is ${}^{actual}(x_j, y_j)$;

$$\text{error per point} += \left({}^{actual}(x_j, y_j) - {}^{estimation}(x_j, y_j)\right)^2 \tag{3.2}$$

    end

$$\text{Error per image} = \sqrt{\frac{\text{error per point}}{m}} \tag{3.3}$$

    Total Error += Error per image;
end

$$\text{Reprojection Error} = \frac{\text{Total Error}}{\text{Total Images}} \tag{3.4}$$

3.2 System implementation

The implementation process of this thesis's two main objectives, the pose estimation, and the free-space detection, will be discussed in this section. The architecture of the system implementation process of this project is illustrated in Figure 3.3. In the first section, the *OpenVSLAM* and the *Pose Estimation* blocks shown in Figure 3.1 will be discussed. The *OpenVSLAM* library [17] is mainly written in C++. However, this library was not fully integrated with the *ROS* system at the time of implementation of this project. Therefore, some extra modification was done so that the system is adapted to *ROS*. In Figure 3.3, the whole process of estimating the vehicle's pose given stereo camera images are labeled as the *mapping stage*. The *OpenVSLAM* library can also solve the localization problem, even though not investigated in this thesis in the localization phase, further optimizing the estimated trajectory by loop-closure detection is possible. Apart from loop closure detection, a map recorded in the initial mission's* mapping stage can be used in a different mission to obtain relative poses with respect to the initial mission. A free-space detection algorithm to identify objects closer to the vehicle was also implemented in this thesis and will be further discussed in Section 3.2.2.

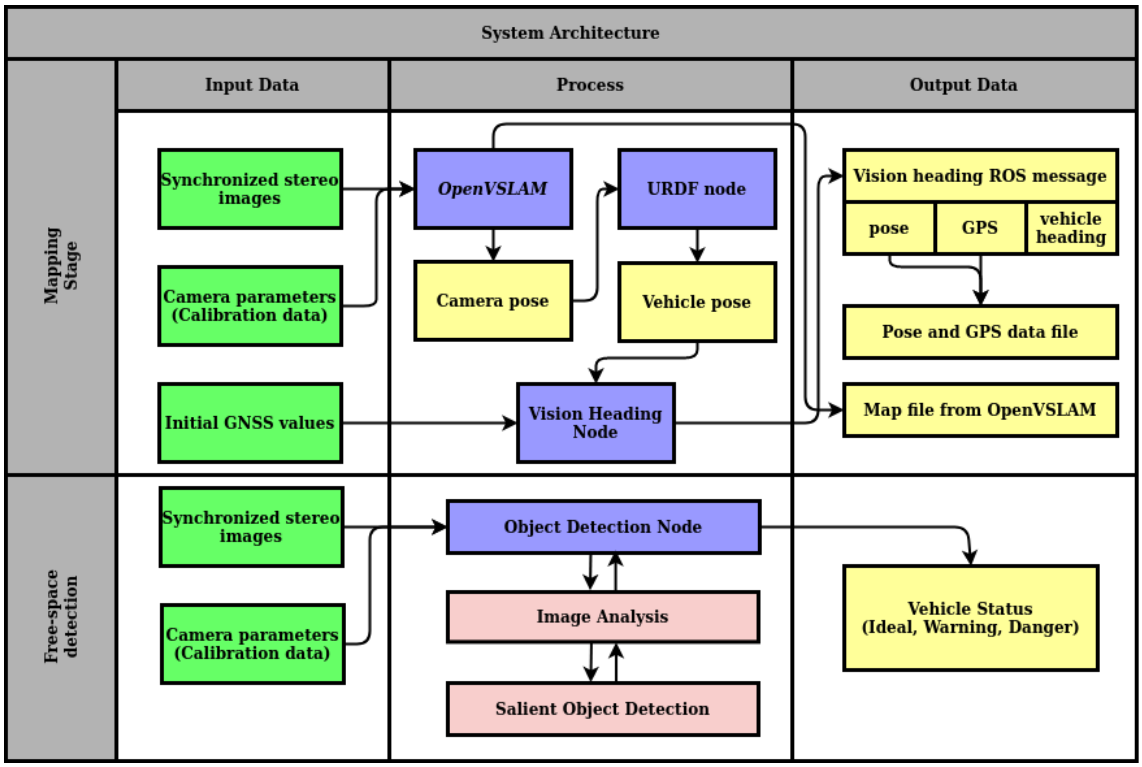


Figure 3.3: Overview of the project architecture. Here, the green color blocks represent the algorithms' input data, and the blue color rectangles are the process. The three blue color blocks labeled as *nodes* are built under the *ROS* environment. Red color blocks are independent libraries that are not dependent on *ROS*. The yellow color blocks represent the outputs.

*The term mission in this project refers to a single drive/activity or a job performed by the utility vehicle. In the context of this thesis, it refers to recording the map data, which includes defining the trajectory, recording landmarks.

3.2.1 Mapping stage

In this subsection the *mapping stage* illustrated in Figure 3.3 will be further evaluated. As an overview, this mapping stage receives data from pre-recorded images, calibration data, and initial GNSS data. In the second processing block, a modified version of the*OpenVSLAM* library will gather the information from the initial images and the calibration parameters to estimate the camera pose. The output camera pose (shown in yellow color) will then pass through the *URDF** node, where necessary transformations are done to estimate the pose in the real world coordinate system. This newly estimated vehicular pose will be fed to the vision heading node that manages further calculations for vehicle headings and GNSS estimations. A detailed description of this flow is given next.

Synchronized images and the calibrated camera parameters will be given as inputs to *OpenVSLAM* library. The entire process from obtaining images to estimating the vehicle's pose will be addressed as the *Mapping stage* in this thesis. The following sub-processes are implemented within the mapping stage:

1. The main *ROS* wrapper for the *OpenVSLAM* library.
2. The URDF node, which describes internal transformation between different coordinate frames.
3. The GPS estimations

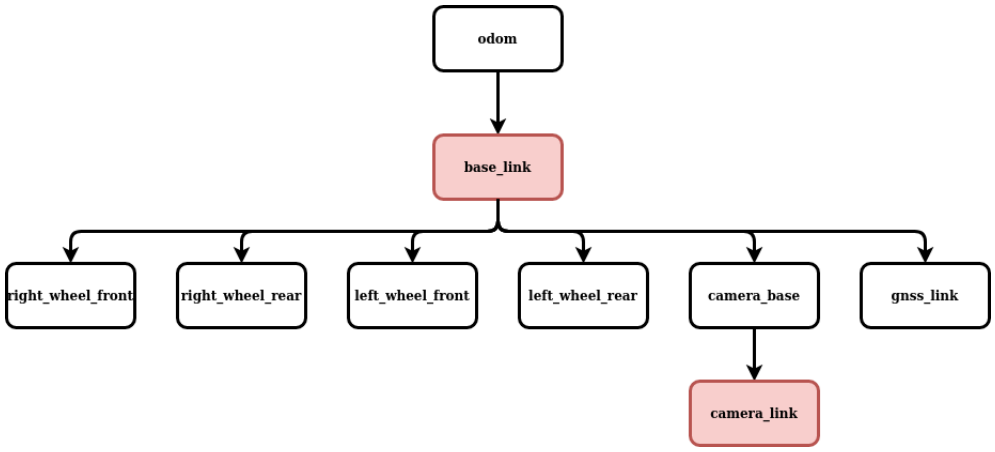
Two types of *ROS* wrappers were implemented and tested in this thesis. In the first type, synchronized images are fed the ROS wrapper as a dataset. Hence no image subscription from a 3^{rd} party ROS node will be performed. The advantage of this method is that it has less stress on RAM usage. The other method was to subscribe to a stream of images that have been published by another node. This method helps simulate real-time performance. However, it uses RAM (computational power) more than the offline method. Another usage of the second method is the capability of publishing data at different rates. Regardless of the two methods, there were mainly three threads† running in parallel to estimate the camera's pose.

1. Parent thread from the *OpenVSLAM* library.
2. *ROS* wrapper to help communicate with the library. This thread also include the data subscription and publishing.
3. Visualization thread for *Pangolin Viewer* [39]. This is the main visualization tool in the *OpenVSLAM* library.

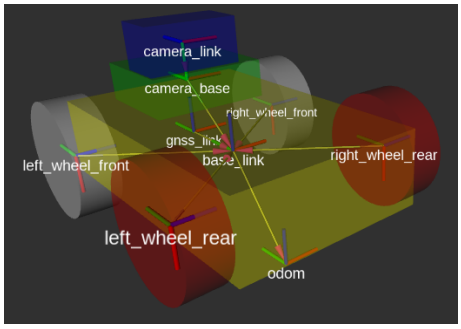
Once the *camera pose* for the stereo rig is obtained, it needs several transformations to describe the same pose in the real world coordinate system. In the *pin-hole* camera model, the depth (the Z axis) is defined outwards from the image plane, and in the case of the real world, usually, the coordinates frames are defined so that it has the Z in the upward direction. In a simple system, manually converting the cartesian values using a known transformation matrix is easy. However, obtaining the transformation matrix in a complex system with moving segments and coordinate frames can be mathematically demanding. To solve this issue of tracking coordinate frames and updating values according to relatively

*URDF stands for unified robotics description format. This is an *xml* file describing all the coordinate frames and dimension of sensors/parts in a robot. A detailed description will be given later in this section

†*OpenVSLAM* library, of course, contain many threads in a single run; however, in this thesis implementation, only the main (or the parent thread) was used to communicate between the library and the local *ROS* system



(a) TF-tree of the system



(b) URDF visualization model

Figure 3.4: TF-tree that describes the system and the URDF visualization model. Figure (a) represents how all the frames are attached to each other. In this thesis, main concern will be the transformation between the *base_link* and the *camera_link*. The term *odom* is a common name that is used to describe the world coordinate frame. The relative pose of the vehicle will be calculated with respect to this *odom* frame. The auto generated ROS version of the tf tree is illustrated in Figure B.2. Figure (b) illustrates the vehicle model and how the different links are connected. The *URDF* configuration file can be adjusted to change the values and properties of each joint.

moving coordinates frames, *ROS* has a concept called *tf**. In simple terms, the *tf* package defines a *tree-like* structure to connect different coordinate frames, and all the branches of this tree are connected with relative transformation matrices. This concept is called defining a *TF-tree*, and it helps to update values throughout the system with respect to any given coordinate frame.

There are many different ways in *ROS* to define the *TF-tree* of a system. When defining the *TF-tree*, one should also consider the actual psychical differences between the coordinate frames. For example, in this thesis, the camera is located about 0.75 m above the vehicle's base. This information needs to feed to the *TF-tree* correctly, or the end transformation matrix between two frames will be inaccurate. The method used in this thesis to define all these connections is the *URDF* node. *URDF* as mentioned is a

*The package name *tf* stands for transformations. More information can be found in the official ros-wiki page, http://wiki.ros.org/tf

configuration file, which describes how the internal sub-systems are connected in a robot. This *URDF* package allows its users to not only define different coordinate frames but also define their joint types. For example, a wheel of a robot (in this thesis, the vehicle) can define as a revolute joint relative to the main link*, while another sensor can be defined as a static link. *URDF* node visualization and the *TF-tree* is given in Figure 3.4.

The *Vision Heading Node* (shown in Figure 3.3, in the mapping stage process section) will take this newly calculated pose after transformation and will publish it as the vehicle pose. Then the heading of the vehicle is calculated using this new pose. In this thesis, the heading is defined as the vehicle direction. Typically, an object in the space has three rotational axes: the roll, pitch, and yaw (or in terms of access in the real world coordinate system X, Y, and Z axis respectively). In the context of this thesis, it is assumed that the vehicle only has rotations around its Z axis or have only yaw angles. Heading direction was therefore calculated as the changes in the Z axis. Two methods were evaluated in calculating the heading, the *previous point* (PP) method, and the *direct yaw* (DY) method. In the PP method, a distance vector between the current point and the previous point was calculated. Then, this vector's arc tangent value was calculated, and the value was assigned as the heading direction. In the DY method, the yaw angle from the newly converted pose was used.

To evaluate the accuracy of the estimations, the estimated pose† was compared with ground truth values. For the *BARMARK* dataset, ground truth values are generated from a GNSS unit and are assumed to be accurate. Two error criteria are used in this thesis to evaluate the estimated poses, namely the Root Square Error (RSE) and Absolute Trajectory Error (ATE) [40]. Further, the mean and median of these two error criteria were also used to summarize the estimations.

Let the ground truth at frame i be $\theta_i = \begin{bmatrix} x_i & y_i \end{bmatrix}$ and the estimation be, $\hat{\theta}_i = \begin{bmatrix} \hat{x_i} & \hat{y_i} \end{bmatrix}$, then RSE for frame i will be,

$$RSE_i = \sqrt{\left(\theta_i - \hat{\theta}_i\right)^2} \tag{3.5}$$

and the mean RSE for N number of frames can be expressed by,

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\theta_i - \hat{\theta}_i\right)^2} \tag{3.6}$$

In heading estimation (orientation part of pose estimation) instead of (x, y) values the three angles,yaw,pitch and roll will be used to calculate RSE values for each iteration.

The second criterion, ATE, is only evaluated for trajectory errors. In ATE, instead of merely taking the difference between ground truth and the estimation, a distance value will be calculated (using x, y coordinates).

$$ATE_i = \sqrt{(x_i - \hat{x_i})^2 + (y_i - \hat{y_i})^2} \tag{3.7}$$

*It is standard practice to define one main link in the system. This is usually the robot's gravity center and often called the *base_link*. Overall, robot motions are defined with respect to this main link.

†In this thesis, the pose is divided into two groups. The translational part of the pose (x, y values, z is constant) is named trajectory estimations, and the orientation value estimation (will only consider the yaw/angle around z axis) is named as heading estimations. Refer to Sections 4.2 for further description.

similar to the mean RSE, a mean ATE (MATE) for the entire trajectory with N iterations can be calculated by,

$$MATE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x_i})^2 + (y_i - \hat{y_i})^2} \tag{3.8}$$

The transformation vector (the x and the y values in meters) estimated from the algorithm (the estimation coming from the yellow color block named *Vehicle pose* in the *Process* section in *Mapping stage* row in Figure 3.3) are relative values to a specific mission. Meaning the initial value will always start from $(0, 0)$. However, these relative values can not be used when stating the vehicles' location in a map with longitude values and latitude values. Therefore a conversion between the relative values x, y coordinates to the absolute location in longitude and latitude is essential. The algorithm used to estimate the longitude and latitude is given in Algorithm 4.

Algorithm 4: Longitude and Latitude estimation

Input Initial GNSS data and continuous pose estimation

Result: Longitude and Latitude

initialization;

1. define initial absolute coordinate in meters (initial GNSS values longitude_0 and latitude_0 will be extracted from the GNSS unit - in Figure 3.3, mapping stage input section; Initial GNSS values block)

$$^{absolute}\text{x}_0 = R \times \cos(\text{longitude}_0) \times \cos(\text{latitude}_0) \tag{3.9a}$$

$$^{absolute}\text{y}_0 = R \times \sin(\text{longitude}_0) \times \cos(\text{latitude}_0)\text{‘} \tag{3.9b}$$

while *pose estimation available* **do**

retrieve the estimated pose values ;

$$^{relative}\text{x}_i \leftarrow \text{estimated pose vector}.x \tag{3.10a}$$

$$^{relative}\text{y}_i \leftarrow \text{estimated pose vector}.y \tag{3.10b}$$

update the absolute cartesian values ;

$$^{absolute}\text{x}_i = ^{absolute}\text{x}_0 + ^{relative}\text{x}_i \tag{3.11a}$$

$$^{absolute}\text{y}_i = ^{absolute}\text{y}_0 + ^{relative}\text{y}_i \tag{3.11b}$$

update longitude and latitude

$$^{estimated}\text{longitude}_i = \tan^{-1}\left(\frac{y}{x}\right) \tag{3.12a}$$

$$^{estimated}\text{latitude}_i = \tan^{-1}\left(\sqrt{\frac{^{absolute}\text{x}_i^2 + ^{absolute}\text{y}_i^2}{^{absolute}\text{x}_i^2 + ^{absolute}\text{y}_i^2 - R^2}}\right) \tag{3.12b}$$

end

In the above algorithm an assumption is made that the initial value from the GNSS unit attchted in the vehicle is accurate (in terms of longitude and latitude values). The symbol R in (3.9) is the earth radius and is taken as 6,378,100 m.

3.2.2 Free-space detection

This section refers to the *Free-space detection* row in Figure 3.3. Here the term *free-space detection* stands for detecting objects in the scene and warning about the distance of that object from the vehicle. For example, an object located at a 1m distance is considered a possible danger, whereas an object at around 10m will be considered ideal. A quantitative analysis was not carried out. Instead, objects visible in the ground plane close to the vehicle were detected. The main idea was to build up a system that can be re-used in the future, and this part of the free-space detection was written in C++ under the ROS environment.

The stereo vision properties and the depth images were used in this *free-space* detection, where an algorithm based on salient object (SO) detection was implemented. A salient object in an image processing context is an object with unique attributes compared to the rest of the environment. While there are many different approaches to SO detection [41], [42], [43], the approach by [15] where the background and foreground objects are classified based on the depth information was used in this thesis.

In this algorithm, the foreground and the background objects are differentiated based on the comparison of whether creatain image regions are closer to the camera than the surrounding environment. Typically SO detection methods as suggested in the literature [44] (among the ones given previously) often work with single objects in space however, this is not the case in the practical world. Therefore, a method that allows detecting multiple salient objects in the environment needs to be implemented. The method suggested by [15] uses a segmentation-based approach to address this issue. The graph-based image segmentation [45] method implemented in the *OpenCV* library was directly used in this thesis.

As inputs for this algorithm, the left RGB image, and the depth image are given. The RGB image is fed to the predefined class to obtain the segmented image. The image is divided into N number of segments where a segment S_i (where $i = 1...N$) will define a unique region of the image. Then a unique property $F(S_i)$ is defined, which tells whether the segment S_i is foreground or background. Once the image segmentation process is finished (and N number of segmented regions are obtained), the image will be scanned from left to right, pixel by pixel, to define the property of $F(S_i)$ for each region. Based on the current pixel p_c a search window W (in this thesis, defined as 3×3) is defined. An example image window is illustrated in Figure 3.5. Algorithm 5 illustrates the method of defining the unique value of $F(S_i)$. Following notations will be used in this algorithm

- d_{p_c}: Describes the current depth value at the image location (u, v)
- w_{p_c}: Describes the current pixel in the selected search window. If d_{p_c} is (u, v), then $w_{p_c} = (u + i, v + j)$, where i, j are two variables varies from 0 to the height of the selected squared window.
- S_{p_c}: Segment value of the current pixel p_c
- $S_{w_{p_c}}$: Segment value of the current pixel in the search window W

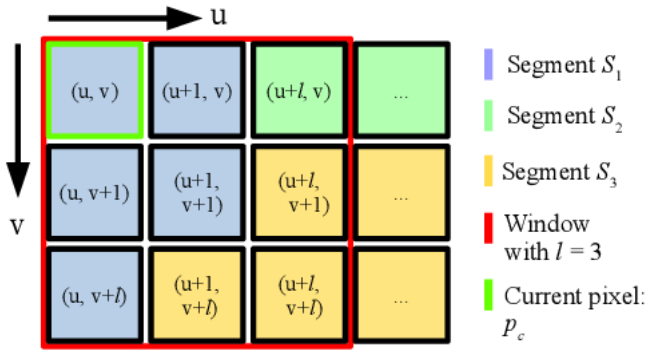


Figure 3.5: Search window of the SO algorithm [15]. Different segmented regions are defined in different colors. Here u and v is the equivalent of the x, y image coordinates in an image plane.

- m_{current}: A matrix that has same dimensions as the search window. This contains the value of the depth difference.
- M: A matrix that has the same height and width as the image. A value at (u, v) in this matrix is defined by (3.17). This value is a depth value, defined relative to its neighboring pixels.

The operation (3.15) calculates the depth difference between the current pixel and another pixel in the search window, and (3.16) calculates the difference between the current pixel and search window pixel that is furthest to the camera (have the maximum distance). These two matrices can be defined as a function of d_{p_c} and d_{p_w}. If the result of this subtraction is positive, then the current pixel p_c is said to be to be closer to the camera and hence can be labeled as foreground compared to the search window pixel w_{p_c}. If the result of this subtraction is negative, the current pixel p_c is considered as background. After processing each window search, the two matrices are summed. Matrix M (in (3.17)) updates this summed-up value for each pixel. This matrix will have the same dimensions as the original image. The property $F(S_i)$ for a segment S_i can then be calculated by taking the summation of all the distances in M that belongs to the segment S_i. A normalized vector of all the properties $F(S_i)$) is be defined as,

$$F_N(S_i) = \frac{F(S_i)}{N_i} \tag{3.13}$$

where N_i is the number of pixels in the segment S_i. A single value $F_N(S_i)$ is a value with unknown limits. Thus, a clamping process is done so that all the values are defined between 0 and 255*. Then a threshold F_{TH} is set to decide whether the segment S_i is foreground or background.

$$S_i = \begin{cases} \text{background} & F_N S(i) \leqslant F_{TH} \\ \text{foreground} & F_N S(i) > F_{TH} \end{cases} \tag{3.14}$$

The ground plane close to the vehicle can be defined as a background area (or a segment) compared to the other objects in the environment. In fact, the threshold F_{TH} is tuned so that the segment covering the ground plane close to the vehicle is chosen as

*$F_N S(i) = F_N S(i) \left(\frac{255}{F_N max}\right)$, where $F_N max$ is the maximum value of the vector F_N

Algorithm 5: Define the unique property of a segment S_i

Input Left RGB image, Depth map and the segmented image

Result: A function $F(S_i)$ that describes whether a selected segment S_i from an segmented image is a foreground region or a background region.

initialization;

1. left RGB image $\Rightarrow^{3\times u\times v}$ Image $= I$
2. obtained segmented image (graph-based segmentation) $\Rightarrow^{1\times u\times v}$ Segmented_Image $= SI$
3. get the respective depth image $\Rightarrow^{1\times u\times v}$ Depth_Image $= DI$

for *for each column* u *in* I **do**
  for *for each row* v *in* I **do**
    $p_c \leftarrow I(u, v)$;
    $d_{p_c} \leftarrow DI(u, v)$;
    $S_{p_c} \leftarrow SI(u, v)$;
    $max\_distance = 0$;
    for *width* i *of the window* W **do**
      for *height* j *of the window* W **do**
        $w_{p_c} \leftarrow I(u + i, v + j)$;
        $S_{w_{p_c}} \leftarrow SI(u + i, v + i)$;
        $m_{\text{current}} \leftarrow \emptyset$;
        $m_{\text{max}} \leftarrow \emptyset$;
        if $p_c \neq w_{p_c}$ & $S_{p_c} \neq S_{w_{p_c}}$ **then**
          $d_{p_w} \leftarrow DI(u + i, v + j)$;

$$m_{\text{current}} = d_{p_c} - d_{p_w} \tag{3.15}$$

          if $max\_distance < d_{p_w}$ **then**
            $max\_distance = d_{p_w}$;

$$m_{\text{max}} = d_{p_c} - d_{p_w} \tag{3.16}$$

          end
        end
      end
    end

$$M_{u,v} = \sum_{i=0}^{i}\sum_{j=0}^{j} m_{\text{current}_{i,j}} + \sum_{i=0}^{i}\sum_{j=0}^{j} m_{\text{max}_{i,j}} \tag{3.17}$$

  end
end

The resulting matrix M contains all the relative depth differences corresponding to each pixel of the image. Each of these values belongs to a specific segment defined in the segmented image. Then the function describing a segment $S(i)$ can be expressed as;

$$F(S_i) = \sum_{p_c \in S_i} M \tag{3.18}$$

background. A predefined area of a projected rectangle[†] is set in the ground plane. In this thesis, this is set as a static plane to simplify the process. If an object is getting closer to the vehicle, then it will mean a change in the backgroundness[‡] of that predefined area. Finally, two threshold values are defined to create three states of the current environment that is close to the vehicle. If the size of the predefined area is reduced, i.e., when objects are entering the zone, depending on the percentage of the size reduction, the three states will change to *Warning, Danger* and *Ideal*.

3.3 Feature extraction and tracking

Feature extraction and tracking is a critical component of any *visual SLAM* system. In the stereo vision context, detecting key-features in the two images (left and right) in two consecutive frames is the initial step for point triangulation to estimate the relative pose between the two frames. After the triangulation phase, with the two cameras' matrices, the current frame's pose will be estimated. Therefore, one should note that correct matches between frames are critical; otherwise, inaccurate triangulation may result in lousy pose estimation.

The next step would be to track the feature movement from frame to frame. In a frame i, the detected feature will change its image location in the next frame $i + 1$. The feature difference between the two frames or the pixel level movement would be the critical factor in estimating the pose difference. Furthermore, the location of the feature is equally important.

3.3.1 Feature extraction and matching

In *OpenVSLAM*, *ORB* (recall Section 2.2) is used as the feature detection and extraction method. For a feature detector, *ORB* performs quite well with respect to the computational time. Thus it was decided to use *ORB* as the feature detector for this thesis. However, one should note that it is more important to assign descriptors that characterize key-feature (landmark) properties, which are detected by the feature extraction method and match the correct features between two frames, rather than detect features from a frame. However, one should note that assigning accurate descriptors to characterize key-feature (landmark) properties is more important than the feature extraction process. These descriptors are responsible for the feature matching process that uses descriptor-generated properties to match landmarks between two or more frames.

Therefore, even though it was decided to use *ORB* as the feature detector, different combinations of descriptors were used to evaluate the feature extraction and matching methods further. The Table 3.1 shows the various combinations used to find a suitable descriptor for this project.

There are mainly three scenarios that affect the accuracy when creating a descriptor of an image and matching two descriptors from a *trained image* and the *query image*. Suppose the query image (in this thesis, this would be the current image) and the trained image

[†]The predefined rectangle has a static area and this area will initially be declared as background segment

[‡]The property of a segment has been the background. This property is a value that fluctuates between 0 and the threshold F_{TH}

Detector	Descriptor	Distance measure
ORB	SIFT	Euclidean
	SURF	
	ORB	Hamming
	FREAK	
	BRISK	

Table 3.1: Different feature detectors and descriptors.

(the previous image) have a significant rotational difference. In that case, this could lead to matching wrong features in between the trained image and the query image. Another factor that would affect is the scaling of the image (down or upscaling). Finally, intensity changes in the two images may also affect the accuracy in matching two descriptors.

(a) Original image

(b) Rotated image

(c) Upscaled

(d) Changed intensities

Figure 3.6: Original (also known as the training image) in Figure (a) and the three different scenarios that affect the accuracy in feature description and matching. Figure (b) illustrates the rotational scenario and Figure (c) represents the upscaled scenario. Finally, in Figure (d), the change in different intensities are shown.

In this thesis, since the utility vehicle is only moving in one plane (the *x-y plane*), rotational movements in the *x/y axis* will not occur, and the rotational scenario shown in Figure 3.6b can be ignored. One might argue that the vehicle movements' vibrations will result in rotations. However, these resulting rotations will be negligible and, therefore, can be ignored. Scaling (up or down) can be noticed if the algorithm tends to skip several

frames. For example, consider a case where images are captured when the vehicle moves towards an object that takes up to 25% image space in frame i. The same object will take more image space in frame $i+n$, where $n > 1$. However, the scale difference between two consecutive frames can be ignored because, in this project, the *synced* images are captured with an average frame rate of $10fps$, and the algorithm will not skip any frames during execution.

One can see a clear difference in figures 3.6a and 3.6d with respect to the intensity difference. These two images are consecutive images, and the reason for this sudden change in the shade is because the direction of the (sun-)light rays change with the change of vehicle direction. In image 3.6a, the vehicle was facing north, and it will slowly start turning to the east in the next few images. This results in clear changes in the intensities between two consecutive images. Therefore, one should note that the descriptor selected in this thesis should be sensitive to intensity changes. An intensity change with different combinations given in Table 3.1 was tested to find a suitable detector/descriptor combination for this project.

The task was only to change the intensity of the images. It was done using changing the intensity value by a factor chosen according to the original image's mean. First, the original image's mean intensity will be calculated, then a percentage of this value will be taken (for example, 10% of the mean). Each pixel of the chosen image will be increased (or decreased in the case of reduction) with the newly calculated value. Finally, all the values will be multiplied with a factor where the highest pixel intensity after the change is set to 255 and the lowest to 0. Results after increasing (and decreasing) pixel intensities by 30% is shown in Figure 3.7.

(a) Intensity reduced image **(b)** Original image **(c)** Intensity scaled image

Figure 3.7: Change in intensities. Figure (a) represents a 30% intensity reduction, where the mean intensity of 114.581 was reduced to a mean of 94.355. In the case of Figure (c), the mean value was increased up to 130.759.

3.3.2 Feature tracking

Feature tracking is equally as important as feature detection when it comes to visual SLAM algorithms. In theory, features that were detected at frame i will have different image locations compared to when detected at frame $i+1$. The pixel difference between features will estimate the moving direction of the vehicle between two frames. The magnitude of the *motion vector* between two consecutive frames will be proportional to the pixel difference between the features in the current frame and the previous frame.

The state estimations will be heavily dependent on this motion vector estimation in the tracking phase of the OpenVSALM or any visual SLAM algorithm. The image

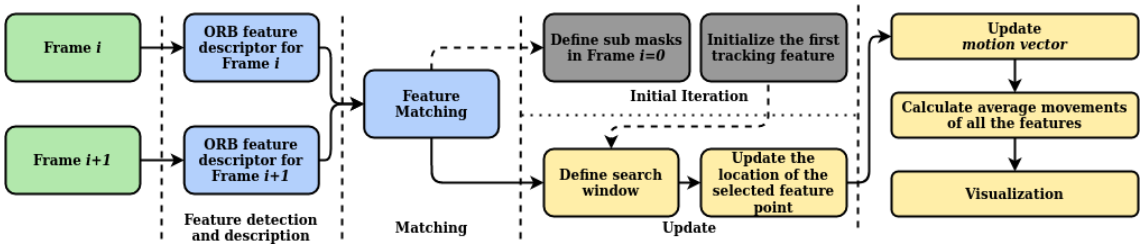


Figure 3.8: An overview of the feature tracking algorithm. Inputs of this algorithm are a sequence of images and are illustrated in the green color blocks. OpenCV implementation of ORB feature detection and matching is shown in blue color blocks. Two gray color block and the two dotted lines connecting blocks in *matching* and *update* phases are only triggered in the initial iteration. Feature tracking (only the selected feature) and update of the motion vector and the search window are shown in yellow color blocks. Visualization block includes drawing the updated motion vector for analytic purposes.

location of the detected features affects the estimated motion vector. For example, image points located in the image center section would have little movement than the image points in the two corners, especially in an eventful situation. What interesting here is that a motion vector between two frames with a broader FoV has a much higher vector length compared to the same two frames with a smaller FoV value. These two vectors have different lengths, even for the same movements, is because that the broader FoV captures more corner pixels than those of a smaller FoV. To observe this behavior, an ORB feature tracking experiment was implemented.

This experiment's objective was to see how feature points in different image locations affect the overall motion vector in a sequence of images. For this experiment, an ORB feature-based tracking algorithm was designed, adapted from the paper *Object tracking based on ORB and temporal-spatial constraint* by Wu et.el [46]. An overview of the implementation is given in Figure 3.8. For illustration purposes, the stereo vision setup was ignored in this experiment. Instead, only the left images of predefined sequences from the KITTI dataset were used.

In the original paper, an object of interest is defined before the algorithm starts. In this thesis, instead of an object, a specific feature point was tracked for a sequence of images. Initially, the image was divided into three sections (3 image masks), the left, the center, and the right. Three feature points from the three masks will be selected in the initial stage, and these three points will be tracked throughout the selected image sequence to calculate three motion vectors for the three masks. Firstly, to select the initial point for each mask, all the said mask features will be generated. The feature point closest to that given mask's center point will be selected as the initial feature point to track. An example image with the three masks is shown in Figure 3.9.

Once the initial points for each mask are defined, three rectangular windows will be defined around the initial points with an initial height (h) and width (w). Let one of those rectangles be $Rec_i(x_i, y_i, w, h)$, in the initial frame $i = 0$, where x_i, y_i represents the initial feature point. Next, a search window for the next frame will be defined according to that said feature point's motion continuity. And the search window for the frame $i + 1$

Figure 3.9: Image with predefined masks. In this example, 50% of the image width is allocated as the center part of the image, whereas 25% each is given for the left and the right regions of the image.

is defined as $S_{i+1}(u_{i+1}, v_{i+1}, w^{'}, h^{'})$ where,

$$(u_{i+1}, v_{i+1}) = (x_i, y_i) + \overrightarrow{M_i} \tag{3.19a}$$

$$w^{'} = w + \alpha \tag{3.19b}$$

$$h^{'} = h + \alpha \tag{3.19c}$$

Here vector $\overrightarrow{M}$ is the newly calculated motion vector, and α is a constant that can be changed according to the feature detection in the selected search window. In the initial case, the motion vector is set to $(0, 0)$, which will be updated in the next iteration.

Once the search window is defined and all the feature points in that search window are detected, then a matching between the window Rec_i in frame i and the search window S_{i+1} in the frame $i+1$ will be performed based on the ORB descriptor obtained at feature detection and description stage (refer to Figure 3.8). This process will detect the initially selected keypoint. Let MK_i be all the matched points between frame i and frame $i+1$ (in the selected mask) that are in frame i, with N number of matched points (which is a subset of all the keypoints detected in frame i), Then the geometric center C_i for MK_i for the selected mask can be formulated as,

$$C_i = \frac{1}{N} \sum_{j=1}^{N} MK_i[j] \tag{3.20}$$

Similarly, a subset set of matched keypoints in frame $i+1$ can be defined as MK_{i+1}, where its geometric center is defined as, C_{i+1}. Then the motion vector $\overrightarrow{M_{i+1}}$ is calculated as $\overrightarrow{M_{i+1}} = C_{i+1} - C_i$. The rectangle for the next frame $i+1$ (which will be compared with $i+2$) can be defined as $Rec_{i+1}(x_{i+1}, y_{i+1}, w, h)$, where

$$(x_{i+1}, y_{i+1}) = (x_i, y_i) + \overrightarrow{M_i} \tag{3.21}$$

By repeating this process for all three masks, one would expect to obtain three motion vectors.

3.4 Effects of FoV in VSLAM

In this thesis, the estimated pose was evaluated against an open-source dataset called KITTI dataset. The images included in KITTI has an FoV close to 82°, which is around a 70% increase compared to the dataset addressed in this project, which has an FoV around 23°. The FoV can be calculated by (3.22), where FoV value in radians will be calculated, given the focal length (f) of the camera lens and the camera sensor width ($d_{horizontal}$). Note that the vehicle is moving in the ground plane, resulting in no rotations around the pitch or the yaw axes. This leads to negligible tracked feature movement in the vertical direction of the image. Therefore one can ignore the effect from the vertical FoV.

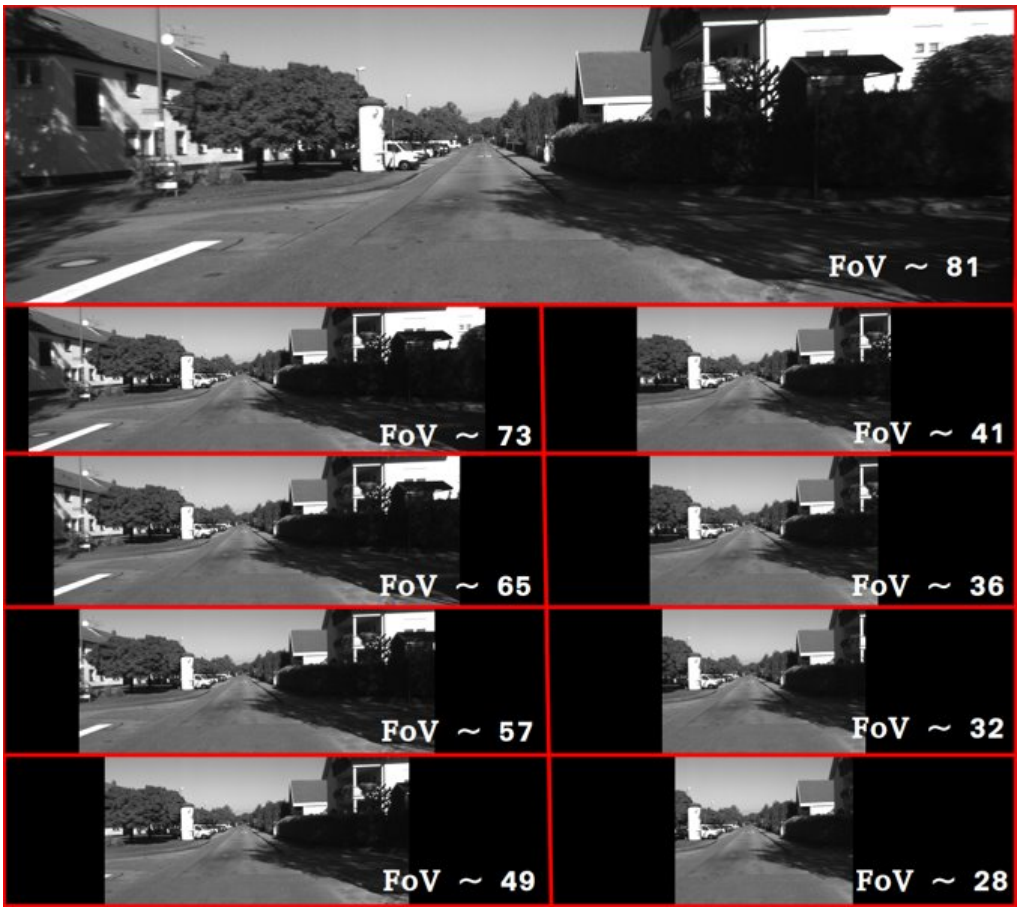


Figure 3.10: FoV simulation masks - *KITTI Odometry* dataset.

$$\text{FoV} = 2\tan^{-1}\left(\frac{d_{horizontal}}{2f}\right) \tag{3.22}$$

A hypothesis was made that the *FoV has an impact on estimating the pose of the camera*. To test this hypothesis, a simulation was carried out using the *KITTI Odometry* dataset. Without changing the actual resolution of the image, which was 1220×370, the image color of the right and left sides were changed into *black* color. When changing the pixel values one should note that this was done for an entire column. The number of columns from the left and right side should be the same and this number was calculated

as a percentage from the maximum width of the image (or the number of columns in the image). Another key thing is that when changing the pixel value of the selected columns, the color will not matter as long as all the selected columns have the same color.

The main idea behind creating this temporary mask is to force the ORB algorithm to detect features only from the middle part of the image. This way, one would be able to simulate the FoV reduction effect without actually changing the camera parameters. Even though the image's actual width is not reduced, $d_{horizontal}$ in (3.22) can be changed since the image visibility is forced to a particular section of the image. The masks were initially starting with 100% visibility, and then the columns were reduced with regular intervals until it reached a point where the FoV can be simulated as 21° degrees (75% from the original width). An example FoV reduction is given in Figure 3.10.

It was observed that the said hypothesis regarding the FoV effect on visual SLAM algorithms was correct. The obtained results of this simulated experiment are presented in Section 4.3. This report further discusses and evaluate the reason behind this mentioned behavior in Section 4.4.2 according to an experiment designed as described in Section 3.3.2.

3.5 Image stitching

The FoV of the *Barmark* dataset, as mentioned before, is around 23°, and one might argue about improving the FoV via merging the two stereo images. This is a valid point; however, if the images are merged, then the advantage of using stereo vision to get an accurate depth estimate would be lost. Hence in the VSLAM algorithm, one would face the monocular drift.

However, an attempt was made to merge the left and right images to see whether it increased the image FoV to a reasonable amount so that the algorithm can be shifted to *monocular SLAM* to achieve acceptable results. The initial attempt of image merging was an offline method known as the *robust homography estimation and image stitching*. As described in the appendix Section A.3 (A.24), a homography matrix can be defined such that it defines a relationship between two sets of image points that are projected from the same 3D points. This thesis defines these two sets of image points by ORB feature extraction and matching (instead of stereo correspondence).

Once the matched image points are obtained, a DTL equation was constructed to find the homography between the two images point vectors. If the two vectors (which include 2D vectors) are define as x_l and x_r (to represent left and right images respectively) then a relationship between two image point vectors can be defined as $\lambda x_l = H x_r$. To estimate the homography matrix, which has 8 DoF one need at least 4 points. Initially, 4 points will be selected randomly, and the DLT equation given in (3.23) will be constructed. In (3.23), the values c_1 to c_4 represent the 4 point correspondences. The point in the left image is given by $x_l = \begin{bmatrix} x_{11} & x_{12} & x_{13} \end{bmatrix}$ and the point in the right image is given by $x_r = \begin{bmatrix} x_{21} & x_{22} & x_{23} \end{bmatrix}$*.

*Please note that in the actual implementation, the vectors x_l and x_r are defined as homogeneous vectors hence both the values x_{13} and x_{23} are equal to one.

$$
[H]\begin{cases} c_1 \\ c_2 \\ c_3 \\ c_4 \end{cases}
\underbrace{\begin{bmatrix}
x_{11} & x_{12} & x_{13} & 0 & 0 & 0 & 0 & 0 & 0 & -x_{21} & 0 & 0 & 0 \\
0 & 0 & 0 & x_{11} & x_{12} & x_{13} & 0 & 0 & 0 & -x_{22} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & x_{11} & x_{12} & x_{13} & -x_{23} & 0 & 0 & 0 \\
x_{11} & x_{12} & x_{13} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -x_{21} & 0 & 0 \\
0 & 0 & 0 & x_{11} & x_{12} & x_{13} & 0 & 0 & 0 & 0 & -x_{22} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & x_{11} & x_{12} & x_{13} & 0 & -x_{23} & 0 & 0 \\
x_{11} & x_{12} & x_{13} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -x_{21} & 0 \\
0 & 0 & 0 & x_{11} & x_{12} & x_{13} & 0 & 0 & 0 & 0 & 0 & -x_{22} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & x_{11} & x_{12} & x_{13} & 0 & 0 & -x_{23} & 0 \\
x_{11} & x_{12} & x_{13} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -x_{21} \\
0 & 0 & 0 & x_{11} & x_{12} & x_{13} & 0 & 0 & 0 & 0 & 0 & 0 & -x_{22} \\
0 & 0 & 0 & 0 & 0 & 0 & x_{11} & x_{12} & x_{13} & 0 & 0 & 0 & -x_{23}
\end{bmatrix}}_{:=M}
\underbrace{\begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ \vdots \\ \vdots \\ \vdots \\ H_{33} \\ \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \end{bmatrix}}_{:=v}
= \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 0 \end{bmatrix}
\tag{3.23}
$$

Then solving M using *SVD*, one gets the solutions to the v vector. The eigenvector corresponding to the smallest singular value will give the optimal solution for the vector given in v, which is the expected homography matrix between the two image point vectors when rearranged to a matrix.

An optimal solution obtained from the method mentioned above is only optimal for the four points that were selected randomly. To obtained an actual optimal solution, an iterative estimation method, *RANSAC* or Random sample consensus [47] method was used. Once estimation is finished using random 4 points, a projection error can be calculated using all the matched points with the estimated homograpy matrix.

$$
\text{Projection error} = \sqrt{\sum_{i=0}^{N} \left(x_i^{left} - {}^{estimated}\mathrm{H} x_i^{right} \right)^2} \tag{3.24}
$$

Ideally, the projection error calculated by (3.24) should be zero if the homography matrix is optimal. Since it will not be zero, a set of inlier points can be defined with a user-defined threshold. Every point that does not fall into the inlier category is defined as an outlier. Initially, the number of inliers is set to zero and is updated with each iteration, and after a given number of iterations, a homography with the maximum inliers will be selected as the optimal homography matrix. Using the calculated optimal homography, the right image can be transformed into the left image, and the results are given in Figure 3.11c.

As shown in Figure 3.11c, even after the image merges, the increment in FoV is negligible. In fact, it was calculated that the merged image only increased the FoV by 1°. Therefore, it is clear that for the *BARMARK* dataset, increasing the FoV using this image stitching method is not feasible.

(a) Left image

(b) Right image

(c) Combined image

Figure 3.11: Image stitching results. Figure (c) represents the merged image of left and right. The black boards on the corners represent the internal rotations that can be seen in the image planes.

4

Results & Discussion

In this chapter, the pose estimation results and analysis will be presented. In SLAM *front-end* as described in Section 2.3, pre-image processing is a crucial step. This step includes image synchronization (described in Section 3.2) and obtaining correct camera parameters and depth estimations. This chapter presents the initial image calibration results and the estimated depth results. Next the pose estimation results from the *ROS* integrated *OpenVSLAM* is presented. The effect of FoV in pose estimation and *VSLAM* in general is presented using the data from *KITTI Odometry* dataset. This is followed by feature extraction and tracking analysis. Finally, the chapter ends with a qualitative analysis of free-space detection.

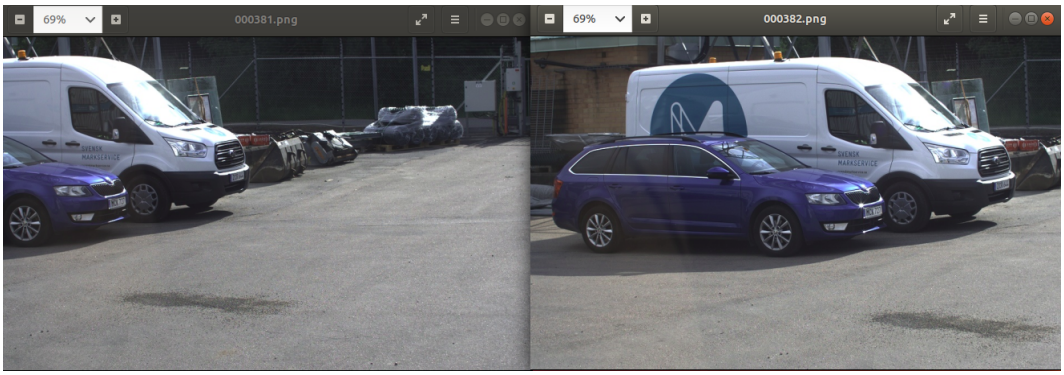
4.1 Image preprocessing - Implementation phase

As mentioned in Chapter 3, the pre-processing image part is very crucial. Due to technical constraints, it was not possible to test this version of implementation in the vehicle itself, and as a solution, a *ros-bag* file was recorded with compressed images. Of course, these compressed images save memory when recording the images that otherwise would cost much memory even for a small amount of data collection. However, some of the applications used in this thesis cannot be directly accessed by these compressed images, such as python rosbag API (which is used to read rosbag files) and rqt (application used for visualization). Hence an image decompression was needed. In an ideal case, the decompression methods suggested in Section 3.1.1 would be able to decompress the entire image stream without any interruption; this is, of course, not the case in a practical implementation. Since the images have a high resolution and are needed to be recorded at a higher frequency (in this thesis, initial images have frequencies close to 50Hz to 55 Hz), the decompression method tends to miss some frames. As a solution, the bag files were fed to the decompression method at a slower rate*. One should note that this frequency decrease is performed only to make sure that all the frames are correctly decompressed. In fact, one would be able to rerun the newly recorded bag file at any desired rate afterward if one wishes to simulate the real-world frequency rates.

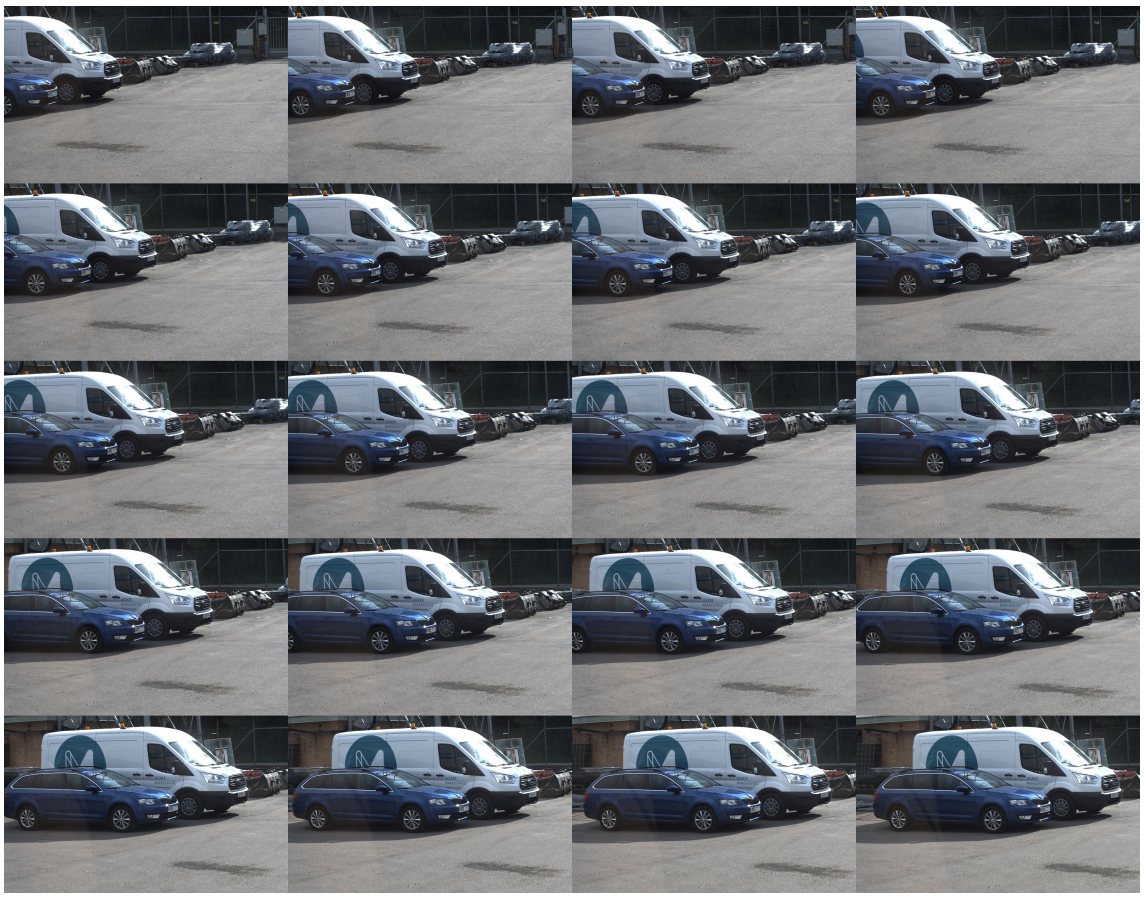
The camera's frequency or, in the computer vision context, the fps of the camera is also an important point that needs attention in real-world applications. In this thesis, mainly two ranges of fps values will be addressed. The first one is when recording the data. In this case, the frequencies need to have a higher rate, otherwise, when performing

*The value for this slower frequency depends on the system that analyzes these rosbag files. Therefore, to make sure that all the images are correctly decompressed and since no real-time performance was needed, a frequency of 250Hz (four images per second) was used in this thesis in an 8GB RAM system.

the *image synchronization,* process (described in Section 3.1.1), a significant amount of image data will be lost. An example of such a scenario is given in Figure 4.1.



(a)

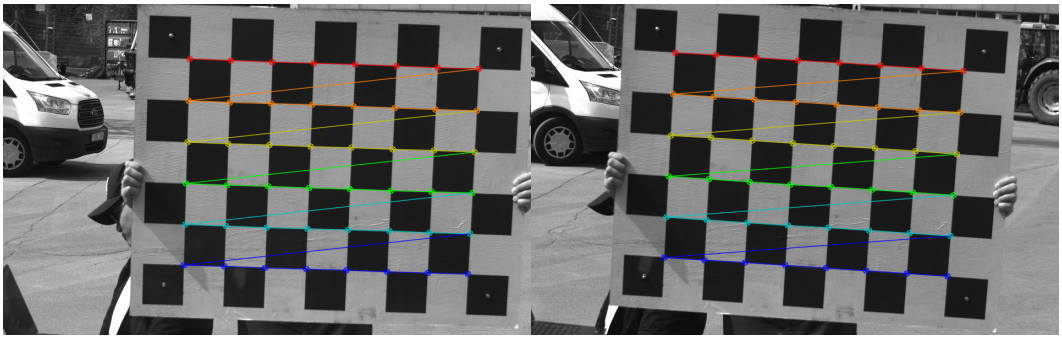


(b)

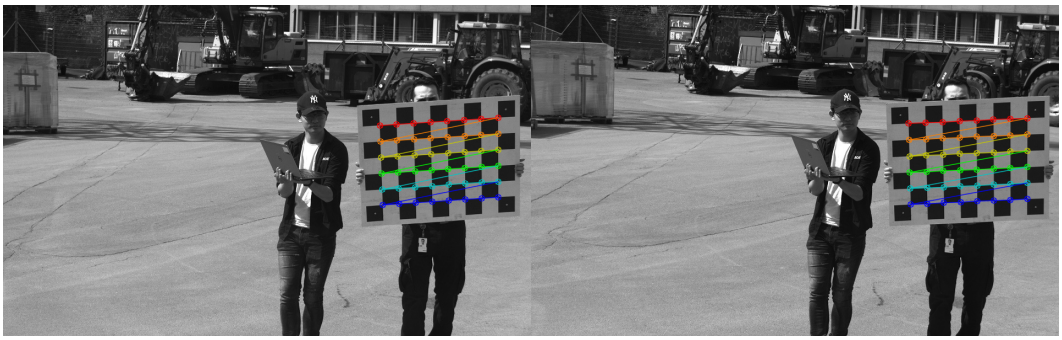
Figure 4.1: Images captured at different fps values. Figure (a) shows images captured at a very low fps value (around 5 to 6 fps), which resulted in losing lots of data between the frames. This losing frames can be troublesome, especially in an eventful situation. Figure (b) shows images recorded at a higher rate (around 18 to 20 fps). At these rates, after the image synchronization process, even if a few frames are lost due to not been synched, the algorithm will still be able to capture a reasonable amount of image frames to proceed to the VSLAM algorithm.

In the second case, the range of fps needs to be much lower than the above-mentioned high range fps. This low range of fps values is used when feeding image data to the *OpenVSLAM* library. Having lower frequencies is beneficial since it will cause less stress to the system, especially if it has insufficient RAM. A workaround solution to force having a lower fps value is to skip some frames when running the algorithm. However, the trade-off here is that when skipping too many frames, the algorithm tends to lose its feature tracking, ending up breaking the algorithm altogether. With several trial and error runs, 10 fps (recall the initial rate when recording the images were around 18 to 20 fps) was selected as the feasible frequency to run without any frames skipping for a computer with 8 GB of RAM.

The next important part of image pre-processing is obtaining the accurate camera parameters, which helps in the depth estimation process and the pose estimation process. As mentioned in Algorithms 2, the initial step is to detect and define feature points in the checkerboard with different views. An example of such detection is given in Figure 4.2.

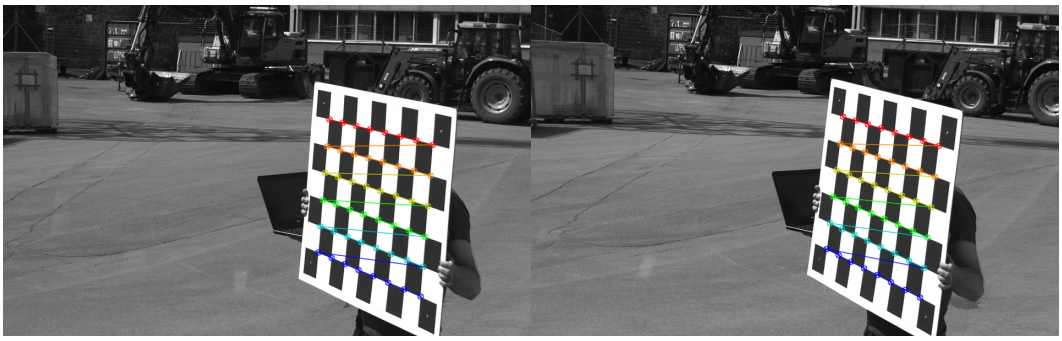


(a) Close-up



(b) Relatively far from the camera

(c) Rotated



(d) Tilted

Figure 4.2: Checkerboard corner detection of stereo images. The entire checkerboard must be visible in both the left and the right images. Different colors for different rows help track rotations in the checkerboard. In this thesis, the dimension of the board is 8×6 and all its 3D locations are defined according to (3.1).

As described in Algorithm 2, one would then be able to perform camera calibration (either for both cameras at the same time - offline method or online) using prebuilt *OpenCV* functions. Even though one would theoretically only need three images, it is recommended to use as many images as possible with different variations, as shown in Figure 4.2. Since the estimation is based on the least-squares method and the quality of the estimation relies on its data, in this case, the homography matrices that are estimated separately for each different image (recall Section 2.1.2.2). The newly estimated camera parameters can be evaluated using the reprojection algorithm described in Algorithm 3. In this thesis, the best average reprojection error achieved was 2.43 pixels; this is acceptable since the entire image has a resolution of 960×600.

This calibration process takes a considerable amount of time since the matrix that it needs to solve in the form of a least-squares problem is quite large. Assuming that one would use at last 15 images (as recommended in the OpenCV*), the matrix V given in (A.18) will be a matrix with 30×6 dimension. When using the offline stereo calibration method, a choice can be made regarding the number of images since it uses pre-recorded images. However, this assumes that all these images have perfect visibility of the checkerboard corners that will be detected using the existing OpenCV functions. This is often not the case in a practical situation, and most of the time, one would be left with fewer images than the initial number of images fed to the algorithm after the *findChessboardCorners* function.

In the online method, this issue of not having adequate images is not a problem. However, the calibration process is done in parallel while capturing the images, and this makes this online process take a lot more time than that of the offline method. In the initial round of collecting images, the calibration process will start after acquiring a sufficient number of images. Still, since the capturing process is also processing in a parallel thread, the next run's number of detected images can be quite big. This value often reached 100 images. Assuming that the initial calibration takes about 20s (which was the average time

*OpenCV camera calibration tutorial, `https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html`

to solve with 15 images), and the synchronized images are captured at 10 fps. Then for 20 seconds, the saved images with checkerboard corners would reach close to 200. In the online method, the reprojection error for each calibration iteration will also be calculated in real-time.

As described, both the procedures have their trade-offs. However, out of the two approaches, it is suggested to use offline calibration with adequate images until it reaches an acceptable reprojection error value. (This value can be relative to different resolutions, but in this thesis, the average of 2.5 was set as a threshold given the high resolution of the images).

With calibration parameters available for the cameras, the next step would be to perform stereo rectification (as described in Section 2.1.3.1). Figure 4.3, illustrates the images after rectification. As shown in the figure, once the rectification is performed, the images will have some rotations, and the edges of the images may disappear. The matrix obtained after rectification will then align the image plane with the green lines shown in Figure 4.3.

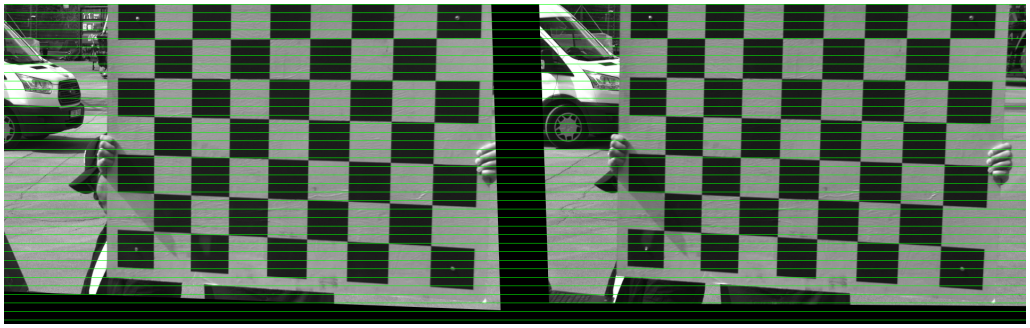
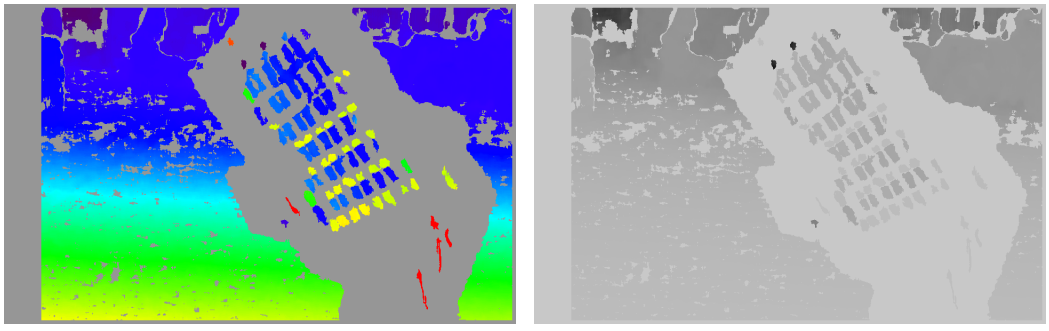


Figure 4.3: Stereo rectification process. Green lines can be viewed as the epipolar lines described in Section 2.1.3.1.

After stereo rectification, one can perform a stereo correspondence to obtain a given scene's disparity map. This disparity map can then be used to estimate the scene's depth, as shown in Figure 4.4.



(a) Disparity map **(b)** Depth map

Figure 4.4: Disparity and depth maps. Gray areas in the disparity map represent pixels that have difficulty finding its corresponding value in the other image. The depth map is generated using the left image and the disparity map.

4.2 Trajectory and Heading Estimation

The trajectory estimation includes the estimation of the x and y coordinates in a cartesian plane. As described in Chapter 3, in the thesis, the changes in the z-axis in the world frame was ignored since the vehicle only moves in the x/y plane. Two error criteria RSE and ATE (recall Section 3.2.1) are used in this thesis to evaluate the algorithm's performance. The *BARMARK* dataset, which was captured by the utility vehicle equipped with two cameras, has a length of approximately 110s (close to 2300×2 images after the image synchronization step), and it covers an area around $30m^2$.

4.2.1 Trajectory estimation

The ground truth pose was estimated using accurate GNSS data after a conversion between the absolute position in meters to a relative position. The data was collected to resemble an urban area with many vehicles and a significant amount of turns (or eventful situations). The pose estimation results were plotted against the actual ground truth as illustrated in Figure 4.5. One can see that the estimated trajectory tends to have huge errors compared to the ground truth. The errors are significant, especially when the vehicle is taking a turn or as expressed in previous chapters in an *eventful situation*, where the feature changes are quite substantial in consecutive image frames. These errors will be accumulated even though the vehicle is moving in a straight line. In this dataset, a turn occurred at the beginning of the sequence, which immediately affects the entire estimation. The timestamps also expressed that the estimates were made quite early compared to the ground truth.

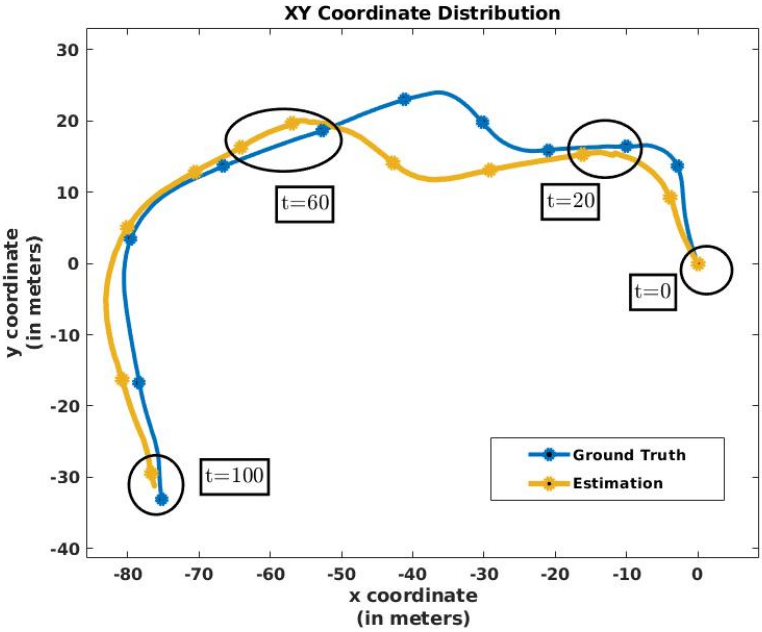


Figure 4.5: Trajectory estimation against the ground truth values obtained from accurate GNSS data. The timestamp data labels and the respective ellipses indicate the cumulative error increment in the algorithm with an inaccurate estimation. The markers in the two graphs are spaced with 10s time intervals.

The *RSE* values for each point were also calculated (recall Section 3.2.1), and the results are shown in Figure 4.6. One can see that the errors tend to increase at time instances where an eventful situation is taking place. The maximum *RSE* error between the estimate and the actual value is 12.26m, which occurred around the time instance 48s. This error occurs after two major turns in the trajectory, which further shows the cumulativeness of the pose estimation errors.

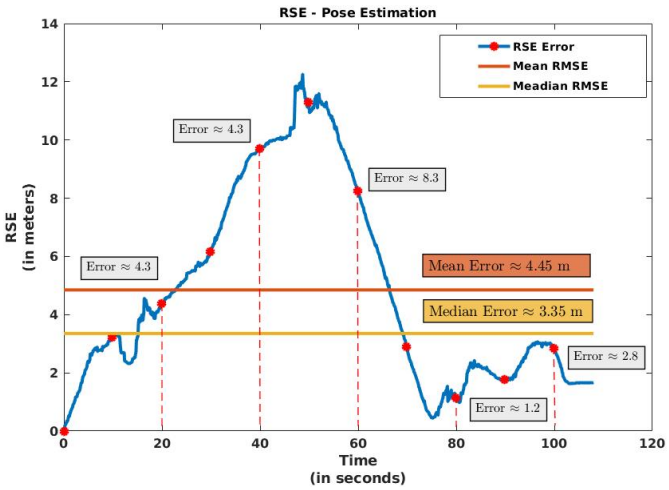


Figure 4.6: RSE in trajectory estimations. The mean of RSE (RMSE) was around 4.45m and the median was around 3.35m in the translational errors with respect of the ground truth.

The *ATE* (recall Section 3.2.1), is the same method used in many VSLAM papers for error evaluation in trajectory estimations. The *ATE* between the above two trajectories shown in Figure 4.5 was also evaluated and the results are illustrated in Figure 4.7. While RSE provides an overview of the error behavior for each time instance, ATE shows the actual deviation from the ground truth in terms of distances.

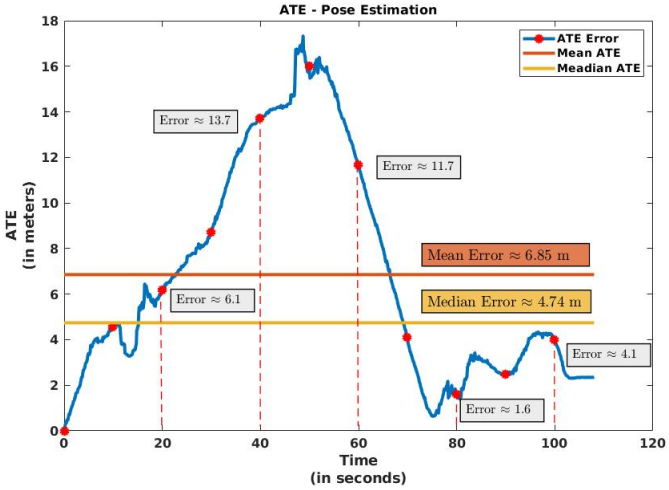


Figure 4.7: ATE in trajectory estimations. The mean ATE was around 6.85m and the median was around 4.47m in the translational errors with respect to the ground truth.

4.2.2 Heading estimation

In pose estimation, one should note that not only the translation but also the orientation in terms of the rotations around *roll, pitch* and *yaw* axes are essential. In this thesis, since the vehicular movement is only a planar motion in the *xy*-plane, only the *yaw angle* in the world coordinate system were considered to be important. The *heading direction* of the vehicle was, therefore, defined in terms of changes in *yaw angle*. Figure 4.8 illustrates the ground truth headings and the estimated headings.

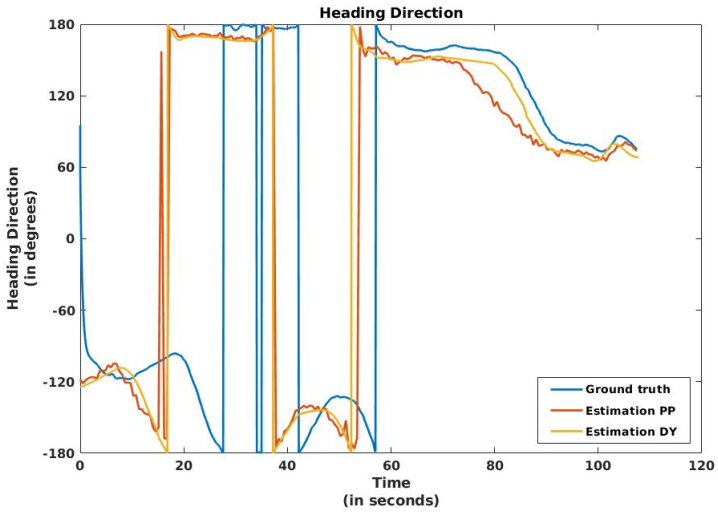


Figure 4.8: Heading direction estimation in degrees. Two methods were used to estimate the heading angle, the *previous point* method denoted as *Estimation PP* and the *direct yaw* angle denoted as *Estimation DY*.

The current vehicle's coordinate system is defined from $-180°$ to $180°$, this can sometimes be misleading to readers. For example, around the $18s$ mark in Figure 4.8 one can see a sudden error increase (close to full $360°$). In fact, this is just an error of around $5°$, and the misleading interpretation is because the error fluctuates between two different quadrants. Therefore, to obtain a better overview of the error, the angles were converted to $0°$ and $360°$ degrees from $-180°$ to $180°$. The results are shown in Figure 4.9. The *RMS* error of the two heading estimation methods, *PP* and *DY* (recall Section 3.2.1) are illustrated in Figure 4.10.

Regardless of the two estimations having a relatively similar shape, one can see that vision pose estimation (both methods *PP* and *DY*) tend to estimate its location quite early compared to the ground truth. As one can notice, from the two methods used in this thesis, *DY* seems to have a much smoother curve compared to that of the *PP* method. The reason behind this is the way that *PP* is calculated. When estimating the heading's direction vector, the difference between two points is taken, and then to calculate the direction, the inverse tangent of the distance vector is calculated. The problem with inverse tangents is that the solution tends to go to infinity when the angle (in this case, the resulting distance vector) is close to zero. Unfortunately, this is often the case with pose estimation. Once the camera pose is estimated, the URDF node publishes a new pose estimation (approximately) at a rate of 10Hz. This results in receiving points that are

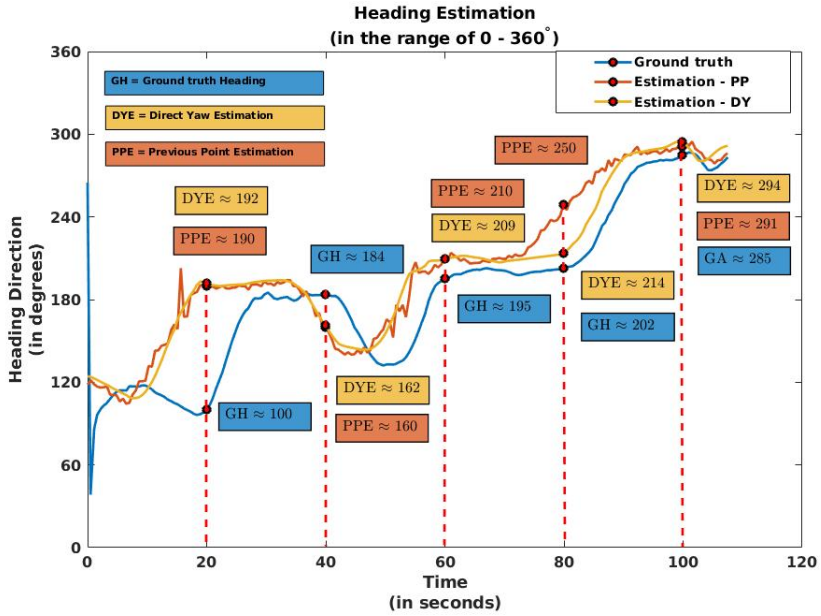


Figure 4.9: Heading direction estimation in degrees. The limits are defined in between 0° and 360°. Markers that are denoted have a time interval of 20s.

very close to each other in consecutive iterations, resulting in a relatively smaller distance difference. A threshold was added to avoid negligible distance. However, this results in a jerky behavior in the end estimations, as shown in Figure 4.9. Therefore from these results, it is suggested that estimating the vehicular heading using DY is better compared to PP.

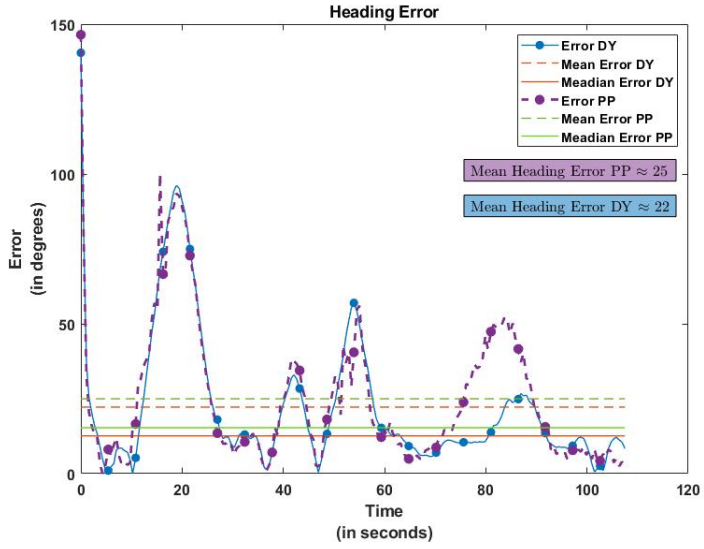


Figure 4.10: Heading estimation error in degrees. The mean and the median error values obtained from the *PP* method were (approximately) 25° and 15° respectively, while from *DY* the values were (approximately) 22° and 12° respectively.

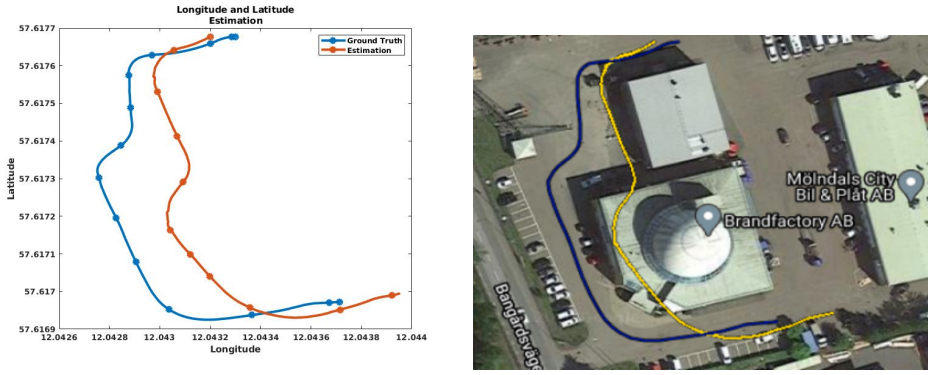
4.2.3 GNSS estimation

GNSS estimations or the longitude and latitude estimations were done according to the algorithm described in Section 3.2.1 (Algorithm 4). The two expressions of (3.9) are modified to:

$$^{absolute}\mathrm{x}_0 = (R+N) * \cos(\mathrm{longitude}_0) * \cos(\mathrm{latitude}_0) \tag{4.1a}$$

$$^{absolute}\mathrm{y}_0 = (R+N) * \sin(\mathrm{longitude}_0) * \cos(\mathrm{latitude}_0)\text{‘} \tag{4.1b}$$

where N is the altitude value, obtaining the altitude values was not possible in this thesis. While various methods estimate GNSS values from a relative position, all of these methods are accurate only to a small area. Furthermore, the value for R used in this thesis is the Earth radius at the equator and was not corrected to the estimation's exact location. Therefore the overall error that occurred with the estimations could also be affected by this value. In mathematical terms, the errors are relatively insignificant (occurs in values of 10^{-4}); however, the errors were relatively high in an actual map.



(a) **(b)**

Figure 4.11: GNSS Estimations. Figure 4.11a illustrates the difference between the ground truth GNSS values and the estimated GNSS values. Figure 4.11b, it shows the estimation in the real map in longitude and latitude.

4.3 Effect of FoV in pose estimation

As mentioned in Section 3.4, a hypothesis was created stating that the FoV has an impact on estimating the pose of the camera*. Images from the KITTI dataset was used with reduced FoV to test this hypothesis. Before looking into the results from reduced FoV, one should look at an actual comparison between the two datasets' images. Figure 4.12 illustrates how the two cameras from the two datasets see the environment. Note that this comparison between the two datasets is just qualitative. It is assumed that both the images were taken at approximately the same distance to vehicles in the image. Clearly, the images from the *KITTI* dataset can capture a much bigger surrounding area than the *BARMARK* dataset.

*In this thesis the goal is to estimate the vehicles' pose, which is basically a transformed version of the camera pose. Therefore analyzing the effects of camera pose estimation is imporatant

(a) KITTI dataset

(b) BARMARK dataset

Figure 4.12: KITTI dataset vs BARMARK dataset. The KITTI dataset's images shown in 4.12a have a resolution of 1220×370 with an FoV of close to 82°. BARMARK images, shown in 4.12b, on the other hand, have a much higher resolution of 960×600 but with FoV around 23°.

One could argue that increasing the FoV of a camera will also reduce the image's quality. Which, of course, is a valid point. For example, when looking at the two cars (which approximately have the same horizontal length) shown in Figure 4.12, they are represented in different pixel lengths, which means one would see more detailed images with low FoV cameras. However, acquiring images with high quality is not the priority in VSLAM. The low resolution affects the feature detection algorithms, but since this project operates in urban environments, detecting features will not be a problem. Covering more area of the surroundings in a single iteration is more important.

The results of the estimations are given in Figure 4.13. A noticeable error increment with FoV reduction can be seen when analyzing those figures. The errors from the pose estimation are cumulative, meaning that if a wrong estimation is calculated at i, the error obtained from frame i would add up in the next frame $i + 1$ and all the future frames in the sequence. ATE and the cumulative errors calculated for the trajectory shown in Figure 4.13 are illustrated in Figure 4.14.

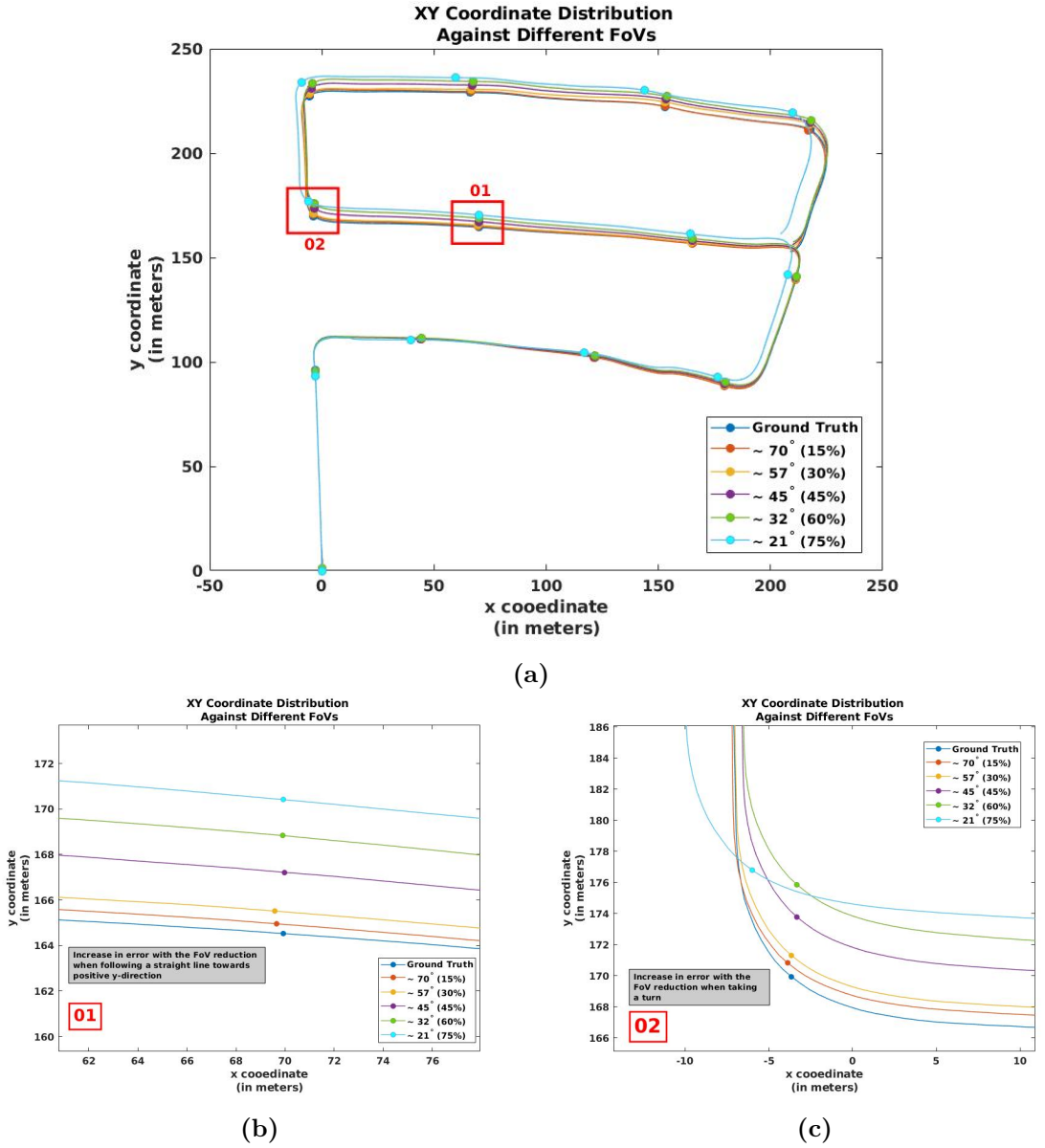
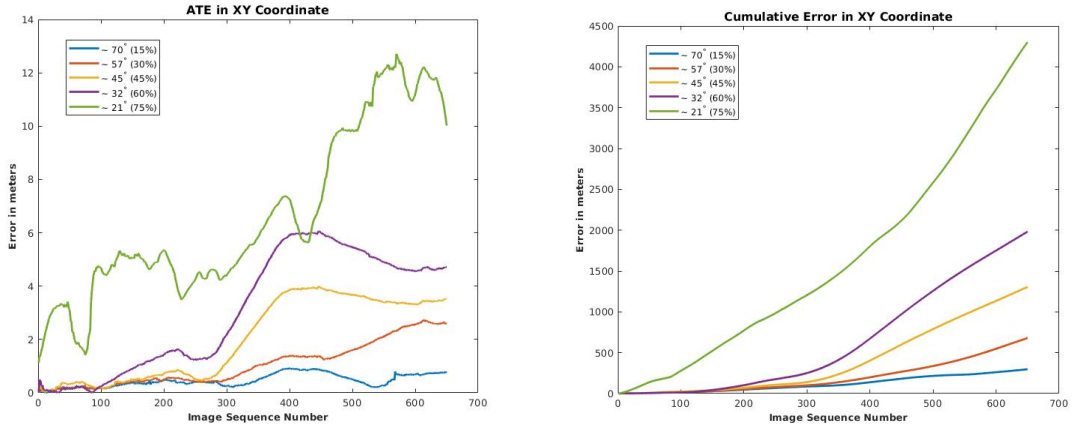


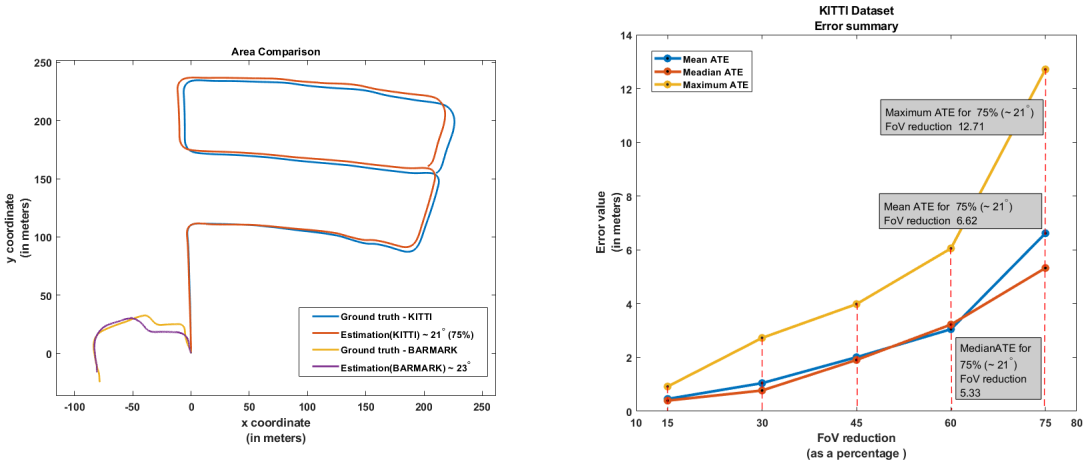
Figure 4.13: Pose estimation with different FoVs. As described in Section 3.4, a manipulated dataset with different FoVs of the same dataset from KITTI was used. Figure (a) illustrates the entire map of an area of approximately $250m^2$. Zoomed-in regions of the same figure are shown in (b) and (c), which shows straight-line motion and a turning scenario respectively. In turning movement, the worst estimation, which has an FoV around 21°, tends to estimate its pose earlier compared to other FoVs. The same effect can be seen in Figure 4.5.

When the vehicle moves in a straight line, the effects seem to be low compared to a scenario of the "vehicle is taking a turn" or, as mentioned in this thesis, in an eventful situation. The reason for this behavior is described in Section 4.4.2. An area comparison of the two datasets and a summary of the errors in ATE with different FoVs are given in Figure 4.15



(a) ATE in pose estimations **(b)** Cumulative error

Figure 4.14: ATE and cumulative error in pose estimation associate with different FoV values.



(a) Area comparison **(b)** Error summary

Figure 4.15: Area comparison between the two datasets (*KITTI* and *BARMARK*) and the error summary associated with different FoVs in the KITTI dataset. BARMARK covers an area close to about $30m^2$ and KITTI covers an area approximately around $250m^2$.

4.4 Feature extraction and tracking

As described in section 3.3.1, features plays a major role in any visual SLAM algorithm. Detecting features unique to a given environment and accurately matching those features between different perspectives of the same scene decides the quality of estimations in visual SLAM. This section presents a performance overview of different feature descriptors with different intensity conditions and a comprehensive analysis of how feature tracking gets affected by FoVs.

4.4.1 Feature description and matching

In this thesis, instead of the feature extractor, the performance of the feature descriptor was prioritized. An analytical performance based on the correct matches against different intensities is considered when choosing the accurate descriptor. A maximum of 2000 matches was considered in this analysis. Usually, it is easy to obtain lots of feature points in an urban area due to the environment's complexity. Therefore, it was not a problem to get 2000 feature points in the feature detection process. Then four different descriptors were created using the *ORB, BRISK, SIFT* and *SURF*. The feature descriptor *FRISK* was not evaluated due to implementation issues related to the OpenCV library in the ROS environment. Finally, descriptor matching was carried out according to the two different distance matching techniques (recall Table 3.1).

The next step is to distinguish suitable matches from bad matches. First, the distances between all matched point pairs are calculated, and the minimum of these values was selected. If the distance between two matched points is greater than twice the minimum distance, that matched, point pair was considered a wrong match. However, the minimum distance will sometimes be very small, so an empirical value of 20 was the lower limit. This would divide the matched point pairs into two groups, the correct matches and the wrong matches. Then, a percentage value is calculated correctly matched points percentage after the descriptor matching process.

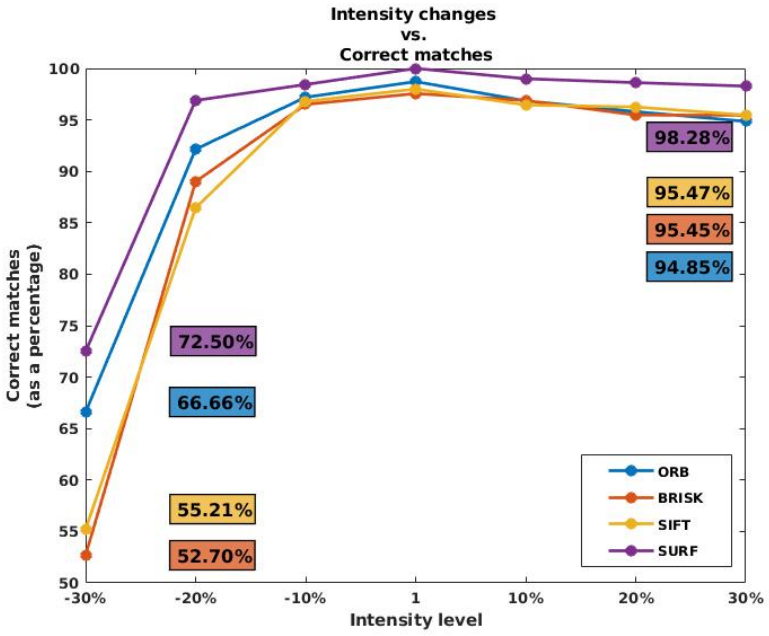


Figure 4.16: Intensity changes versus correct matches.ORB was always used as the detector in the detector/descriptor combination with different descriptors. The percentages in the two corners illustrate the different descriptor performance with the most challenging intensity changes.

Detector/ Descriptor combination	Detect time(ms)	Descriptor time(ms)	Matching time(ms)	Total time(ms)
ORB/ORB	2.6	2.6	4.4	9.7
ORB/BRISK	2.8	5.5	3.3	11.6
ORB/SIFT	2.5	86.6	10.9	100.4
ORB/SURF	2.6	90.4	5.8	98.8

Table 4.1: Different feature detectors and descriptors. Only the ORB detector was used in this experiment, hence the similar detection time for all the detector/descriptor pairs. Out of the two different distance measurement methods used in matching, Ecludian distance measurement and matching methods (used in ORB and BRISK) have higher computational time compared to Hamming distance measurement methods (used in SIFT and SURF, recall Table 3.1). Note that these times are approximately measured using C++14 libraries.

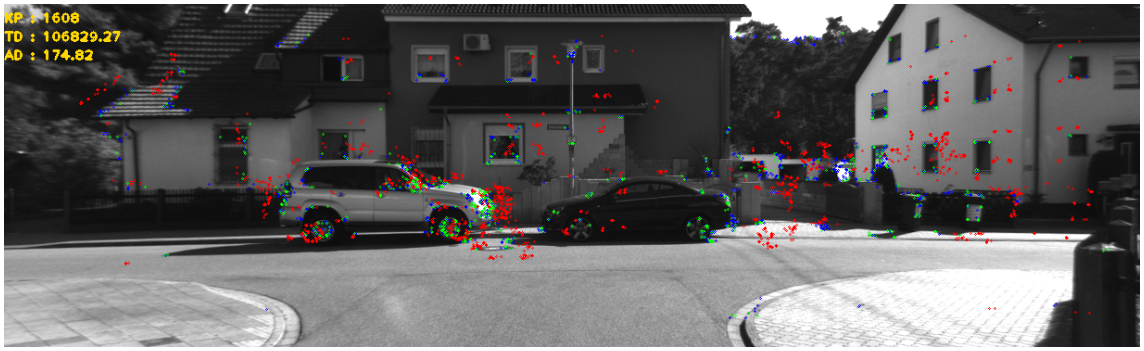
As can be seen in Figure 4.16, the *ORB/SURF* combination provides better robustness over other combinations, with correct matches percentage of 98.28% with 30% intensity increase. Even in intensity decrease, when all the other combinations fail, *ORB/-SURF* reaches to 72.50% accuracy with 30% intensity decrease. However, after analyzing the computational time, the method *ORB/SURF* was shown not to be feasible to run in real-time (refer Table 4.1). From the image synchronization algorithm, images will be passed through at a rate of 10Hz (approximately a new set of images for every 100ms). A feature detection/matching algorithm running at a much slower rate is not feasible. Therefore, to balance out all the factors, the feasibility in real-time, and the accuracy in feature matching under intensity changes, it is recommended to use the feature detector/descriptor combination of *ORB/ORB*.

4.4.2 Feature tracking analysis

In navigation, estimating the current pose, in general, is done via different sensor readings, such as odometry measurement, IMU measurement etc. In visual pose estimation methods, the substitution for these typical sensor data is the features and their movements. Feature movement in a sequence of images describes, how the camera capturing those features moves in the space, similarly to how continuous odometry data describes a robot/vehicle's wheel movement.

As described in Section 3.3.2, the behavior of feature movement was analyzed with two different FoV values over a sequence of images. In the first case, feature movement was analyzed with full FoV of $\approx 80°$. In the second case, the image was divided into three masks the center, the left, and right regions, as illustrated in Figure 3.9. For this experiment, FoV of the center region was set to $\approx 40°$ by reducing 50% of the full FoV. The other two regions (left and right) was not removed like in the pose estimations with reduced FoVs experiment described in sections 3.4 and 4.3. Instead, features in those two areas were also tracked and analyzed for better qualitative analysis.

(a)



(b)

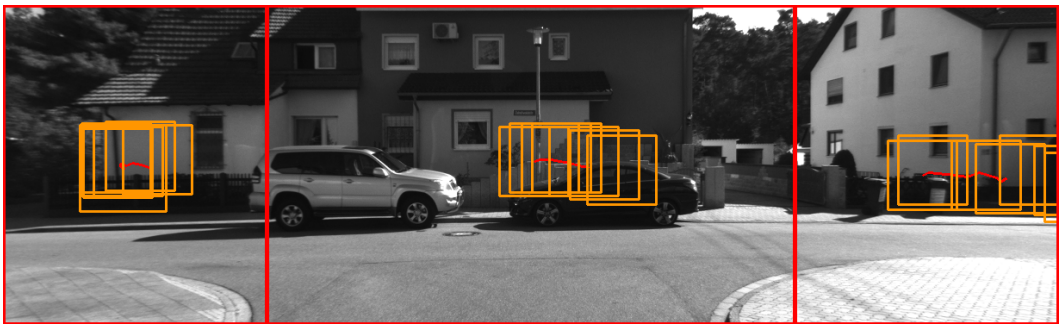
Figure 4.17: Feature movements in pixel level between three frames. In both figures, the points that are colored in blue represent the features detected in the initial frame. Let the initial frame be $i = 1$, then the features that were detected and matched between frames $i = 1$ and $i = 2$ are represented in red color. Features detected and matched between frames $i = 2$ and $i = 3$ are colored in green. Abbreviation KP stands for the number of keypoints detected in each region. TD means the total distance moved by all the features in a given region, and AD means the average distance moved in pixel level. In both figures, calculated distances are between frames $i = 2$ and $i = 3$.

The movement of the features in consecutive images decides the motion vector of the camera. The magnitude of this motion vector between two images decides how much the camera moves compared to its previous position. As shown in Figure 4.17, features that in the center region tend to move very little compared to the other two regions, despite having a comparatively higher number of keypoints. This affects the average feature movement between two frames as shown in the two figures 4.17a and 4.17b, which eventually decides the magnitude of the motion vector. In this Figure 4.17, the center region represents the scenario of having a lower FoV and its effect on the overall motion vector. Note that in this Figure 4.17, a left turn was simulated. Therefore a comparatively larger movement of features can be seen in the right region.

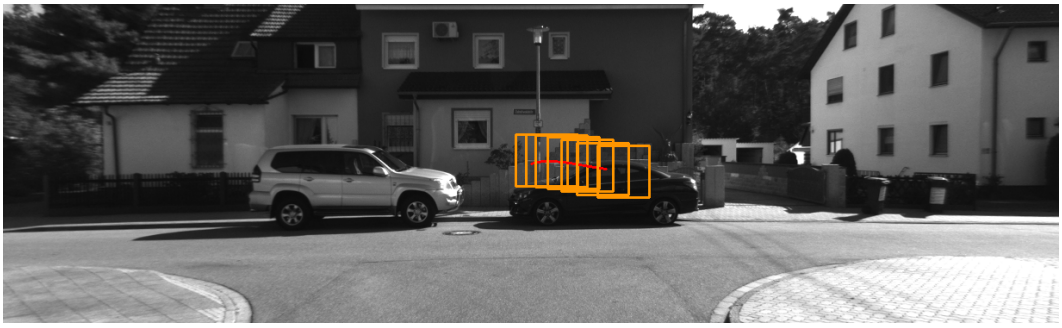
Using the method described in Section 3.3.2, a motion vector was calculated between two frames and was updated throughout a sequence of images. As illustrated in Figure 4.17a, three motion vectors were calculated, and these three motion vectors were compared with a motion vector calculated with an image with full FoV (an image similar to Figure 4.17b).

(a) Initial image of the sequence (frame $i = 1$)

(b) Final image of the sequence (frame $i = 7$)



(c) Motion vectors in three masks



(d) Motion vector in full FoV image

Figure 4.18: Motion vector tracking over a sequence of seven images. In figures (b) and (c), the red color path represents the motion vector. Yellow color rectangles represent the search windows. Since no specific object was defined in this tracking algorithm, in the initial case, the keypoint closest to each mask's center (or in the case of full FoV, the keypoint closest to the image center) was selected and tracked. The search windows are used to track this initially selected keypoint throughout the image sequence accurately.

As it can be seen in Figure 4.18, the motion vector estimated with full FoV have much larger magnitude compared to the motion vector estimated in the center region of the Figure 4.18c. The next logical question one could ask is why the average feature movements are low with lower FoV images. When considering the features in the center part of the image, the movement is low regardless of the FoV value, but the real effect of having a lower FoV is that it tends to lose matched features more frequently than in the case of a much larger FoV. It is true that regardless of the FoV, for each frame, a new set of features are introduced. However, these new features will not have any matching features in the previous frame. Hence those new features will not be categorized as matched features and will not affect the overall motion vector.

A set of images was selected to get a better overview of feature lost, and features of these images were detected and tracked. In this experiment, since it was essential to track only the center region's features, the two corners of the image were removed. First, all the features in the initial frame (frame $i = 1$) were detected and saved, and then in the matching stage, these saved features were matched with different frames from $i = 2$ to $i = 7$ to see the feature lost in a sequence of images. The results of this experiment are given in Figure 4.19.

In this experiment the frames $i = 1$ and $i = 2$ are the initial image pairs. Therefore, between these two frames, the matched percentage will be 100%. The comparison will start with frame $i = 3$, and as one could see in Figure 4.19a. Even after a single frame change, almost 10% of the initial features were lost. Analyzing the image in Figure 4.19e, just five frames after the initial pair, it already loose close 40% of its initial features.

(a)

(b)

(c)

(d)

(e)

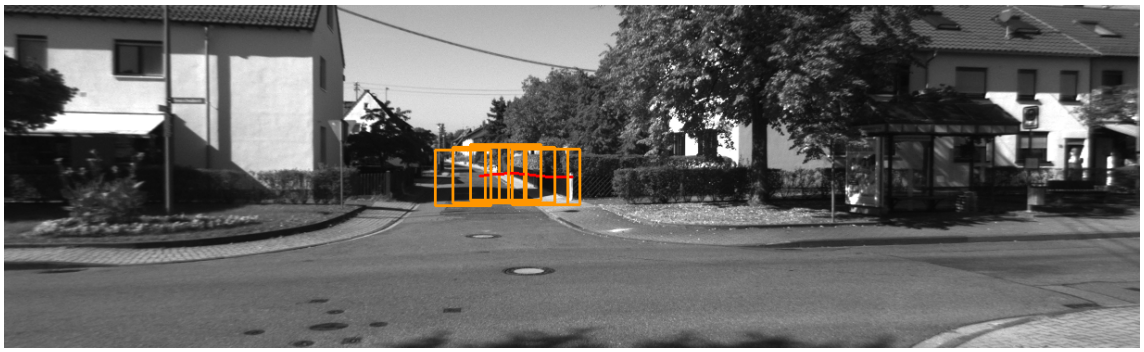
Figure 4.19: Feature lost percentages in a sequence of images. Blue color points marked in all the figures are the initial feature points detected at frame $i = 1$. Green color features are the ones that were detected and matched with the previous frame. Red color once are the features that were detected in the initial frame but were not detected and matched in the current frame.

This feature lost is for an image with an FoV of $\approx 40^\circ$ and one should understand that the percentage of losing features would only increase with FoV reduction. This feature lost not only effects the overrall magnitude of the motion vector but also it effects the smoothness of the motion vector. it can be also seen in Figure 4.18, that in full FoV case, the estimated vector have a much smoother than with less FoV version (consider the center region of Figure 4.18c).

In Section 4.3, when analyzing the pose estimations with different FoVs, it was observed that the errors were significantly high in eventful situations compared to when moving in straight lines. This behavior is illustrated in Figure 4.13a. This error behavior in pose estimation can also be explained via the motion vector estimations. In eventful

situations, features tend to move fast compared, resulting in motion vectors with higher magnitude. However, the magnitude of estimated motion vectors with low FoV will be lower than the motion vectors estimated with higher FoVs. Hence the resulting errors will also be higher in eventful situations compared to straight-line motion. A comparison of a straight line motion and a turn motion (eventful situation) is illustrated in Figure 4.20.

(a)



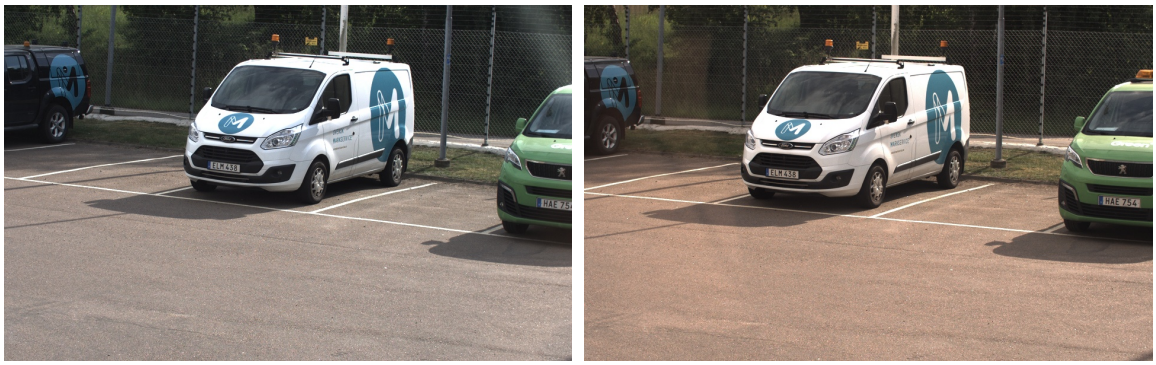
(b)

Figure 4.20: Motion vectors in straight-line motion and turn motion. Figure (a) illustrates a motion vector estimation in a straight line throughout seven frames. The motion vector estimation with the same number of frames but with a turning motion is shown in Figure (b). These two scenarios represent approximately the same areas illustrates in figures 4.13b and 4.13c.

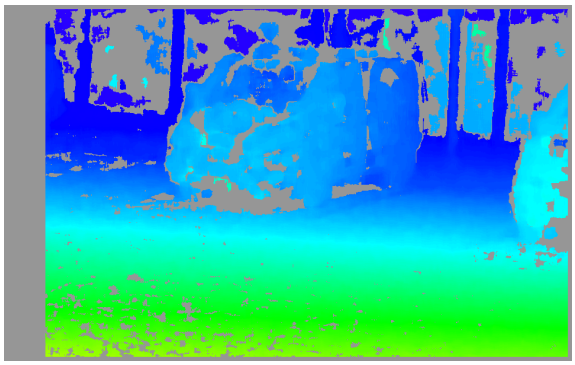
4.5 Free-space detection

As mentioned in Section 3.2.2, free-space detection aims to detect objects that are closer to the vehicle. Once objects in the space are detected, one can identify whether that detected object is close to the vehicle (the camera) or not based on a pre-defined threshold. Before defining the said threshold, the initial task would be to identify all different objects in the environment. To correctly differentiate objects from the background, a multi salient object detection method is used as described in Algorithm 5. This algorithm initially needs the depth map of a scene. This is achieved by creating the disparity map of a scene given two synchronized left and right images. Figure 4.21 illustrates the first steps of the process.

The gray pixels in the middle of the image illustrates the confusion of stereo matching. One can see that areas with gray color pixels tend to have similar environments throughout



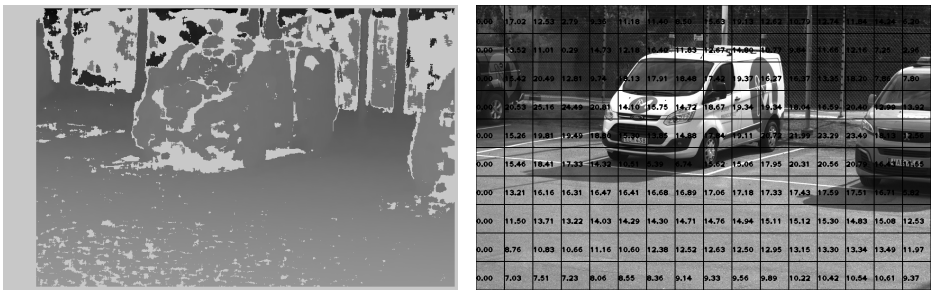
(a) Left image **(b)** Right image



(c) Disparity map

Figure 4.21: Left, right images and the resulting disparity map. In Figure 4.21c, the gray color represents the disparity values that were not calculated. The far left columns tend to have gray values because that section in the left image is not visible from the right image.

a larger area in the original left and right images (4.22a, 4.22b), which tends to confuse the stereo correspondence algorithm even if the searches are carried out along a single epipolar line. Once the disparity map is obtained, the next part would be to estimate the scene's depth (as described in Section 2.1). The resulting depth image is illustrated in Figure 4.22.

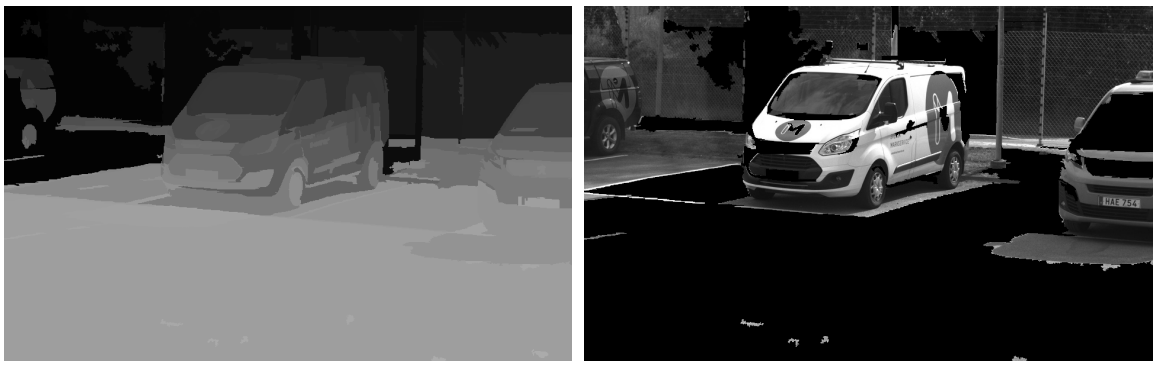


(a) Depth estimation **(b)** Depth grid

Figure 4.22: Estimated depth map of the scene. Figure (a) is the resulting depth, and Figure (b) is a visualization of the depth values in meters. These depth values are illustrated as a grid, where each value is the average of the given smaller marked regions.

Once the depth is estimated, the next task to obtain the segmented image. This is done using graph-based segmentation as described in Section 3.2.2. The process of graph-based segmentation is already implemented in OpenCV. However, this process tends to take a considerable amount of time (around 90 ms) to process an image with a complex environmental feature, which one would find in an urban environment. Since in this thesis, the main concern is about detecting objects closer to the vehicle, it is justifiable to ignore objects that are far away in the scene, using the estimated depth values. Setting the intensity value to zero of all the pixels that are larger than a pre-defined threshold value can make the segmentation process a little bit computationally feasible with real-time*. By ignoring these far-away pixels, one would not only obtain faster results from the segmentation process, but this would also create fewer segments. Setting all intensity values to a specific value that corresponds to the far-away depth pixels in the depth map forces all the selected far-away pixels to be a single region that would otherwise be categorized as lots of smaller segments. Hence more processing time to analyze each of these segments in the later part of the algorithm.

The resulting segmentation image is shown in Figure 4.23a. The left image (in Figure 4.22a), the depth map and the segmented image will be given as inputs to the Algorithm 5 to estimate the backgroundness of the scene as described in Section 3.2.2. Figure 4.23b illustrates the resulting image after categorizing all the segments according to (3.14).



(a) Graph based segmentation **(b)** Salient object

Figure 4.23: Graph-based segmentation and the final results of multi salient object detection. In Figure (b), the pixels with intensities zero are categorized as background, and the non-zero pixels are identified as foreground pixels.

This free-space detection's objective is to build a warning system. Therefore accurately identify each object was not prioritized; instead, a more computational reliable method was expected. Figure 4.23b, it can be seen that the area that is quite far away from the camera has inaccurate background/foreground detection. However, the detection is quite reliable in the close regions. As mentioned, pixel locations with relatively larger values in the corresponding depth map are ignored, deliberately to reduce the computational cost results due to the segmentation process, and the downside is, of course, some unreliable background/foreground detection.

*Compared to the 90 ms value, after the modifications to the far away pixels, the time tends to decrease close to 60 ms.

Once the salient objects are identified in the scene, the next task is to check whether any of these detected objects are in close range with the vehicle. This was done by allocating a *danger zone* in the space in front of the vehicle. This danger zone represents the vehicle's location in future iterations.

The ground plane was estimated as the first step of defining the danger zone. Here a rectangular area, which is very close to the camera is selected*. In this thesis, a method based on homography projection is used to estimate the ground plane.

The first step of this process is defining a rectangular area using four points in the ground plane close to the vehicle. These four points can be marked by placing some specific objects (like traffic cones). Then an image of the scene covering all the four objects (markers) needs to be captured using a fixed stereo rig. This captured image represents the ground plane in the image plane and let that four images points be represented by a vector $x = \begin{pmatrix} x_i & y_i & 1 \end{pmatrix}^T$, where $i = 1..4$. Since the actual distance between the four markers are known, one could define a different coordinate frame, where these four markers are defined with respect to a selected marker and that selected marker has the coordinate value $(0, 0)$. For example, assume that the selected rectangle has a height of 2m and a width of 1m. Then if the left bottom corner point is $(0, 0)$, then the top-left point would be $(0, 2)$. Right bottom and top will then be defined as $(1, 0)$ and $(1, 2)$. These newly defined four coordinates crates a plane that is also bounded within the selected four marker points. If these four points are defined by a vector $X = \begin{pmatrix} X_i & Y_i & 1 \end{pmatrix}^T$ (where $i = 1..4$), then there exists, a relationship between the two planes (the image plane and the user-defined ground plane),

$$X = Hx \tag{4.2}$$

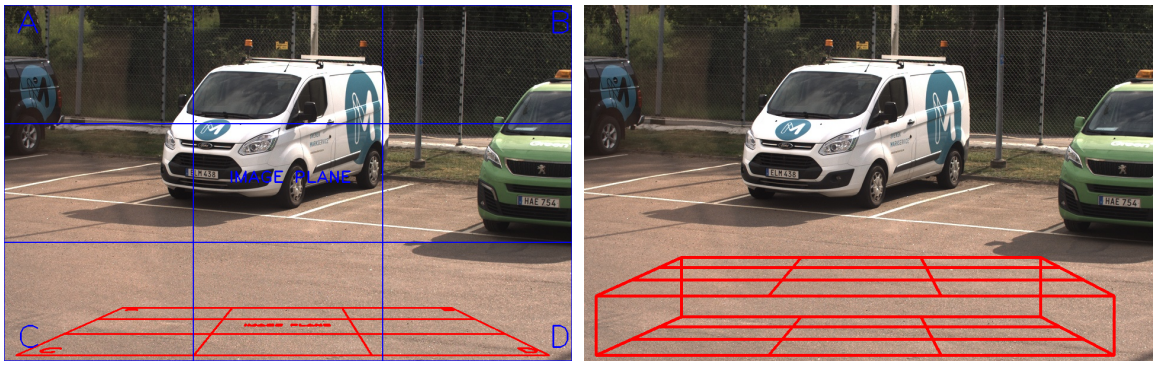
where H is a homography matrix† that describes the projection between the image plane and the newly defined coordinate frame (the ground plane bounded by the four marker points).

Since the vector x in (4.2) represents image coordinates in the current image plane, the matrix H will allow any point in the image plane to be transformed into the newly defined ground plane. Figure 4.24a present a grid covering the entire image being transformed into the estimated ground plane. Since this newly transformed plane represents the user-defined area in the initial step of this process, this new estimated ground plane can be named the *danger zone* for the vehicle. These are often called region of interest (ROI) areas and are bounded by predefined points. Furthermore, this area can be defined as closed polygons, and operations like finding an area or counting the number of non-zero pixels are possible.

This method of estimating the ground plane is quite popular and effective since almost all the ground planes' points can usually be fit into a plane (as ground planes are often defined as flat surfaces). However, due to practical constraints, placing markers to obtain actual distances was not possible in this thesis. Therefore, initial marker values were

*The image coordinate frame in the OpenCV library defines its initial coordinate $0, 0$ in the top left corner. Therefore when defining a space close to the camera in the image plane, pixel locations with large y coordinate need to be selected.

†To estimate this matrix, technique described in either Section A.4 or in Appendix A.3 can be used. In the implementation, this matrix was obtained using OpenCV function *findHomography*



(a) **(b)**

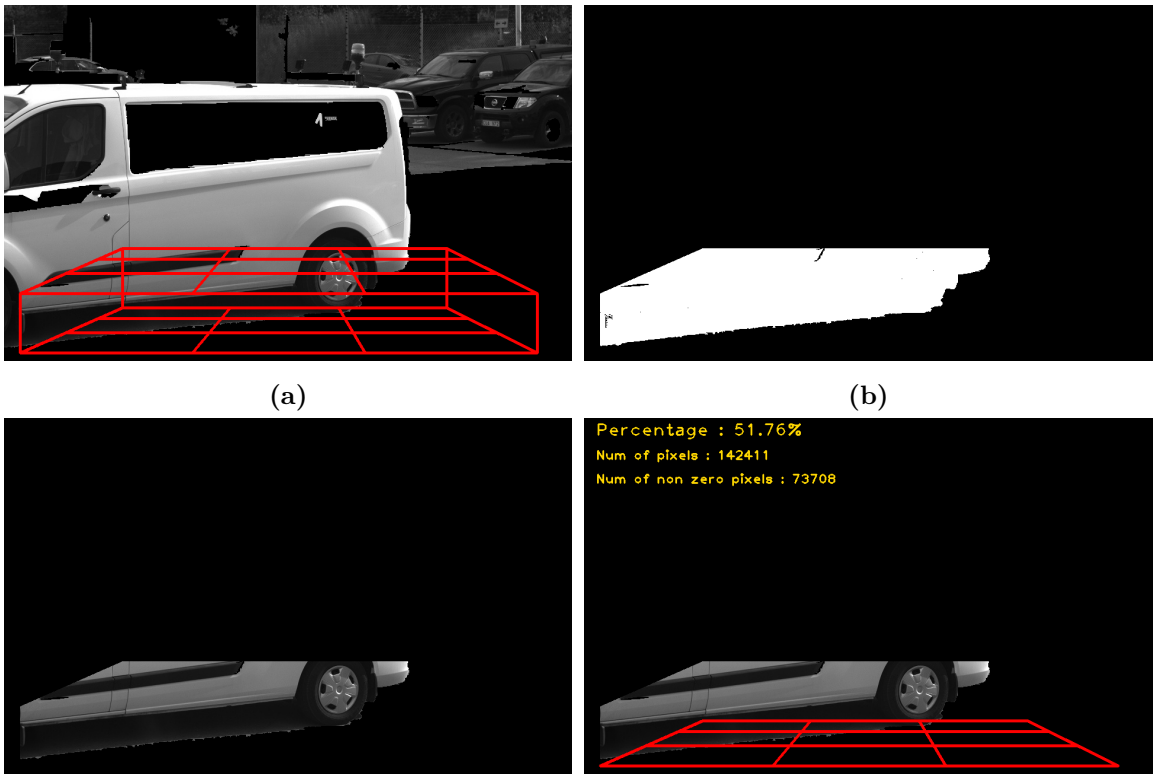
Figure 4.24: Ground plane estimation and 3D bounding box estimation. Figure (a) illustrates the estimated ground plane. This ground plane is a projection from the blue color grid, and one can see how the letters are projected ground plane after the projection (in red color). Here the four corners marked by letters *A,B,C* and *D* are the initial marker locations. Figure (b) is a 3D bounding box, estimated using the projected ground plane. The height of this box can be adjusted according to the user's need.

estimated using the depth grid illustrated in Figure 4.4. This resulted in minor inaccuracies in the estimated 3D bounding box.

As described in the salient object detection phase, all the pixels that are estimated as background pixels are assigned with zero pixel values making all those regions that are categorized as background to have black color. The rest of the pixels will have their original intensity values, making only the foreground objects (pixels) to be visible. With ROI defined, such foreground pixels been entered into the danger zone can be easily detected. This technique is used in detecting objects closer to the region, but since the objective is to perform free-space object detection, this ground plane was further extended to resemble a rectangular box as illustrated in Figure 4.24b.

The first thing that needs to be done in this process is to define the close polygon that represents the defined 3D bounding box. This closed polygon can be defined using a minimum of six points*. Once the polygon is defined, it can be recognized as a sub-mask in the image. Next, the number of pixels in that polygon (newly created sub-mask) is calculated. Since in this thesis, the estimated 3D bounding box is static, the total number of pixels in this polygon will remain the same throughout the operation. Taking the intersection between the created mask and the final results from the SO detection process (Figure 4.23b), one would obtain all the foreground objects in the selected 3D bounding box as illustrated in Figure 4.25c. Since the total number of pixels in the selected 3D bounding box is known, one could calculate a percentage between the zero and non-zero pixels ratio as only the foreground pixels have non-zero values. Using two per-define threshold values according to the user's need, one can categorize any percentage value into three categories. The entire process starting from the results from SO detection until calculating the percentage value for the selected mask is illustrated in figure 4.25.

*Four corner points in the top layer of the 3D box and two points with the largest y value in the bottom layer



(a) **(b)**

(c) **(d)**

Figure 4.25: Object detection process in the danger zone. Figure (a) represents the 3D bounding box and the output image from the SO detection phase. Next, in Figure (b), the binary mask, which contains all the non-zero values, is presented. The intersection of the mask and Figure (a) is given in Figure (c). Figure (d) illustrates a visualization of the ground plane and the calculated percentages.

The three scenarios of *Ideal, Warning* and *Danger* after the final processing in *free-space* detection is illustrated in Figure 4.26.

The objective of free-space detection addressed in this thesis was to warn about the objects in front of the vehicle. Since this was only meant to be a warning system, high accuracy in detecting all the objects in the surrounding environment or object classification was not expected. Processing an entire image with a resolution of 960×600 is quite computationally heavy. Therefore a few extra measurements like ignoring the pixels that correspond to high depth values were performed. The trade-off was that the objects in those ignored areas had inaccurate interpretations of objects being foreground or background. This behavior can be seen in Figure 4.23b. However, the pixels that are closer to the camera (specifically the pixels with high y coordinate values) have reasonably good estimations. For example, the figure recognizes the entire ground plane as a background region compared to the two vehicles in the image's middle section. This SO detection method can therefore be recommended to solve small scale free-space detection problems.

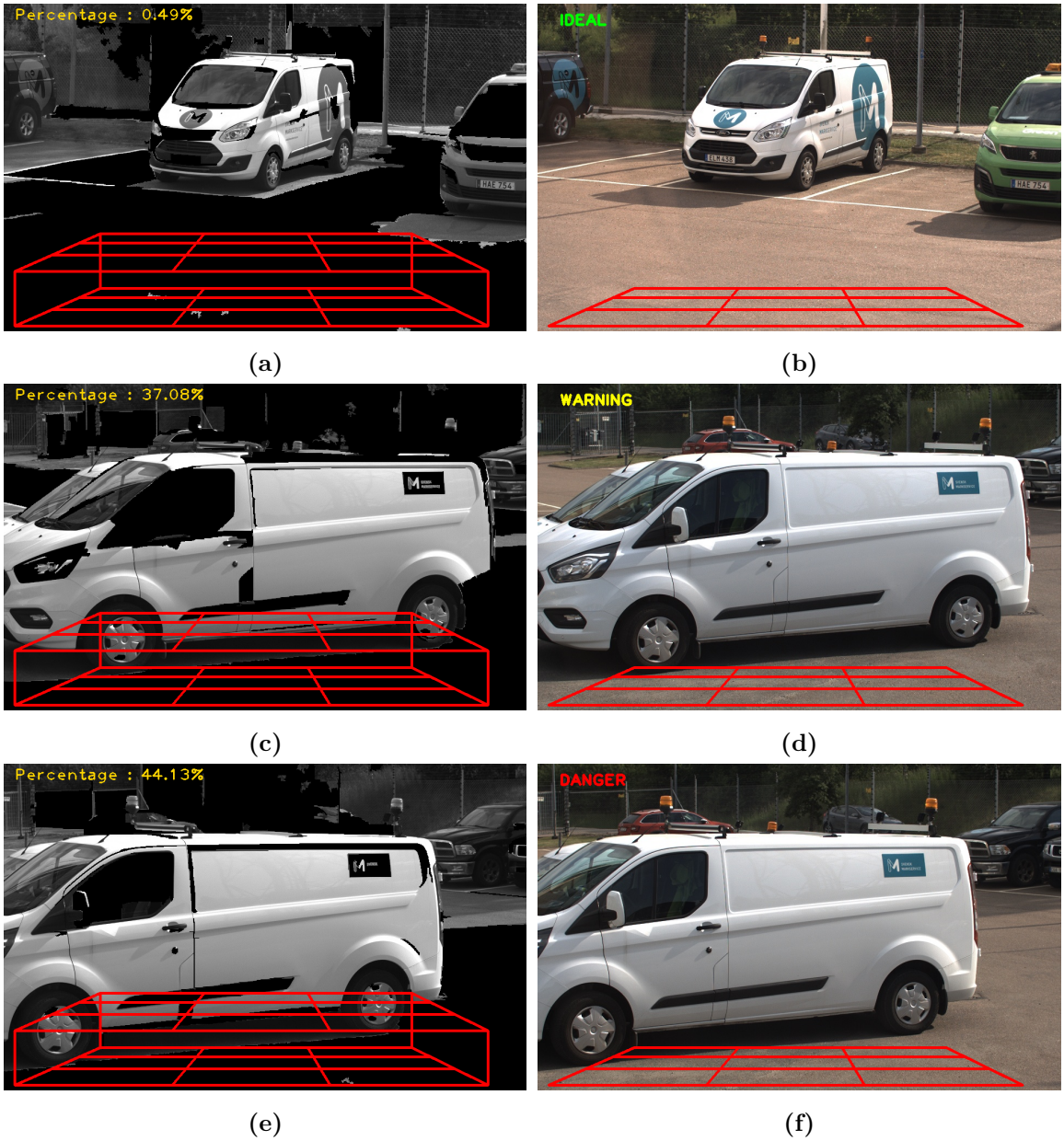


Figure 4.26: The three different warning types after the detection process. The percentage is shown on the top left corner in all these figures (a),(c),(e) is the zero to non-zero pixel ratio. The first pair, figures (a),(b), shows an ideal case, where no object is in the danger area. Figures (c) and (d) represent a warning scenario. The final pair, figures (e) and (f), illustrates the "danger" scenario. In this thesis, percentages above 40% were considered a "danger" scenario, and a "warning" scenario is defined between 10% and 40%. "Ideal" case is where the percentages are below 10%.

5

Conclusion & Future Work

In this chapter, a conclusion to the problem, the feasibility of using a computer vision-based pose estimation method for urban navigation is presented. This thesis mainly focused on evaluating a visual SLAM-based approach to solve a common yet challenging problem in autonomous navigation; the pose estimation problem. Apart from the primary goal, a free space detection method is also implemented and evaluated. After a thorough analysis of the current systems' results, few suggestions are presented for future developers who might be interested in addressing autonomous navigation problems for outdoor urban vehicles.

5.1 Conclusions

This thesis starts by discussing one of the most talked-about topics in autonomous navigation areas, the pose estimation problem and the currently used methods in obtaining accurate pose estimation results. As described in Chapter 1, the main issue with the widely used traditional estimation methods is their questionable accuracy in estimations, particularly in urban environments. Therefore a computer vision-based method is suggested in this thesis due to its lightweight hardware configurations and minimal interference on the sensor data (images) caused by the environmental factors.

In this thesis, to evaluate a computer vision-based pose estimation method's feasibility, a state-of-the-art visual SLAM method was implemented and tested using a dataset collected by a utility vehicle. This *BARMARK* dataset used to evaluate the performance of the vision-based pose estimation system contains high-quality images with a resolution of 960×600. However, the FoV of the camera used to capture these images was around $23°$. The estimation generated from *BARMARK* dataset did not have the expected level of accuracy, and an initial guess was made that the mentioned low FoV value affected the results. Hence a hypothesis was generated that *FoV of a camera affect the performance of visual SLAM algorithms.*

It was observed that the FoV of a camera does play a major role in pose estimations. An experiment was carried out using a well-known benchmark dataset, the *KITTI* dataset to test the hypothesis mentioned above. The main difference between the two datasets was the FoV value. *KITTI* dataset had a FoV close to $82°$, which is approximately around 72% FoV increase compared to *BARMARK* dataset. As mentioned in Chapter 3, *KITTI* dataset was re-arranged to simulate different FoV values. These new images were then used to estimate the pose. From this experiment, it was proven that when the FoV values decrease, the estimation accuracy also decreases exponentially.

When FoV reduced to a minimum of $21°$ in the *KITTI* dataset, the mean ATE

reached to 6.5m, which was quite close to the ATE value obtained in the *BARMARK* dataset of 6.8m. One critical point about the *BARMARK* dataset was that it had two quick turns in quick successions at the beginning of the trajectory. This is why the mean ATE in *BARMARK* is slightly larger than *KITTI*, even though the area is relatively small compared to *KITTI*. As described in Chapter 4 pose estimations error are cumulative, and since taking a turn does have more effects on the overall estimation as described in Section 4.4.2, the mean ATE of *BARMARK* was recorded as slightly higher than the *KITTI* dataset with approximately similar FOV.

The reason behind the FoVs effects on pose estimation in visual SLAM was also analyzed. As described in Chapter 2, feature detection and tracking are critical aspects of visual SLAM. It was observed that the feature tracking process was affected heavily by the FoV of the camera. An ORB tracking algorithm was used to capture how features move around when the camera moves, and using these movements, a motion vector was calculated. This motion vector calculated in the SLAM algorithm's front-end decides the final pose estimation accuracy from the SLAM algorithm's back-end. The movements of key-features decide the magnitude and the direction of this motion vector in a scene, and the movements of the features are often affected by narrow FoV values. It was observed that the inaccurate and unsteady motion vector estimations were due to the constant feature lost between consecutive frames, which are caused by narrow FoV. This behavior of losing constant features leads to lousy pose estimations. Apart from the narrow FoVs, it was observed that feature loss could also be caused by sudden intensity changes in the scene between frames, which fails the feature matching and tracking process. After a few experiments, ORB/ORB proven to be the most robust and computationally effective detector/descriptor combination to handle these sudden intensity changes.

Even with the full FoV from the *KITTI* dataset, it was observed that the minimum mean ATE achieved around 0.3m. This was also not an accepted mean error value compared to the other traditional methods mentioned above. In a nutshell, it is not recommended to use stand-alone vision systems for pose estimations. This pose estimation using vision approaches is still an open reach field, and hence achieving millimeter level accuracy in outdoor urban navigation can be quite challenging.

5.2 Future work

Analyzing the current results in pose estimations using the *BARMARK* dataset, it was observed that the FoV of the camera is the reason why the proposed system failed to achieve an acceptable result. However, as mentioned in the previous section, even with a reasonable FoV value, the system still fails to achieve millimeter length accuracy. Therefore an extra sensor(s) for better accuracy is encouraged. After a simulated experiment using the *KITTI* dataset, an optimal value of at least 50° can be recommended, where the mean ATE reached close of 1.2m for an area of $250m^2$ (recall figures 4.14a and 4.15b). This estimation should then be fused with an external sensor for better accuracy.

When using sensor fusion techniques with external sensor data, it is important to know accurate transformations between the said two sensor frames. This is a challenging task, especially if these coordinate frames have relative motions with respect to each other. For example, an articulated vehicle (as addressed in this thesis) can have two sensors

that need to be fused but are located in different vehicle sections, making them move in separate trajectories at any random given time, this makes the fusion inaccurate if the transformation matrix has inaccurate values. Implementing a full-scale *URDF* node is the best solution for this issue since all the submodules are already connected under the ROS environment.

In regards to the system memory allocation, when running the vision application, the RAM of the system tends to reach up to 4GB. This was considered reasonable in this thesis since no other operation was running in parallel while executing *OpenVSLAM* library and its related submodules. However, this will not be the case in a practical implementation. For a utility vehicle like the one addressed in this thesis, multiple other operations need to be handled along with vision application. Therefore acquiring such memory value is not acceptable. It was noticed that the maximum amount of memory is required by the *Pangoine viewer*, which was the visualization tool in *OpenVSLAM* library. It is true that in the implementation phase, a visualization aid can be beneficial, however, this could be avoided in the actual execution phase. Another suggestion is to replace the current visualization tool with a lightweight tool in *ROS*.

The free-space detection was implemented as an external tool that acts as a warning system. This, of course, is not a full-scale object detection system, but this library is written so that the main algorithm can be modified using external plug-ins. The current algorithm is an adaption from the method suggested in [15], but this original paper further includes a fusion between another two different methods for better accuracy. Object detection can further be improved using deep learning techniques. Furthermore, the technique provided in this thesis to identify objects that are closer to the vehicle using ground plane estimation is a simple yet effective method. However, the ground plane estimation suggested in the thesis might not be the most robust method since its a static estimation. Hence more effective ground plane estimation using methods like *Iterative close point* (ICP) [48] or estimations based on the geometric consistency of the scene as presented by Yunze Man et.el. [49] is suggested.

Bibliography

[1] Saeed Asadi Bagloee, Madjid Tavana, Mohsen Asadi, and Tracey Oliver. Autonomous vehicles: challenges, opportunities, and future implications for transportation policies. *Journal of Modern Transportation*, 24(4):284–303, 2016. ISSN 21960577. doi: 10.1007/s40534-016-0117-3.

[2] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *IEEE Access*, 8:58443–58469, 2020. ISSN 21693536. doi: 10.1109/ACCESS.2020.2983149.

[3] Elliot Fishman. Cycling as transport. *Transport Reviews*, 36(1):1–8, 2016. ISSN 14645327. doi: 10.1080/01441647.2015.1114271.

[4] European Transport Safety Council (ETSC). RANKING EU PROGRESS on road safty 14th Road Safety Performance Index Report. (June):40, 2020.

[5] Chuck Thorpe and Takeo Kanade. Vision and Navigation for the CMU Navlab. *Mobile Robots I*, 0727(3):261, 1987. ISSN 1996756X. doi: 10.1117/12.937805.

[6] Mark Campbell, Magnus Egerstedt, Jonathan P. How, and Richard M. Murray. Autonomous driving in urban environments: Approaches, lessons and challenges. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368(1928):4649–4672, 2010. ISSN 1364503X. doi: 10.1098/rsta.2010.0110.

[7] Mojdeh Azad, Nima Hoseinzadeh, Candace Brakewood, Christopher R. Cherry, and Lee D. Han. Fully Autonomous Buses: A Literature Review and Future Research Directions. *Journal of Advanced Transportation*, 2019(Figure 1), 2019. ISSN 20423195. doi: 10.1155/2019/4603548.

[8] Jin Zhao Yuan, Hui Chen, Bin Zhao, and Yanyan Xu. Estimation of Vehicle Pose and Position with Monocular Camera at Urban Road Intersections. *Journal of Computer Science and Technology*, 32(6):1150–1161, 2017. ISSN 18604749. doi: 10.1007/s11390-017-1790-3.

[9] David Wong, Daisuke Deguchi, Ichiro Ide, and Hiroshi Murase. Vision-based vehicle localization using a visual street map with embedded SURF scale. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8925:167–179, 2015. ISSN 16113349. doi: 10.1007/978-3-319-16178-5{_}11.

[10] Motilal Agrawal and Kurt Konolige. Real-time localization in outdoor environments

using stereo vision and inexpensive GPS. *Proceedings - International Conference on Pattern Recognition*, 3:1063–1068, 2006. ISSN 10514651. doi: 10.1109/ICPR.2006.962.

[11] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based SLAM. *IEEE Intelligent Transportation Systems Magazine*, 2 (4):31–43, 2010. ISSN 1939-1390. doi: 10.1109/MITS.2010.939925.

[12] M. W.M. Gamini Dissanayake, Paul Newman, Steven Clark, Hugh F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001. ISSN 1042296X. doi: 10.1109/70.938381.

[13] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g2o: A General Framework for Graph Optimization. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3607–3613, 2011. ISBN 9781612843865. doi: 10.1109/ICRA.2011.5979949.

[14] IEEE Hugh Durrant-Whyte, Fellow and Tim Bailey. Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms. *American Journal of Veterinary Research*, 45(5):963–966, 1984. ISSN 00029645.

[15] Christian Hofmann, Florian Particke, Markus Hiller, and Jörn Thielecke. Object detection, classification and localization by infrastructural stereo cameras. *VISIGRAPP 2019 - Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 5(Visigrapp):808–815, 2019. doi: 10.5220/0007370408080815.

[16] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The KITTI dataset. The International Journal of Robotics Research. *The International Journal of Robotics Research*, (October), 2013.

[17] Shinya Sumikura, Mikiya Shibuya, and Ken Sakurada. OpenVSLAM: A versatile visual SLAM framework. *MM 2019 - Proceedings of the 27th ACM International Conference on Multimedia*, pages 2292–2295, 2019. doi: 10.1145/3343031.3350539.

[18] Alex Zelinsky. *Learning OpenCV—Computer Vision with the OpenCV Library (Bradski, G.R. et al.; 2008)*, volume 16. 2009. ISBN 9780596516130. doi: 10.1109/mra.2009.933612.

[19] Luis Fernandez, Viviana Avila, and Luiz Gonçalves. A Generic Approach for Error Estimation of Depth Data from (Stereo and RGB-D) 3D Sensors. *Preprints*, (May): 1–12, 2017. doi: 10.20944/preprints201705.0170.v1.

[20] Zhengyou Zhang. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:1330–1334, 12 2000. URL `https://www.microsoft.com/en-us/research/publication/a-flexible-new-technique-for-camera-calibration/`.

[21] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[22] Wilhelm Burger. Zhang's Camera Calibration Algorithm: In-Depth Tutorial and Implementation. *Technical Report*, page 55, 2016. doi: 10.13140/RG.2.1.1166.1688. URL `http://staff.fh-hagenberg.at/burger/.`

[23] Richard Szeliski. Computer vision - Algorithms and Applications. *Computer Science Handbook, Second Edition*, pages 43–1, 2010. ISSN 1340-5551. doi: 10.4324/9780429042522-10.

[24] Wei Li, Trevor Gee, Heide Friedrich, and Patrice Delmas. A practical comparison between Zhang's and Tsai's calibration approaches. *ACM International Conference Proceeding Series*, 19-21-Nove:166–171, 2014. doi: 10.1145/2683405.2683443.

[25] R.I. Hartley. Theory and Practice of Projective Rectification. *International Journal of Computer Vision*, 35:115–127, 1999. doi: https://doi.org/10.1023/A:1008115206617.

[26] Marsha Jo Hannan. *Computer Matching of Ares in Stereo Images*. PhD thesis, Stanford University, 1974. URL `http://www.dtic.mil/dtic/tr/fulltext/u2/p004398.pdf`.

[27] Takeo Kanade, Hiroshi Kano, Shigeru Kimura, Atsushi Yoshida, and Kazuo Oda. Development of a video-rate stereo machine. *IEEE International Conference on Intelligent Robots and Systems*, 3:95–100, 1995. ISSN 0289-1824. doi: 10.7210/jrsj.15.261.

[28] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. ISSN 09205691. doi: 10.1023/B:VISI.0000029664.99615.94.

[29] Tinne Tuytelaars Herbert Bay and Luc Van Gool. SURF: Speeded Up Robust Features. *European Conference on Computer Vision (ECCV)*, 3951:404–417, 2006. ISSN 978-3-540-33833-8. doi: https://doi.org/10.1007/11744023{_}32.

[30] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF: Binary robust independent elementary features. *European Conference on Computer Vision - ECCV 2010*, pages 778–792, 2010. ISSN 16113349. doi: 10.1007/978-3-642-15561-1{_}56.

[31] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. *Proceedings of the IEEE International Conference on Computer Vision*, pages 2564–2571, 2011. doi: 10.1109/ICCV.2011.6126544.

[32] Margarita Chli Stefan Leutenegger and Roland Y. Siegwart. BRISK: Binary Robust Invariant Scalable Keypoints Stefan. *IEICE Transactions on Information and Systems*, E96-D(2):392–395, 2013. ISSN 17451361. doi: 10.1587/transinf.E96.D.392.

[33] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. FREAK: Fast retina keypoint. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 510–517, 2012. ISSN 10636919. doi: 10.1109/CVPR.2012.6247715.

[34] Paul L. Rosin. Measuring Corner Properties. *Computer Vision and Image Understanding*, 73(2):291–307, 1999. ISSN 10773142. doi: 10.1006/cviu.1998.0719.

[35] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016. ISSN 15523098. doi: 10.1109/TRO.2016.2624754.

[36] Richard Hartley Bill Triggs, Philip McLauchlan and Andrew Fitzgibbon. Bundle Adjustment - A Modern Synthesis. *International Workshop on Vision Algorithms IWVA 1999: Vision Algorithms: Theory and Practice*, pages 298–372, 1999. ISSN 01608371. doi: https://doi.org/10.1007/3-540-44480-7{_}21.

[37] Stanford Artificial Intelligence Laboratory et al. Robotic Operating System - ROS Kinetic Kame, 2016. URL `https://www.ros.org`.

[38] Morgan Quigley and Brian Gerkey and Ken Conley and Josh Faust and Tully Foote and Jeremy Leibs and Eric Berger and Rob Wheeler and Andrew Ng. ROS: an open-source Robot Operating System. In *Robotics Operating System*. URL `http://wiki.ros.org`.

[39] Steven Lovegrove and Richard Newcombe. Pangolin, 2011. URL `https://github.com/stevenlovegrove/Pangolin`.

[40] David Prokhorov, Dmitry Zhukov, Olga Barinova, Anna Vorontsova, and Anton Konushin. Measuring robustness of visual SLAM. *Samsung AI Center*, 2019.

[41] Qibin Hou, Ming-ming Cheng, Xiaowei Hu, Ali Borji, Zhuowen Tu, and Philip H S Torr. Deeply Supervised Salient Object Detection with Short Connections. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

[42] Ming-ming Cheng, Niloy J Mitra, Xiaolei Huang, Philip H S Torr, and Shi-min Hu. Global Contrast based Salient Region Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015.

[43] Yong Zhang, Yihua Lan, Haozheng Ren, and Ming Li. Robust frequency-tuned salient region detection. *International Journal of Digital Content Technology and its Applications*, 6(20):361–369, 2012. ISSN 19759339. doi: 10.4156/jdcta.vol6.issue20.39.

[44] S. You D. Feng, N. Barnes and C. McCarthy. Local Background Enclosure for RGB-D Salient Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2343–2350, 2016. doi: 10.1109/CVPR.2016.257.

[45] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient Graph-Based Image Segmentation. 2004. ISSN 1573-1405. URL `papers3://publication/uuid/D1250C05-2FC7-4954-A734-E33EBBEECB95`.

[46] Shuang Wu, Yawen Fan, Shibao Zheng, and Hua Yang. Object tracking based on ORB and temporal-spacial constraint. *2012 IEEE 5th International Conference on Advanced Computational Intelligence, ICACI 2012*, pages 597–600, 2012. doi: 10.1109/ICACI.2012.6463235.

[47] Martin A. Fischler and Robert C. Bolles. Random Sample Consensus: A Paradigm

for Model Fitting with Applications to Image Analysis and Automated Cartography. Technical report, 1981.

[48] Nikolay Chumerin and Marc M Van Hulle. Ground Plane Estimation Based on Dense Stereo Disparity. *The Fifth International Conference on Neural Networks and Artificial Intelligence (ICNNAI)*, (April 1930):209–213, 2008.

[49] Yunze Man, Xinshuo Weng, Xi Li, and Kris Kitani. GroundNet: Monocular ground plane normal estimation with geometric consistency. *MM 2019 - Proceedings of the 27th ACM International Conference on Multimedia*, (August):2170–2178, 2019. doi: 10.1145/3343031.3351068.

[50] Advisory Board. *Numerical Methods for Partial Differential Equations*, volume 79. 2004. ISBN 9781846280405. doi: 10.1086/423055. URL http://books.google.com/books?hl=en&lr=&id=OyD4toi-XDIC&oi=fnd&pg=PA2&dq=Game+theory+:+Decisions,+Interactions+and+Evolution&ots=TfgJJReigk&sig=AnVOJ8VTGk60lUCldvLrPPnrJD4.

A

Indepth Theoretical Explanations

A.1 Camera Calibration - DLT

Given the P matrix as,

$$P = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{pmatrix} \tag{A.1}$$

One could notice that each point gives 2 observation equations, one for each image coordinate,

$$x' = \frac{p_{11}X + p_{12}Y + p_{13}Z + p_{14}}{p_{31}X + p_{32}Y + p_{33}Z + p_{34}} \tag{A.2a}$$

$$y' = \frac{p_{21}X + p_{22}Y + p_{23}Z + p_{24}}{p_{31}X + p_{32}Y + p_{33}Z + p_{34}} \tag{A.2b}$$

Where, x', y' represents the image coordinates and X, Y, Z represents the world coordinates. Therefore to estimate the camera matrix which has 11 DoF, one might need at least 6 points. For the simplification purpose, if the matrix P is denoted as $P = \begin{bmatrix} A^T & B^T & C^T \end{bmatrix}$, then the above A.2a and A.2b can be further written as,

$$x' = \frac{A^T}{C^T} \cdot X \Rightarrow x'C^TX - A^TX = 0 \tag{A.3a}$$

$$y' = \frac{B^T}{C^T} \cdot X \Rightarrow y'B^TX - A^TX = 0 \tag{A.3b}$$

This could be written in the form of a matrix A where x_i and X_i are the i^{th} image point and its corresponding world coordinate.

$$\underbrace{\begin{bmatrix} -X_1 & -Y_1 & -Z_1 & -1 & 0 & 0 & 0 & 0 & x_1X_1 & x_1Y_1 & x_1Z_1 & x_1 \\ 0 & 0 & 0 & 0 & -X_1 & -Y_1 & -Z_1 & -1 & y_1X_1 & y_1Y_1 & y_1Z_1 & y_1 \\ ... & ... & ... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\ -X_i & -Y_i & -Z_i & -1 & 0 & 0 & 0 & 0 & x_iX_i & x_iY_i & x_iZ_i & x_i \\ 0 & 0 & 0 & 0 & -X_i & -Y_i & -Z_i & -1 & y_iX_i & y_iY_i & y_iZ_i & y_i \\ ... & ... & ... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\ -X_n & -Y_n & -Z_n & -1 & 0 & 0 & 0 & 0 & x_nX_n & x_nY_n & x_nZ_n & x_n \\ 0 & 0 & 0 & 0 & -X_n & -Y_n & -Z_n & -1 & y_nX_n & y_nY_n & y_nZ_n & y_n \end{bmatrix}}_{:=A} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{A.4}$$

Solving the equation $Ax = 0$, and finding the null space of A using *Singular Value Decomposition (SVD)*, will give an estimated P matrix. Since the intrinsic matrix K (as expressed in 2.1), is a positive definite upper-triangular matrix, one could use *RQ Factorization* to decompose the K matrix.

A.2 Zhang's Method

Since the third column of the rotational matrix will be eliminated once its multiplied with the Z value, the same equation can be re-written as,

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{pmatrix}}_{K} \underbrace{\begin{pmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{pmatrix}}_{r_1,r_2,t} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \tag{A.5}$$

For the purpose of simplification the product of K and the r_1, r_2, t matrices will be defined as a new matrix which is a 3×3 as $H = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix}$. Then for an image with multiple image points (the corner points which are detected using an edge detection algorithm) the following equation can be derived (where L is the number of feature points in a single image),

$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \underbrace{H}_{3\times3} \begin{pmatrix} X_i \\ Y_i \\ 1 \end{pmatrix} \quad i = 1, ..., L \tag{A.6}$$

This can then be solved using the previously explained *DLT* method. Instead of a 3×4 camera matrix, in this case it will estimate a 3×3 *Homograpy* matrix. Similar to the previous method, the following equation can be obtained from A.6,

$$x' = \frac{h_{11}X + h_{12}Y + h_{13}}{h_{31}X + h_{32}Y + h_{33}} \tag{A.7a}$$

$$y' = \frac{h_{21}X + h_{22}Y + h_{23}}{h_{31}X + h_{32}Y + h_{33}} \tag{A.7b}$$

This could be written in the form of a matrix A where x_i and X_i are the i^{th} image point and its corresponding world coordinate.

$$\underbrace{\begin{bmatrix} -X_1 & -Y_1 & -1 & 0 & 0 & 0 & x_1X_1 & x_1Y_1 & x_1 \\ 0 & 0 & 0 & -X_1 & -Y_1 & -1 & y_1X_1 & y_1Y_1 & y_1 \\ ... & ... & ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & ... & ... & ... \\ -X_i & -Y_i & -1 & 0 & 0 & 0 & x_iX_i & x_iY_i & x_i \\ 0 & 0 & 0 & -X_i & -Y_i & -1 & y_iX_i & y_iY_i & y_i \\ ... & ... & ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... & ... & ... & ... & ... \\ -X_n & -Y_n & -1 & 0 & 0 & 0 & x_nX_n & x_nY_n & x_n \\ 0 & 0 & 0 & -X_n & -Y_n & -1 & y_nX_n & y_nY_n & y_n \end{bmatrix}}_{:=A} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{A.8}$$

To solve the system of liner equations, which would estimate H, one would need at least 4 points since the matrix H has 8 DoF. Once the DLT is setup, a similar method of finding the null point vector of a type $Ax = 0$ can be performed. This would give the estimated matrix H. One should note that in the equation A.5, the matrix r_1, r_2, t is not a rotational matrix, hence obtaining the K matrix using *QR factorization* is no longer possible.

Since the matrix K is a positive definite matrix with and diagonal with zero values, K can be recognize as an invertable matrix.

$$H = \begin{pmatrix} h_1 & h_2 & h_3 \end{pmatrix} = K \begin{pmatrix} r_1 & r_2 & t \end{pmatrix} \tag{A.9a}$$

$$\begin{pmatrix} r_1 & r_2 & t \end{pmatrix} = K^{-1} \begin{pmatrix} h_1 & h_2 & h_3 \end{pmatrix} \tag{A.9b}$$

$$r_1 = K^{-1}h_1 \text{ and } r_2 = K^{-1}h_2 \tag{A.9c}$$

To obtain the K matrix, first the properties of the r_1, r_2, t matrix in (A.5) will be evaluated. Even though the matrix itself is not a rotational matrix, the two columns r_1 and r_2 are still *orthogonal* hence the dot product of the two column vectors will be zero. Furthermore, the norm of the two vectors will be equal to one (vector are of unit length).

$$r_1^T r_2 = 0 \text{ and } \|r_1\| = \|r_2\| = 1 \tag{A.10}$$

From the first constrain shown above in the equation A.10,

$$r_1^T = h_1^T K^{-T} \text{ and } r_2 = K^{-1}h_2 \Rightarrow h_1^T K^{-T} K^{-1} h_2 = 0 \tag{A.11}$$

From the second constrain in the equation A.10,

$$\begin{aligned} \|r_1\| = \|r_2\| = 1 \Rightarrow r_1^T r_1 &= r_2^T r_2 \\ \Rightarrow h_1^T K^{-T} K^{-1} h_1 &- h_2^T K^{-T} K^{-1} h_2 = 0 \end{aligned} \tag{A.12}$$

One can see that in the both constraints A.11 and A.12, the matrix $K^{-T}K^{-1}$ is a symmetric positive definite matrix. This is redefine as a new matrix called B.

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{12} & b_{22} & b_{23} \\ b_{13} & b_{23} & b_{33} \end{pmatrix} \tag{A.13}$$

Then there are standard decomposition methods that decompose a positive define matrix into two triangular matrices. This decomposition method is known as the *Cholesky decomposition* [50]. $chol(B) = AA^T$ where $A = K^{-T}$. Then the two constraints that are given in (A.11) and A.12 can be re-written as,

$$h_1^T B h_2 = 0 \tag{A.14a}$$

$$h_1^T B h_1 - h_2^T B h_2 = 0 \tag{A.14b}$$

The matrix B has only six unknowns. To apply SVD method to solve the unknowns in B, a vector $b = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{22} & b_{23} & b_{33} \end{pmatrix}^T$ can be defined. In (A.14), the two vectors of $h_1 = \begin{pmatrix} h_{11} & h_{12} & h_{13} \end{pmatrix}^T$ and $h_2 = \begin{pmatrix} h_{21} & h_{22} & h_{23} \end{pmatrix}^T$ are known, then recalling the method used in DLT (in (A.8) and A.4) one can formulate the equation $Vb = 0$,

$$\underbrace{\begin{pmatrix} h_{11}h_{21} & (h_{12}h_{21} + h_{11}h_{22}) & (h_{13}h_{21} + h_{11}h_{23}) & h_{12}h_{22} & (h_{13}h_{22} + h_{12}h_{23}) & h_{13}h_{23} \end{pmatrix}}_{v_{ij}^T} b = 0 \tag{A.15}$$

$$\underbrace{\begin{pmatrix} h_{11}^2 - h_{21}^2 & 2\begin{pmatrix} h_{11}h_{12} - \\ h_{21}h_{22} \end{pmatrix} & 2\begin{pmatrix} h_{11}h_{13} - \\ h_{21}h_{23} \end{pmatrix} & h_{12}^2 - h_{22}^2 & \begin{pmatrix} h_{12}\left(h_{11} + h_{13}\right) - \\ h_{22}\left(h_{21} + h_{23}\right) \end{pmatrix} & h_{13}^2 - h_{23}^2 \end{pmatrix}}_{v_i^T - v_j^T} b = 0 \tag{A.16}$$

In the two equations A.15 and A.16, $i = 1$ and $j = 2$. Then $Vb = 0$ in a matrix for a single image can be given as,

$$Vb = \begin{pmatrix} v_{ij}^T \\ v_i^T - v_j^T \end{pmatrix} b = 0 \tag{A.17}$$

When expand this equation A.17 for n images one would be able to and matrix of size $2n \times 6$. Solving this with SVD would give the matrix B and then applying *Cholesky decomposition* one would be able to obtain the intrinsic matrix K.

$$\underbrace{\begin{pmatrix} {}^1v_{12}^T \\ {}^1v_1^T - {}^1v_2^T \\ \dots \\ {}^kv_{12}^T \\ {}^kv_1^T - {}^kv_2^T \\ \dots \\ {}^nv_{12}^T \\ {}^nv_1^T - {}^nv_2^T \end{pmatrix}}_{V} \underbrace{\begin{pmatrix} b_{11} \\ b_{12} \\ b_{13} \\ b_{22} \\ b_{23} \\ b_{33} \end{pmatrix}}_{b} = 0 \tag{A.18}$$

A.3 Robust Homography Estimation and Image Stitching

Given the two camera matrices $P_1 = \begin{bmatrix} R_1 & t_1 \end{bmatrix}$ and $P_2 = \begin{bmatrix} R_2 & t_2 \end{bmatrix}$, one can see that there exists an expression,

$$\lambda_1 x_1 = P_1 X \text{ and } \lambda_2 x_2 = P_2 X \tag{A.19}$$

Where X is the real world 3D coordinate, and the x_1 and x_2 are the image points in the two images. The expression can further expand in to,

$$\lambda_1 x_1 = \begin{bmatrix} R_1 & t_1 \end{bmatrix} \begin{bmatrix} X \\ 1 \end{bmatrix} \text{ and } \lambda_2 x_2 = \begin{bmatrix} R_2 & t_2 \end{bmatrix} \begin{bmatrix} X \\ 1 \end{bmatrix} \tag{A.20a}$$

$$\lambda_1 x_1 = R_1 X + t_1 \text{ and } \lambda_2 x_2 = R_2 X + t_2 \tag{A.20b}$$

$$\lambda_1 R_1^{-1} x_1 = X + R_1^{-1} t_1 \text{ and } \lambda_2 R_2^{-1} x_2 = X + R_2^{-1} t_2 \tag{A.20c}$$

If the camera center is projected using the camera matrix, then the resulting projection will be a null vector. If the normalized version of a camera matrix is considered, then the camera center can be calculated using,

$$\begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} C \\ 1 \end{bmatrix} = 0 \tag{A.21a}$$

$$RC + t = 0 \Leftrightarrow C = -R^{-1} t \tag{A.21b}$$

Where R is the rotational part and t is the transnational part of a normalized camera matrix P.

Using the expression given in (A.21b), one can see that the two camera centers can be defined as, $C_1 = -A_1^{-1} t_1$ and $C_2 = -A_2^{-1} t_2$. Substituting these values to (A.20c), one could get the expressions,

$$\lambda_1 A_1^{-1} x_1 = X - C_1 \text{ and } \lambda_2 A_2^{-1} x_2 = X - C_2 \tag{A.22}$$

Since the two camera centers are equal, one could see that the two equations given in (A.22) are equal to each other then,

$$\lambda_1 A_1^{-1} x_1 = \lambda_2 A_2^{-1} x_2 \tag{A.23a}$$

$$\lambda_1 x_1 = \lambda_2 A_1 A_2^{-1} x_2 \tag{A.23b}$$

Here since the values of λ_1 and λ_2 are scalar values one could write,

$$x_1 \sim \underbrace{A_1 A_2^{-1}}_{H} x_2 \tag{A.24}$$

According to the expression given in the equation A.24, if the two camera centers are equal, then there exists a transformation, which can transform the first image into the second one, and its transformation can be expressed by the homography matrix between the two images.

A.4 Graph-based SLAM

Given an unknown state, a predicted measurement can be generated, the error can then be expressed as the difference between predicted measurement and real measurement.

$$e_i(x) = z_i - f_i(x) \tag{A.25}$$

With the assumption that the measurement error is normally distributed with zero mean, the squared error of a measurement can be expressed in the following form,

$$e_i(x) = e_i(x)^T \Omega_i e_i(x) \tag{A.26}$$

where Ω_i is information matrix of a constraint. The goal is to find the state x^* that minimizes the error given all measurements.

$$x^* = \underset{x}{argmin} \sum_i e_i^T(x) \Omega_i e_i(x) \tag{A.27}$$

One way to solve the above problem is applying iterative local linearization, which firstly linearizes the error term around a initial guess point, then compute the solution of the derivative of the squared error function, the obtained solution would be set as new working point to proceed to next iteration. With the initial guess x, the Taylor expansion provides:

$$e_i(x + \Delta x) \simeq \underbrace{e_i}_{e_i}(x) + J_i(x)\Delta x \tag{A.28}$$

where J_i is the Jacobian matrix computed in point x and Δx is small increment. Inserting above expression into A.26:

$$\begin{aligned} e_i(x + \Delta x) &= e_i^T(x + \Delta x)\Omega_i e_i(x + \Delta x) \\ &\simeq (e_i + J_i \Delta x)^T \Omega_i (e_i + J_i \Delta x) \\ &= e_i^T \Omega_i e_i + e_i^T \Omega_i J_i \Delta x + \Delta x^T J_i^T \Omega_i e_i + \Delta x^T J_i^T \Omega_i J_i \Delta x \end{aligned} \tag{A.29}$$

The expression of squared error could be further rewritten as follows:

$$\begin{aligned} e_i(x + \Delta x) &\simeq \underbrace{e_i^T \Omega_i e_i}_{c_i} + 2\underbrace{e_i^T \Omega_i J_i}_{b_i^T} \Delta x + \Delta x^T \underbrace{J_i^T \Omega_i J_i}_{H_i} \Delta x \\ &= c_i + 2b_i^T \Delta x + \Delta x^T H_i \Delta x \end{aligned} \tag{A.30}$$

the global error then can be computed as the sum of the squared errors, which is:

$$\begin{aligned} F(x + \Delta x) &\simeq \sum_i (c_i + b_i^T \Delta x + \Delta x^T H_i \Delta x) \\ &= \underbrace{\sum_i c_i}_{c} + 2\underbrace{(\sum_i b_i^T)}_{b^T} \Delta x + \Delta x^T \underbrace{(\sum_i H_i)}_{H} \Delta x \\ &= c + 2b^T \Delta x + \Delta x^T H \Delta x \end{aligned} \tag{A.31}$$

where $b^T = \sum_i e_i^T \Omega_i J_i$, and $H = \sum_i J_i^T \Omega_i J_i$

As the goal is to minimize the error function, one way is computing the derivative of the error function with respect to small increment:

$$\frac{\partial F(x + \Delta x)}{\partial \Delta x} \simeq 2b + 2H\Delta x \tag{A.32}$$

Setting the above expression to zero yields:

$$0 = 2b + 2H\Delta x \rightarrow H\Delta x = -b \tag{A.33}$$

thus the solution is:

$$\Delta x^* = -H^{-1}b \tag{A.34}$$

the new state for next iteration is then:

$$x \leftarrow x + \Delta x^* \tag{A.35}$$

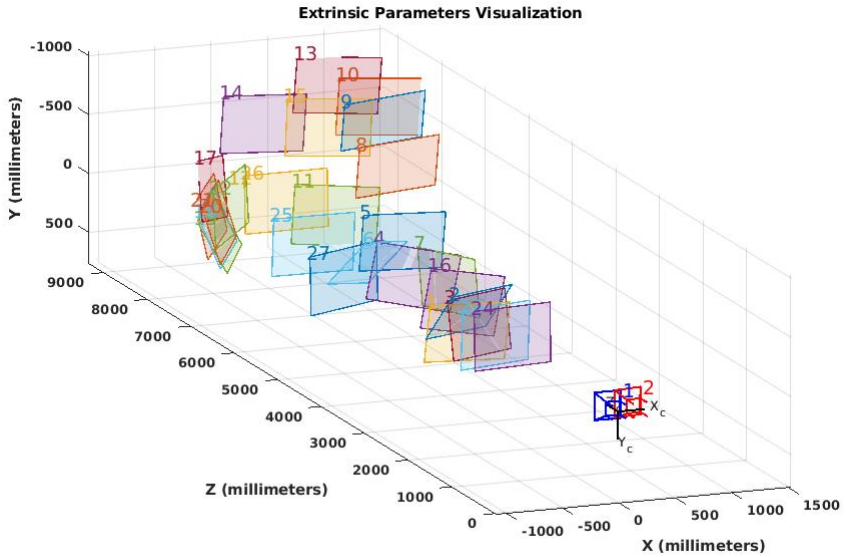
B

Results

In this chapter, the auto-generated results, figures will be illustrated and explained briefly. One should note that both ROS and OpenVSLAM have their own methods of data visualizations. Some of these methods were used directly in this thesis, and those figures are illustrated and discussed here in this chapter.

(a) **(b)**

(c) **(d)**



(e)

Figure B.1: Extrinsic matrices visualization. In Figure (e), the blue color camera icon represents the left camera while the red icon represents the left camera.

Figure B.1 shows different positions covering the entire area of the image space and the scene's respective camera poses. These camera poses are, in fact, the different extrinsic matrices for different poses. Once these matrices are multiplied with the K (intrinsic) matrix, one would be able to obtain the camera matrix P for each different position. Typically in pose estimation problems, the matrix P will be estimated, and then multiplying with the inverse of K, one would get the desired camera pose. The key important factor in Figure B.1e is that it shows how the detected images are spread in space that represents the narrow FoV value one had to deal with in this thesis.

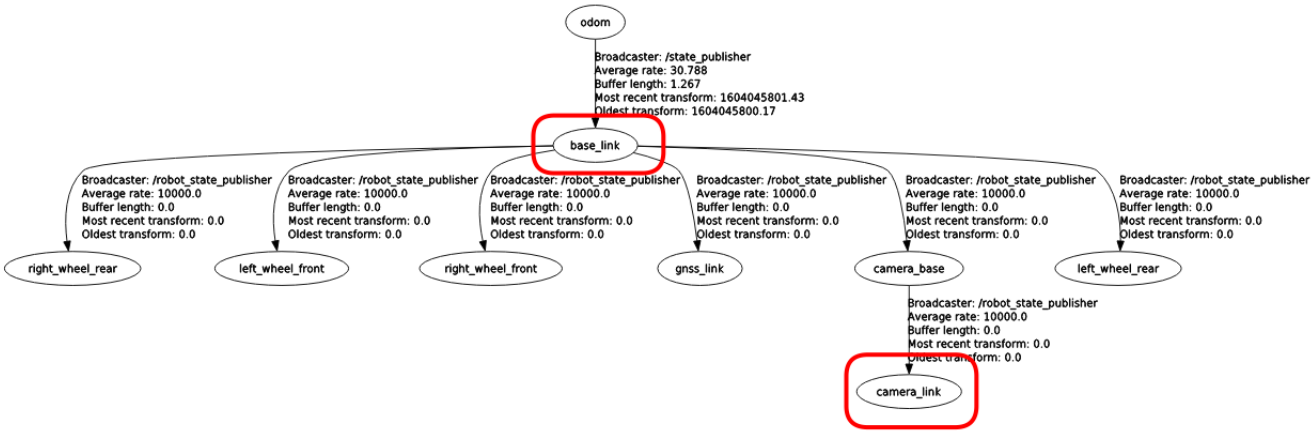


Figure B.2: ROS generated TF-tree of the current system. Here the red colored frames are the most important. The initial estimation from the OpenVSLAM will give the camera pose in a coordinate frame with the Z axis facing outwards from the camera.

```
<link name="camera_base">
  <visual>
    <geometry>
      <box size="0.3 0.25 0.1"/>>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <material name="green"/>
  </visual>
</link>

<joint name="base_to_camera_base" type="fixed">
    <parent link="base_link"/>
    <child link="camera_base"/>
    <origin rpy="0 0 0" xyz="0 0.2 0.15"/>
</joint>

<link name="camera_link">
  <visual>
    <geometry>
      <box size="0.25 0.125 0.125"/>>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <material name="blue"/>
  </visual>
</link>

<joint name="camera_base_to_camera_link" type="fixed">
    <parent link="camera_base"/>
    <child link="camera_link"/>
    <origin rpy="-1.57075 0 0" xyz="0 0 0.1"/>
</joint>
```

Figure B.3: Part of the URDF file description of the vehicle. Here the structure of the two links (coordinate frames) marked in red rectangles in Figure B.2 are defined along with the two connection parameters.

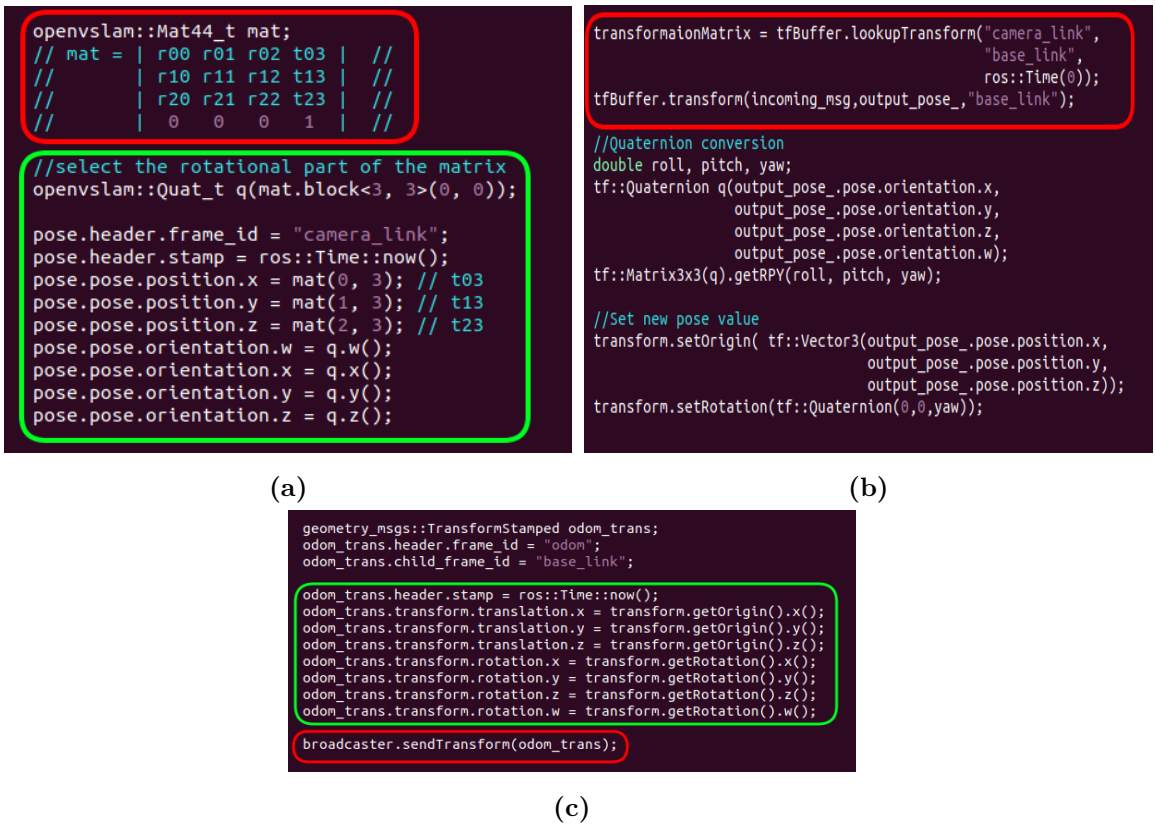


Figure B.4: Process of pose transformation between two coordinate frames.

Figure B.2 illustrates how ROS defines the connection between the internal system (in this thesis, the utility vehicle). The first branch connecting *odom* to *base_link* is defined by *TF* package. Origin of *odom* coordinate frame is the initial fixed starting point of the robot and this updates with every new pose. All the other connections are defined by a custom node, which describes the URDF description's connections. The term *Broadcaster* is the name of the node publishing the respective coordinate status and values. *Average rate* refers to the rate at which the messages are published. *Buffer lenght* is the number of messages that the respective transformation can hold. The custom node is set to 20 messages, but since this figure was captured at the very beginning with no incoming messages, it shows the value of zero. The bugger in *TF* node is quite big since it all captures other messages which are not related to the current system. The two timestamps *Most recent trasform* and *Oldest trasform* represents the timestamps of the most recent and the oldest transformations in the buffer (This is a first-come-last-out buffer).

The TF structure is defined from top to bottom, meaning in Figure B.2, the top most frame is called the *parent frame* and the *base_link* bellow it called the *child frame*. When calculating transformations in the next section of the tree, *base_link* will be denoted as the *parent*, and all the other links will be called as the *child* links. This hierarchy is important when defining values in the URDF description, as illustrated in Figure B.3. First, one would define the structure of the selected coordinate frame. In this example the *camera_link* is defined as a cube. Next, the connection should be presented between the two links. First as illustrated in Figure B.2, *camera_link* is connected to *camera_base* and then the *camera_base* is linked to *base_link*. When defining these connections, it

is important to properly define the distances and the orientations between each link (as described in Section 3.2.1).

Figure B.4 illustrated the process of transforming a pose from one coordinate to another, in this case from transforming the estimated camera pose to the world coordinates. First, as shown in Figure B.4a, the estimated pose matrix from *OpenVSLAM* needs to be converted to a ROS message. The red color rectangle represents the matrix definition in *OpenVSLAM* library, which is a 4×4 matrix containing the rotational part and the transformation vector from the camera pose estimation. Figure B.4b shows the instance of capturing the newly estimated camera pose. The red color box in Figure B.4b illustrates the transformation process between the needed two coordinates frame. One would need the transformation between the *base_link* and *camera_link*. This transformation is saved in the *tfbuffer* object (defined by ROS-TF library). The incoming message from Figure B.4a will then be transformed accordingly such that the *output_pose_* describes the transformation and the orientation of the link *base_link*. In this thesis, one should note that the roll and the pitch values were ignored as described in Section 3.2.1. Finally, as shown in B.4c the newly defined transformation will be updated in the *odom* transformation (shown in the green color rectangle) and will be broadcast (shown in red color box) accordingly.

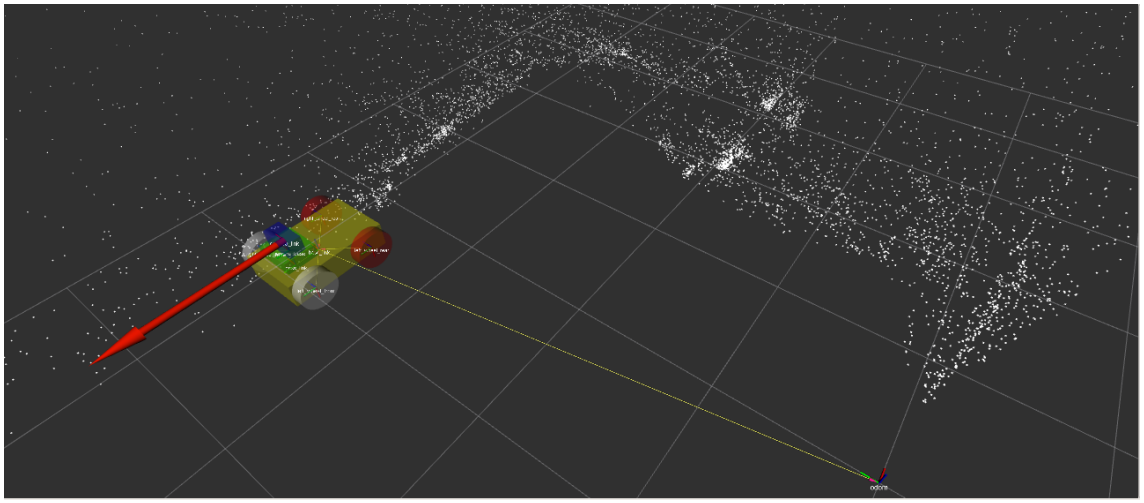
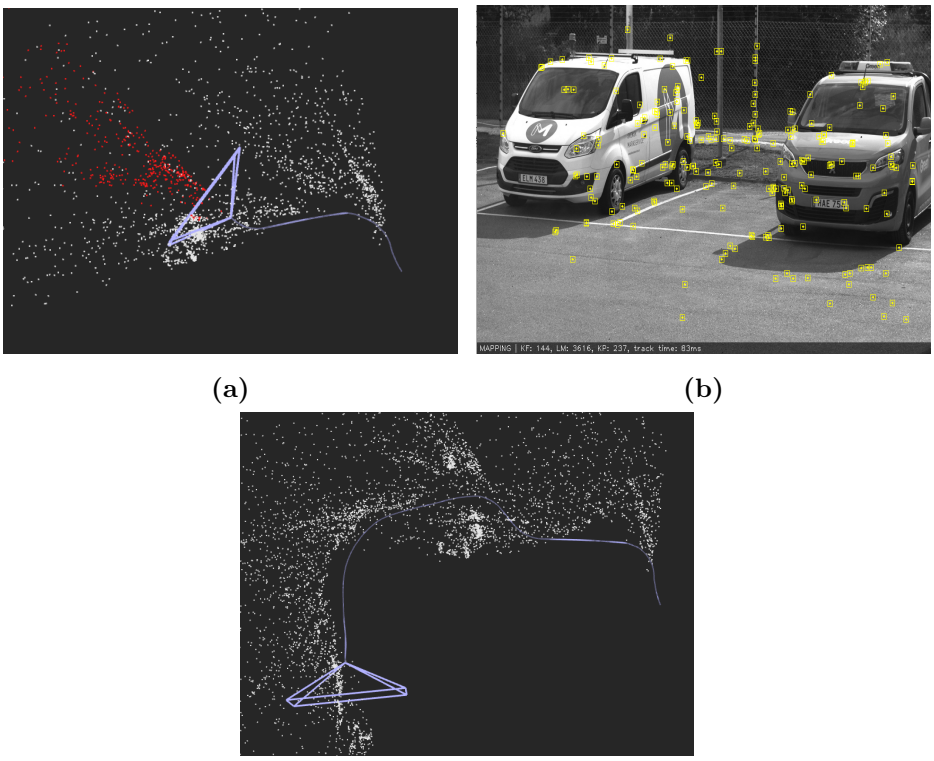


Figure B.5: Partial migration from *Pangoine viewer* to *ROS-Rviz*. Red color arrow represents the heading direction of the vehicle. White color dots represents the 3D point cloud generated by OpenVSLAM similar to what one would see in *Pangoine viewer*.

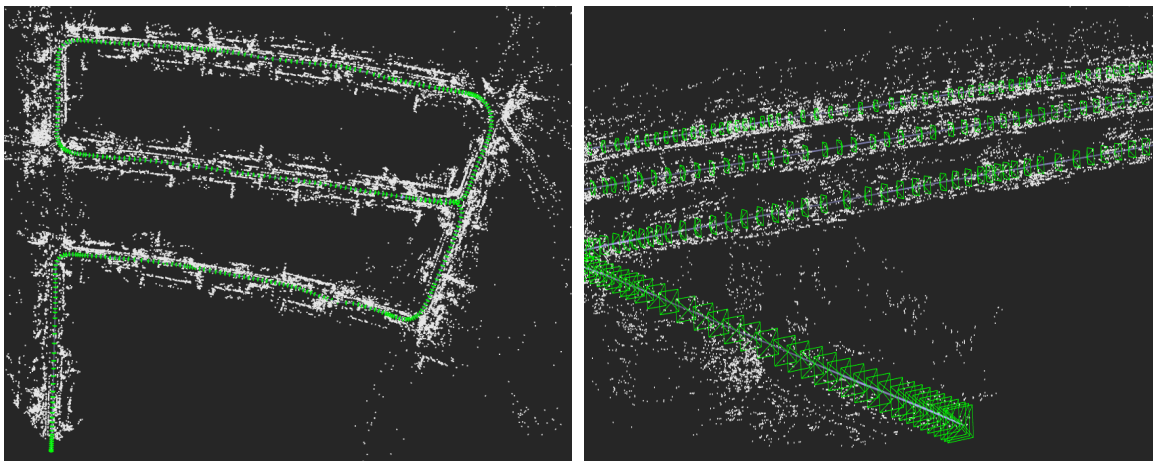
A partial implementation of *Pangoine viewer* visualization tool in ROS is illustrated in figure B.5. As mentioned in Chapter 5 it is important to migrate to a *Pangoine viewer* free OpenVSALM implementation. As the first step of that implementation, few functions in *Pangoine viewer* has currently been implemented along with ROS-Rviz.



(a) **(b)**

(c)

Figure B.6: *BARMARK* dataset with OpenVSLAM library. Figure (a) is the respective map status when capturing the image shown in Figure (b). The yellow color points in Figure (b) represents the currently detected and tracked feature points. These points are marked in red color in Figure (a). White color points represent the already saved features (landmarks). Figure (c) shows the dataset's final map, and the purple color path is the estimated trajectory.



(a) **(b)**

Figure B.7: The estimated trajectory of the KITTI dataset visualized in the *Pangoine viewer*. The green color pyramids in the zoomed-in version of the same trajectory shown in Figure (b) are the estimated keyframes.