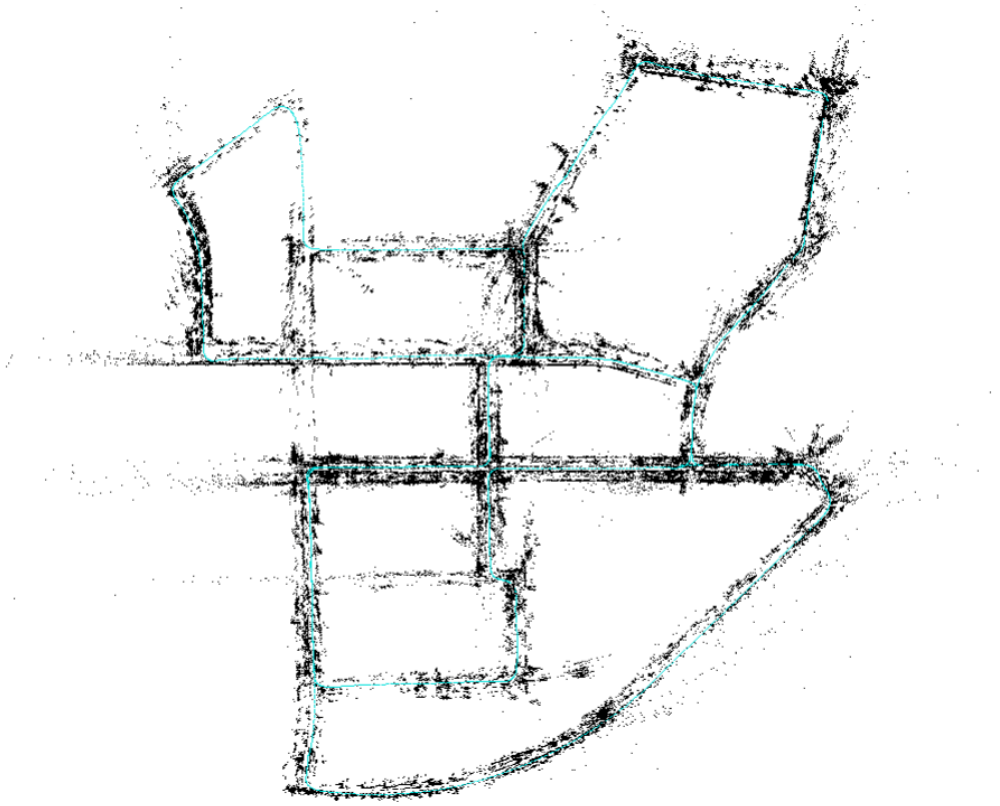
CHALMERS
UNIVERSITY OF TECHNOLOGY



Simultaneous localization and mapping for vehicles using ORB-SLAM2

Master's thesis in Systems, Control and Mechatronics and Complex adaptive systems

MARCUS ANDERSSON
MARTIN BAERVELDT

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

MASTER'S THESIS IN SYSTEMS, CONTROL AND MECHATRONICS AND COMPLEX ADAPTIVE SYSTEMS

Simultaneous localization and mapping for cars using ORB2-SLAM

MARCUS ANDERSSON
MARTIN BAERVELDT

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2018

Simultaneous localization and mapping for cars using ORB-SLAM2
MARCUS ANDERSSON
MARTIN BAERVELDT

Supervisor: Christian Berger, Department of Computer Science and Engineering
Examiner: Ola Benderius, Department of Mechanics and Maritime Sciences

Master's Thesis 2018:85
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A complete sequence of mapping and localization

Typeset in LaTeX
Gothenburg, Sweden 2018

Simultaneous localization and mapping for cars using ORB-SLAM2
Master's thesis in Systems, Control and Mechatronics and Complex Adaptive Systems
MARCUS ANDERSSON
MARTIN BAERVELDT
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems Chalmers University of Technology

Abstract

This thesis details work developing a containerized version of ORB-SLAM2 implemented in the open source software framework OpenDLV as well as testing it both on established datasets as well as on gathered data in a real traffic scenario in the Gothenburg area. Parallel to the real traffic scenarios this thesis also evaluates the feasibility for ORB-SLAM2 to be run on a formula student race car, due to the authors involvement in the Chalmers formula student driverless project which will compete with an autonomous race car in the formula student germany competition. The thesis also evaluates ORB-SLAM2 as a SLAM solution on a regular car and the strengths and weaknesses of the system in that area.

Keywords: Localization, mapping, SLAM, ORB-SLAM visual odometry, autonomous vehicles, stereovision, OpenDLV, 3D reconstruction

Acknowledgements

A very special thanks to Linus Eiderström Swahn and Pontus Pohl for their collaboration in implementing ORB-SLAM2 into OpenDLV, without their computer science expertise this project would not have gone this smoothly! And a big gratitude to the developers of OpenDLV, Christian Berger, Ola Benderius and Björnborg Nguyen who have always been there when we were in desperate need of humble support. Also thanks to Camilla Apoy, Arpit Karsolia and Fredrik von Corswant at the Revere lab for hosting us, offering their expertise in all areas and providing us with a test vehicle. And a special gratitude to everyone in the Chalmers formula student driverless team that we shared workspace with during our thesis!

Marcus Andersson, Gothenburg, 05 2018
Martin Baerveldt, Gothenburg, 05 2018

Thesis advisor: Christian Berger
Thesis examiner: Ola Benderius

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

Simultaneous localization and mapping (SLAM) is a highly relevant algorithmic approach in robotics and computer vision to realize autonomous systems. The capability to map an environment and resolve your position is very important if one is to achieve autonomy. The topic has been studied extensively during the last decades and different methods have emerged to solve this problem.

The earliest method is EKF-SLAM which uses an *extended Kalman filter* (EKF) and was developed in 1990 [24]. The EKF uses observations of the environment to compute and improve upon an estimation of the position of a landmark and the robots position. However, a problem is that all landmarks are stored as state variables and as such the computational cost grows in the order of N^2 [21]. Another approach is to use a network based representation, in which an observation is stored as a node with the edges representing the relation between the nodes [12].

A SLAM method that emerged out of EKF-SLAM is FAST-SLAM [18]. A problem with Kalman filter SLAM methods is that they scale poorly for larger maps, which stems from the quadratic growth of state space of an observed then incorporated landmark. The FAST-SLAM architecture differs from Kalman filter based algorithms by embedding a particle filter for pose estimation, and using the observation that if the robot's path is known, the landmarks will be independent from each other. This results in every particle receiving K independent landmark estimation problems using a Kalman filter, which in turn results in a map that scales logarithmically with the number of landmarks obtained.

1.1.1 Visual SLAM

The sensor systems in use are another important detail to consider for a SLAM system. While a lot of sensors could be used to obtain a state estimation, the specific sensors used are usually a trade-off between detail, computational complexity, and cost. One particular approach is the use of only a monocular camera as a sensor which is appealing due to wide availability of monocular cameras and the simplicity and cost of the hardware. The first paper demonstrating the usage of a monocular camera in a real-time SLAM system was MonoSLAM [5]. MonoSLAM used an EKF to perform the mapping and landmarks were given by features in the acquired

images. A motion model is used to correlate different pictures taken at different times so as to produce 3D-information about the features and localize the robot.

1.2 Motivation

This master's thesis is performed at the Chalmers Revere research facility which facilitates automotive related research by providing test vehicles and platforms. As a part of that work Revere is the developer and primary user of the open source framework OpenDLV, a framework for software in use on autonomous vehicles. It is in the interest of Revere to extend OpenDLV with more features to make the framework more complete and to develop the ecosystem. As ORB-SLAM2 [19] is one of the most widely used visual SLAM systems it is of interest to evaluate to what extent ORB-SLAM2 can be used on autonomous vehicles and provide it within the OpenDLV framework. Therefore the main aim of this thesis is to provide an implementation of ORB-SLAM2 within the OpenDLV framework as well as evaluate the performance of ORB-SLAM2 in driving scenarios.

1.2.1 Chalmers formula student driverless

Both authors are part of the *Chalmers formula student driverless* (CFSD) 2018 team. Formula student spans back to 1998 where the society of automotive engineers and the institute of mechanical engineers joined together to create the formula student competition. All around the world university students are participating in designing and manufacturing their own race car that they will compete with, the competition is not only about car performance, but takes the design choices, cost efficiency and more into account when scoring. Events are being held world wide with unified rules that enables the teams travel around the world and compete in different locations with their race cars.

In 2017 the formula student organization introduced the driverless class, a new set of students were now able to participate with more focus on software development rather than mechanical engineering. Most teams, including Chalmer's driverless team inherited their former race car developed for the formula student electric class in the 2017 competition, this alleviates the scope of the driverless project and sets the focus on subjects outside the mechanical engineering realm. One of these subjects are SLAM, a very important topic for autonomous driving and critical to the CFSD team to perform well on the 2018 competition.

While developing a SLAM solution based on a lidar sensor the authors wanted to experiment with visual SLAM and evaluate the possibility to use such a solution for the competition. To be able to perform well with the other subsystems of the CFSD race car, the SLAM solution must robustly find cones. Cones are the main feature of the race track and are used to plan the drive path, set velocity profiles and more. ORB-SLAM2 is a fresh and exciting algorithm that was chosen to be implemented and evaluated for general automotive applications as well as in the formula student competition.

1.3 Aim

- Implement an executable version of ORB-SLAM2 in OpenDLV for monocular or stereo cameras for use on an autonomous vehicle.
- Minimum running frequency of 5 Hz on commonly used CPU's.
- The accuracy of the system will be evaluated with common test datasets, with the goal of an error with the translation being less than 5 % of the path length.
- The accuracy of the SLAM solution will be compared to a lidar based SLAM system used in the Chalmers Formula Student Driverless team for a racing scenario.

1.4 Limitations

We will not be focusing on implementing support for RGB-D cameras and focus only on the implementation for stereo and monocular cameras.

1.5 Specification of issue under investigation

Can ORB-SLAM2 be used as a robust SLAM-solution in a continuously running system for autonomous vehicles in OpenDLV?

The first question to verify is if ORB-SLAM2 can be used on an autonomous vehicle in OpenDLV and furthermore that the execution time is such that the system is able to be used in real time. The way to verify this is by testing the performance of our system on gathered data that can be compared to ground truth. Furthermore, to enable comparison to other implementations we can use common test datasets like KITTI [11] to evaluate the performance of our system. KITTI is a dataset gathered by driving a car around an urban environment and ground truth is provided by a Velodyne lidar and RTK-GPS. It is meant for use in research in the computer vision and autonomous driving field and it was used to benchmark both ORB-SLAM and ORB-SLAM2 against other SLAM systems. [20, 19]

To what extent is ORB-SLAM2 comparable in performance with current solutions utilizing lidar based SLAM?

The current state of the art uses expensive sensors like lidar to gather data for the SLAM system. lidars can be very expensive and as such there is interest to bring costs down to enable further usage of SLAM systems in industry. The question to investigate is if comparable performance can be achieved using only a monocular or stereo camera and ORB-SLAM2 and if not, how much they differ in performance. To test this we will compare the performance of our system with a lidar based SLAM system currently in use by the CFSD team. By comparing the accuracy of the mapping of the landmarks used in the competition we can determine how ORB-SLAM2 compares to a lidar based approach.

1.6 Algorithm overview

1.6.1 ORB-SLAM

The purpose of this thesis is to study the SLAM system ORB-SLAM2 which uses monocular, stereoscopic and depth cameras and implement the method in an autonomous vehicle [19]. ORB-SLAM2 builds on its predecessor ORB-SLAM outlined in the paper *ORB-SLAM: A Versatile and Accurate Monocular SLAM System* [20]. The structure contains three threads that work in parallel, the tracking, local mapping, and loop closure which is visualized in Figure 1.1. The input to ORB-SLAM is an image from which the first task is to extract ORB features. This is done by finding edge features in the input frame based on a feature extraction method called FAST, then creating a descriptor for the features with a method called BRIEF. A descriptor represent the features in an image and based on which method is used the descriptor can take different forms. BRIEF represents the features as a string of binary numbers that are evaluated based on the different pixel intensities between the features [23]. Then the information is sent to the different threads.

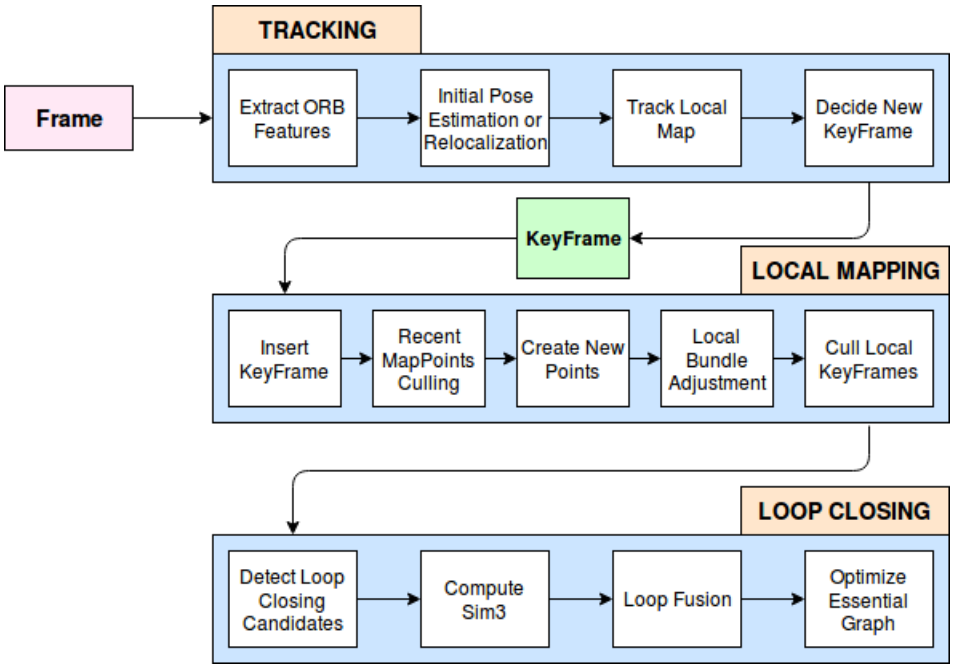


Figure 1.1: Algorithm structure for ORB-SLAM.

1.6.1.1 Tracking

The tracking thread uses each frame to find the localization of the camera and also decides which frame should be used as a keyframe. Keyframes are the set of selected frames used to build the map. ORB-SLAM uses a 'survival of the fittest' approach to finding the most useful frames. An initial pose estimation is found by using feature matching on the chosen keyframe and the previous keyframe. Thereafter it is used to optimize the current pose with a motion model to predict the correspondent map

points on the keyframe to obtain the optimal 3D reconstruction. If the tracking is lost from a previous frame, a global localization algorithm is performed finding a new suitable keyframe and matching it to the current frame to find a map fit that has the largest amount of inliers. A successful tracking step results in finding a camera pose estimation and an initial set of feature matches from which a local map can be projected [20].

1.6.1.2 Map

The map is built up by the acquired keyframes and the map points contained therein. When a keyframe is acquired it is placed in a covisibility graph which is a graph with keyframes for nodes and the edges being the map points observed in common by different keyframes. The weighting of the edges is the number of map points observed by both of the keyframes. To prevent unbounded growth of the graph, keyframes are culled if they are deemed to be too similar to other keyframes. Map points are also removed if they are observed by too few keyframes at a time. By doing it this way keyframes can be inserted quite liberally and then subsequently removed if they are found to be redundant [20].

1.6.1.3 Loop closing

The final thread performs loop closing to try to recognize if this location has been visited before. Loop closing is essential as it allows the system to update beliefs about the location and determine the drift accumulated while the loop was traversed. In ORB-SLAM this is achieved so that for every new keyframe loop closure is attempted by looking at the covisibility graph and the similarity between keyframes. A similarity score is computed by looking at the binary *bag of words* representation of the keyframe. If a keyframe has more similarities with a newly acquired keyframe than its neighbors in the visibility graph it is a candidate for loop closure. If three loop candidates that are connected to each other are found the loop is accepted. The covisibility graph is then updated accordingly [20].

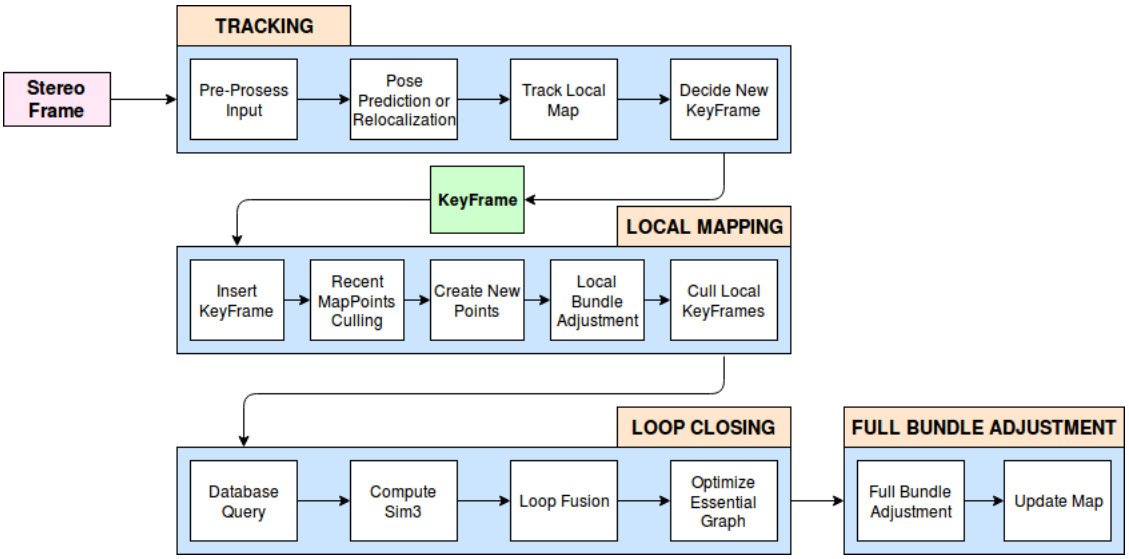


Figure 1.2: Algorithm structure for ORB-SLAM2.

1.6.2 ORB-SLAM2

ORB-SLAM2 is an extension to ORB-SLAM that also enables the usage of either an RGB-D camera or stereo camera instead of a monocular camera [19]. As its predecessor it uses three threads that work in parallel. First the tracking thread, that is used to match the local map with the extracted features for every frame, and also to the minimize the reprojection error to together localize the camera. The second thread local mapping is used to optimize and manage the local map through local bundle adjustment. The last thread loop closing performs pose-graph optimization to correct drift and detect loops. Another new aspect in ORB-SLAM2 is that in the loop closing thread a fourth thread is initialized, as seen in Figure 1.2. This thread performs a full bundle adjustment of the entire map to achieve an optimized and consistent reconstruction of the environment.

ORB-SLAM2 also uses bag of words, which is a place recognition module within the system. It is used for loop detection, if there is already a mapped local map it reinitializes itself and if the system has lost track of where it is. One other important aspect of ORB-SLAM2 is the covisibility-graph, which is used to link two keyframes that have similar observations of points. The graph is used to define a local neighbourhood which enables the tracking and mapping to work locally.

For visual place recognition ORB-SLAM2 uses ORB-features, which consist of a combination of the two algorithms FAST and BRIEF. For stereo cameras ORB-features are extracted from both the left and right images, the left image is used as a reference image for feature matching and its ORB-features are then matched against the ORB-features of the right image. This results in ORB-features with depth-information enabling direct 3D-mapping without triangulation [19].

2
Theory

2.1 Microservices

As vehicles become more advanced and automated, the complexity of the deployed software also grows. As that happens the design principles of *continuous integration and continuous deployment* become more important. At the Revere lab this has been addressed by developing a software framework based on the concept of microservices [3]. A microservice is defined as developing an application as a suite of small services, each running in its own process and communicating with lightweight mechanisms. This is in contrast to a more traditional monolithic application where an application would consist of three parts, a front end user interface, a database and a server side application. The server side application will handle most of the logic, and if some logic needs to be changed a new version needs to be built and deployed. In contrast a microservice style architecture can be independently deployed and verified. In addition the smaller services provide a firm modular boundary allowing different teams to work in parallel and manage different services as long as an agreed upon communication structure is used [8].

The software framework OpenDLV developed at Revere is designed to adopt this principle through the use of Docker containers. The containers all run one individual service and communicate using an agreed upon standard message set. The communication is done over a UDP multicast instance that all services subscribe to but the messages are differentiated with different ID's enabling each service to subscribe only to the messages it is interested in [3].

2.2 Rigid transformation representation

When mapping and estimating location in a 3D environment it is important to have a coherent way to represent the transformations. The standard in computer vision and robotics is to use Lie groups [6]. The Lie groups used in this thesis can be seen in Table 2.1.

2.3 Visual odometry

In any SLAM system it is important to be able to determine the odometry of the robot. Classically this has been achieved with sensors such as wheel encoders that

Groups	Description	Dim.	Matrix Representation
SO(3)	3D rotation	3	3D rotation matrix
SE(3)	3D rigid transformation	6	homogeneous 4x4 matrix
Sim(3)	3D rigid transformation with scale	7	homogeneous 4x4 matrix

Table 2.1: The Lie groups used in this thesis to represent the transformations

measure the number of revolutions of the wheel which can be translated to a linear displacement. The problem with this approach is that it doesn't account for slippage and as such is subject to drift errors. Another approach is the use of an *Inertial Measurement Unit*(IMU). An IMU uses accelerometers and gyroscopes to calculate acceleration and rotation speed and this can then be integrated to compute velocity and position. However an IMU is also subject to drift, especially since the errors accumulate due to the integration procedure. Therefore an IMU is typically combined with an additional sensor to provide correction.

A different approach is the use of a camera to estimate the odometry of a robot, this is called visual odometry. Cameras have the advantage of being low cost and yet the images can provide a large amount of information. Furthermore a camera is a passive sensor so it will not suffer from interference. The problem is that it requires a large amount of image processing and as such the computational cost is large. There is also a need for good features that are easy to extract yet robust [1].

Visual odometry is very different depending on if you have a stereo or monocular camera. We will give a brief overview of both approaches.

2.3.1 Monocular

Using only a monocular camera introduces an additional difficulty in that it is harder to acquire 3D points from the image due to the need for sequential frames to perform the 3D reconstruction. Furthermore a monocular camera also faces the problem of scale variance between frames which cannot fully be resolved meaning that the translation cannot be exactly determined. The first step of monocular based visual odometry is to perform some kind of initialization to create the transformation from 2D camera space to the 3D space. After this initialization the features in two subsequent frames can be matched to each other and then the features in the newest frame can be matched to the acquired 3D points in the previous frame. A transformation can then be calculated by minimizing the sum of squared differences between these two points. The matched feature pairs from the two latest frames can then be triangulated into 3D points using the transformation acquired during the initialization. The procedure is then repeated with a new frame [29][1].

2.3.2 Stereo

In the stereo case, the 3D points can be acquired from a single frame by observing the features in both the left and right images. This requires rectifying and matching

the features in both images which introduces a slight additional computational cost. These features can then be matched with the features in the following frames to obtain a motion transformation using the sum of squared differences. A *random sample consensus* (RANSAC) [7] algorithm can be used to discard outliers and refine the transformation. This is then repeated for each frame. Compared to the monocular case there is no need for an initialization step which means that there will be no errors introduced by scale drift [1][29].

2.4 Pinhole camera model and intrinsic parameters

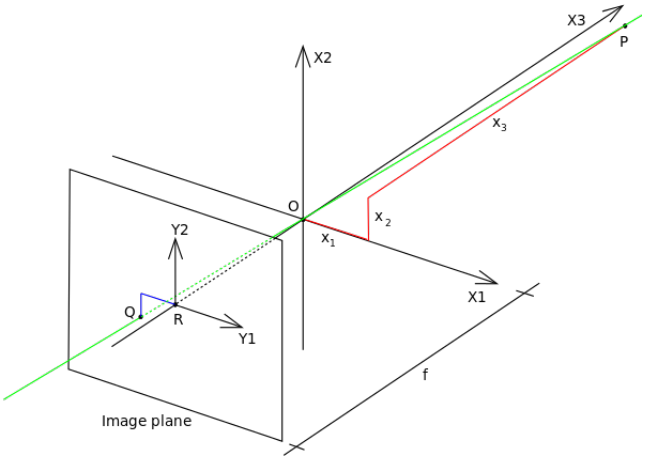


Figure 2.1: The pinhole camera model and its geometrical properties.

In order to project 3D world coordinates onto an image plane we need a model for how this is achieved in a camera. The model most commonly used is the pinhole camera model which assumes that the light hitting the film only goes through a small aperture in a barrier at a position O which is also referred to as the cameras center point. The light passes through this aperture and then projects an image of the scene on the film. The image is placed a distance f from O where f is the focal length of the camera model. If we consider a coordinate system with the origin at O and the $\hat{z}$-axis defined in the viewing direction of the camera and the $\hat{x}$ and $\hat{y}$ axes spanning the image plane. Consider also a point $\mathbf{P} = (x_p, y_p, z_p)$ in this coordinate system. If we also consider a 2D coordinate system in the image plane with origin at the intersection of the $\hat{z}$-axis of the 3D world coordinate system and the point $\mathbf{Q} = (X_q, Y_q)$ within the image plane. A Figure showing this setup can be seen in Figure 2.1. With this model we can define a transformation from $\mathbf{P}$ and $\mathbf{Q}$ by observing that a triangle can be formed between $\mathbf{Q}$, the origin of the image plane and $\mathbf{O}$. An opposite triangle can be formed with $\mathbf{P}$, a projection of $\mathbf{P}$ on $\hat{z}$ and $\mathbf{O}$. Using the law of similar triangles we arrive at the following relationship [9][28].

$$Q = \begin{pmatrix} X_q \\ Y_q \end{pmatrix} = \frac{f}{z_p} \begin{pmatrix} x_p \\ y_p \end{pmatrix} \tag{2.1}$$

However the image that is acquired from the camera is digital, and is thus discretized into pixels. This introduces some additional parameters and complicates the transformation somewhat. The first problem is that a digital image usually has a defined origin in the upper left corner so we have an offset between the projection and the digital image which is represented by a translation vector (c_x, c_y). In addition we need to define a relation between the distance between discretized pixels and the same distance in the image projection. We therefore replace f with f_x and f_y respectively in the relation above. So in a digital image the relation can be written as

$$Q = \begin{pmatrix} X_q \\ Y_q \end{pmatrix} = \begin{pmatrix} f_x \frac{x_p}{z_p} + c_x \\ f_y \frac{y_p}{z_p} + c_y \end{pmatrix} \tag{2.2}$$

To write this transformation more compactly we can instead use a homogeneous coordinate system. Rewriting the transformation in matrix form we get

$$Q = \begin{pmatrix} f_x x_p + c_x z_p \\ f_y y_p + c_y z_p \\ z_p \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = \mathbf{MP} \tag{2.3}$$

This can be decomposed further to get the camera matrix $\mathbf{K}$

$$Q = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{0} \end{pmatrix} \mathbf{P} = \mathbf{K} \begin{pmatrix} \mathbf{I} & \mathbf{0} \end{pmatrix} \mathbf{P} \tag{2.4}$$

Where $\mathbf{I}$ is the 3x3 Identity matrix and $\mathbf{0}$ is the 1x3 null vector. $\mathbf{K}$ therefore encodes all the information needed to transform a world coordinate to a digital pixel coordinate for anyone camera hence the name camera matrix. It is also called the calibration matrix. These are called the cameras intrinsic parameters [13][9].

2.4.1 Extrinsic parameters

If the camera is moving it is no longer feasible to define the aperture as the origin of the world coordinate system. In this case to properly project a point in the world to an image we also need a transformation from the world coordinate system to the camera coordinate system. In a 3D system we need a 3x3 rotation matrix $\mathbf{R}$ defining the rotation and a translation vector $\mathbf{t}$ defining the translation from the world coordinate system origin to the camera system origin. This can be formulated as follows

$$\mathbf{P} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{pmatrix} P_w \tag{2.5}$$

Where $\mathbf{P}_w$ is the point in world coordinates. Substituting $\mathbf{P}$ with this expression and simplifying gives

$$\mathbf{Q} = \mathbf{K} \begin{pmatrix} \mathbf{R} & \mathbf{t} \end{pmatrix} \mathbf{P}_w \tag{2.6}$$

$\mathbf{R}$ and $\mathbf{t}$ plays an important role in visual odometry and visual SLAM as the pose of the camera in the world coordinate system can be obtained by the relation

$$\mathbf{C}_w = -\mathbf{R}^T\mathbf{t} \tag{2.7}$$

Where $\mathbf{C_w}$ is the position of the camera in the world coordinate system. In SLAM this is equivalent to the pose of the robot [13].

2.4.2 Epipolar geometry

With the pinhole camera model a number of geometric relationships can be extracted for stereo image pairs. Defined as two camera views looking at the same 3D scene from different positions the geometrical relationship can be used to find depth information of points of interest that can be observed in both camera images. The method of transforming a 3D space into a 2D plane with triangulation is a common method in computer vision and for the pinhole model is called perspective projection.

An arbitrary point of interest observed in both image planes can be defined as X, to best describe the real camera which places the image plane not in the camera origin, but behind the focal center. The pinhole camera model places the virtual image plane in the optical center, which lies in front of the focal centre [9].

2.4.2.1 Epipolar points

Due to the pinhole camera model the optical centers of a stereo camera can be projected into each other's image plane. These two points are called the epipolar points. These epipolar points and the optical centers of respective camera form a line which travel through all four points [9].

2.4.2.2 Epipolar line

The epipolar line consists of the epipole point and the reprojected point x_r of X in the respective camera. The epipolar line is therefor dependent on the point of interest X viewed by the two cameras. As the epipolar line is a function of X, the set of X then defines a set of epipolar lines. These epipolar relationships then creates constraints that results in useful mathematical properties.

For two cameras where the relative position is known and the projection points and epipolar points are known, the observed point X can then be triangulated by knowing that the projection lines will intersect at the position of X. Then the reprojected points in the left and the right images are enough information to triangulate X and get the 3D information [9].

2.4.2.3 Epipolar rectification

The goal of rectifying a stereo image pair is to transform the focal planes of each camera so the epipolar lines created from the same observed point becomes parallel,

a common choice is to transform the focal planes to create horizontal epipolar lines. The resulting rectification creates a new perspective projection matrix i.e. keeping the optical centre of the cameras static, but changing the world coordinate to camera coordinate system transformation.

To achieve proper rectification the focal plane are rotated as described above, then the baseline are kept intact and hold the constraint on the epipolar line being parallel with each other. The new x-axis of both camera optical centres should now be parallel to the baseline, while reprojected points have the same vertical position for both left and right image [9].

2.4.2.4 Disparity and depth

With the rectified properties of the two images, disparity and depth can be estimated through Equation 2.8 [15]. As visualized in Figure 2.2, the disparity d is the distance $x_1 - x_2$ between two corresponding pixels pointing at the same feature P in the world coordinate frame $[X, Y, Z]$. f is the focal length and b is the baseline which is the measured translational distance between the cameras.

$$\frac{d}{b} = \frac{f}{Z} \tag{2.8}$$

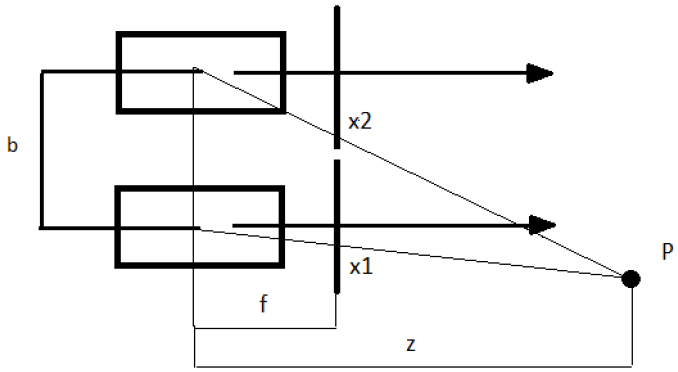


Figure 2.2: The mathematical properties that can be extracted from a rectified stereo image pair.

2.5 ORB features

2.5.1 FAST keypoints

There are a number of methods to extract image features like Moravec, SIFT, Harris and SUSAN. Moravec is one of the earliest methods and test each image pixel by evaluating the similarity of a patch centered on the current pixel to patches of bigger size which are nearby. SIFT, Harris Stephens and SUSAN are reliable corner detection methods proven to give high quality features but lack the computational

efficiency for real-time feature extraction. FAST was developed to reliably detect corner features with the necessary frame rate for real-time applications. It uses a pixel as point of interest as seen in Figure 2.3. To be a valid interest point the selected pixel must be over a certain intensity threshold, if it is valid a Bresenham circle is applied around it. For it to be classified as a corner N of the pixels in the surrounding Bresenham circle must have an intensity that is below or above the interest point with minimum of a set intensity threshold.

To achieve the computational efficiency needed to perform this algorithm the pixel intensities of pixel 1, 5, 9, and 13 of the Bresenham circle is compared with the interest point. These pixel intensity values should hold when evaluating them against the intensity threshold. If three of the four pixel intensities are not satisfying the point of interest intensity and the intensity threshold, the point of interest is not valid and is therefore rejected. If the four pixels hold under scrutiny the remaining N pixels in the Bresenham circle will be evaluated with the same criterion. This evaluation is then iterated for all pixels in the image [22].

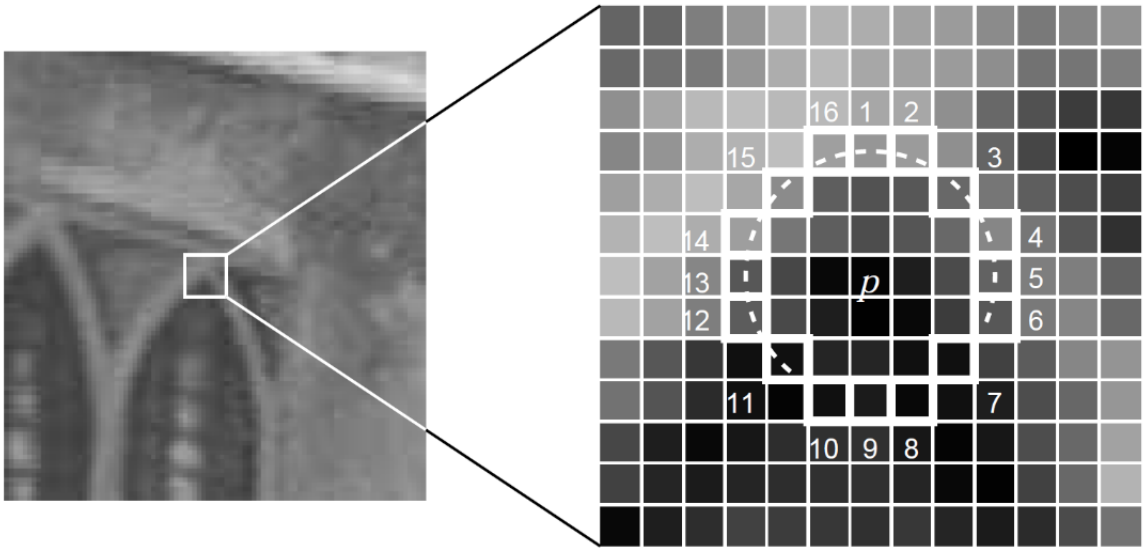


Figure 2.3: The point of interest p and the surrounding Bresenham circle with radius 3 [22].

2.5.2 BRIEF descriptors

The use of BRIEF descriptors is one method that stems from the increasing demands of a more computational restricting working environment where computer vision based algorithms needs to be more efficient. Older methods like SIFT and SURF tends to suffer from large memory consumption with large set of features. A SIFT descriptor is built with local gradient histograms and are stored as a vector of size 128. For real-time applications the SIFT descriptor then becomes too computationally inefficient [4]. The SURF descriptor also computes the local gradients histograms but stores them as floating points in a size 64 vector. SURF addressed

performance related problems by doing the gaussian smoothing as a filter approximation with a square shape, combined with the integral image that generates the sum of pixels within a subgrid of rectangles. However, for the real-time applications SURF is still too expensive since the amount of descriptors needed to be stored does not scale well.

BRIEF use the approach of directly computing binary strings from patches extracted from the image. The pixel intensity values are evaluated pair-wise to acquire the unique bits in each bitstring. For each smoothed image patch a test is set up as defined in [4] :

$$\tau(p; x, y) = \begin{cases} 1, & p(x) < p(y) \\ 0, & \text{otherwise} \end{cases} \tag{2.9}$$

where $p(x)$ is the pixel intensities at $x = (u, v)^T$ in the image patch under scrutiny. The set of n unique pixel pairs $(x, y)^T$ describe the set of binary tests.
The BRIEF descriptor then obtained by [4]:

$$f_n(p) = \sum_{i=1}^{n} 2^{i-1} \tau(p; x_i, y_i) \tag{2.10}$$

This generates an n-dimensional bitstring. The metric used to match these descriptors is the Hamming distance. The Hamming distance can be described as evaluating the number of different symbols for the same positions in two different strings of same length.

2.6 Bag of words

In order to recognize already visited places a hierarchical bag of words structure is created based on an image database. To achieve a fast and manageable feature matching in the big image datasets, the bag of words method uses a visual vocabulary to represent image features. The vocabulary is created by extracting features from a diverse dataset of training images. All features found in the training process is then discretized into visual words. The hierarchical bag of words structure can be described as a tree, where the branches are the results of clustering the descriptors with k-median clustering with k-means++ seeding [2], and for each new lower level branch new clusters are made, all the way down to the leaf which is the word representation of the vocabulary. To compute the bag of words vector for an image, its features go through the tree from the stem to the leaves and at each branching it chooses the branch to which it has the minimum Hamming distance to the corresponding median feature [10].

2.7 Bundle adjustment

After a 3D reconstruction has been acquired it can be refined using *Bundle adjustment*. Bundle adjustment is defined as the problem of refining a 3D reconstruction

to produce a jointly optimal 3D structure and viewing estimates (i.e. camera pose). Jointly optimal means that the solution should be optimal with respect to camera and structure variations. The name *bundle adjustment* refers to the fact that both the 3D structure and the camera pose is optimized jointly in one bundle. In essence it is an optimization problem with the parameters being the combined 3D coordinates of the features, the pose of the cameras and, if needed, the calibration parameters. As with any optimization problem it is necessary to define a good cost function $f(x)$ to be minimized. One way to define the optimization problem is using a graph structure wherein the vertices are the objects that form part of the bundle (i.e. camera poses and 3D coordinates of features) with the observations of features forming the edges between the vertices. Each edge then encodes an error based on the observations and the 3D positions which can be minimized by applying an optimization algorithm [27].

2.8 Graph optimization

SLAM is a problem that can be defined in mathematical terms as a least square minimization problem. The inherited sensor biases creates a distance error over time that result in that a long term mapping sequence will have a mapping error according to these sensor biases. Graph based SLAM approaches builds on this definition and its purpose is to solve a non-linear least square problem. The goal of a graph based approach is to minimize a cost function containing a non-linear error function. Which can also be described in as maximizing the explanation of the set of measurements affected by gaussian noise that are added into the graph [16]. A graph can be described with nodes and edges. A node can for example contain a state vector of the current pose or the landmark states. Edges connect the state transitions and its observed landmarks and represent the spatial constraints between them. The general least square optimizing methods assume the state parameters to be in the euclidean space, while for the SLAM problem that is usually not the case. The SLAM space is represented more frequently in a non-euclidean form as a translational and rotational space. This creates a necessary non-linear operator to transform the parameters through the different spaces. As the goal is to minimize a least square cost function, the problem can be written as [16]:

$$F(x) = \sum_{ij} e(x_i, x_j, z_{ij})^T \Omega_{ij} e(x_i, x_j, z_{ij}) \tag{2.11}$$

$$x^* = \underset{\mathbf{x}}{argmin}\, F(x) \tag{2.12}$$

x is the parameter vector containing parameters through $x = (x_1, ..., x_n)$, the generic parameter block is written as z_{ij}, and Ω_{ij} contains the information matrix and the mean of a constraint for the parameter x_i and x_j. $e(\cdot)$ is the error function that represents the performance of how well the parameter block satisfy the constraints on z_{ij}. With a solid initial guess at x_0 the use of general numerical solving methods can be implemented such as Gauss-Newton or Levenberg-Marquart solvers. These methods use the idea of approximating the error function by its first order Taylor expansion and then iterating for every step, starting at x_0. The detailed derivation

is well described in [16] and results in a quadratic representation of the cost function in Equation (2.11):

$$F(x) = c + 2b^T \Delta x + \Delta x^T H \Delta x \tag{2.13}$$

which then can be minimized by solving the linear equation:

$$H\Delta x^* = -b \tag{2.14}$$

where H is the information matrix of the system. By adding Δx^{x*} incrementally to the initial guess a solution can be found.

This graph based approach to ORB-SLAM makes the algorithm highly effective. For ORB-SLAM the graph is built by including the camera pose and the map points in the nodes. All nodes are then connected with edges containing the spatial constraints between the current camera pose and map points, as well as the camera pose transitions.

3
Methods

3.1 Data collection

We gathered our own data to test the performance of the developed system and we describe how the data was gathered below.

3.1.1 Data collection set up

We used the ZED-camera developed by Stereolabs to gather stereo and monocular data [25]. The calibration parameters of the camera was provided by Stereolabs and was used to rectify the stereo images to align the epipolar lines horizontally. To gather the data used in the evaluation of the data set we used a Volvo XC-90, codenamed Snowfox, housed at Revere. Snowfox is equipped with a multitude of sensors, however we only used the ZED-camera as well as GNSS information gathered using the Ellipse-2N INS/GNSS unit from SBG Systems [26] to evaluate the accuracy of the localization and compare it with the ground truth. The data itself was gathered at, and around, Lindholmen in Gothenburg. The data was gathered at 20 FPS and we carried out two runs with the camera mounted on the hood for the first collection and on the roof for the second collection. A map of the ground truth trajectory superimposed on a roadmap along with the mounting positions of the camera can be found in appendix.

The KITTI dataset was used to benchmark the performance of the developed system against other SLAM-techniques. The gathering of our own dataset enabled us to evaluate the performance of the algorithm in driving scenarios in Gothenburg as well as to evaluate and compare the performance of the algorithm with GNSS data. Furthermore, we also gathered data within the scope of Chalmers formula student driverless to evaluate the mapping capabilities of the algorithm by localizing cones detected by our VLP-16 lidar [17] in the ORB-SLAM map to determine the accuracy of the object mapping.

3.1.2 Dataset evaluation

We evaluate the performance of our developed solution on both the established KITTI dataset and our own gathered dataset. In our evaluation based on KITTI, we use the provided evaluation program to generate the statistics used to score

methods on the dataset. The stereo trajectory is generated as in the original ORB-SLAM2 by printing the camera poses for each frame. Since the monocular version is unable to provide a mapping to metric coordinates due to being unable to resolve the scale we rescale the generated trajectory to enable a valid comparison with the data. With our own dataset we compare our trajectory with the gathered IMU data by providing ORB-SLAM2 with a reference geodetic coordinate so the trajectories can be compared using geodetic coordinates. We again scale the monocular data to provide a valid comparison.

3.1.2.1 CFSD evaluation

To evaluate the performance of ORB-SLAM2 within the context of Chalmers Formula Student Driverless we used image data gathered at 5 FPS to generate a map of the environment. From the lidar we extracted the cone position by clustering and object classification. In the ORB-SLAM2 map we manually tagged the cone position to compare the two positions. We also looked at the accuracy of the localization using the IMU and compared that data as we did with our gathered dataset.

3.2 Software framework

As a software framework we used the OpenDLV framework developed at Revere. This further enabled the use of our work with the vehicles at Revere as well as being able to use our solution as part of a larger system on a vehicle. OpenDLV is based on the Libcluon library and provides communication between microservices. Microservices are contained within separate Docker images. Each image contains the lightweight operative system Alpine as well as needed library dependencies and the microservice binary.

3.3 Software design

The main work effort required for this thesis is the implementation of ORB-SLAM2 in the OpenDLV framework with performance that is good enough to be used in real-time. This entailed creating one OpenDLV microservice which runs ORB-SLAM2 taking input from a camera and creating and updating a map and perform localization within the map. An accompanying *graphical user interface* (GUI) was also developed so that the results can be visualized.

The code was divided into three segments: tracking, local mapping and, loop closing as described in the original paper. This was implemented as three separate threads that share data between them. Furthermore we created classes for the custom datastructures needed such as map point, Frame, and KeyFrame. In addition, we defined several helper classes to deal with conversion, optimization, initialization, and ORB extraction and comparison. Furthermore some classes and data structures from the DBoW2 library was also implemented as they are needed in ORB-SLAM2 but to avoid introducing implementation specific dependencies we refactored the

library to OpenDLV ourselves. Further dependencies used were OpenCV, Eigen, and g2o which is an open source graph optimization library which was used for the *loop closing*. There was also a master class that initializes these threads and reads configuration settings. We present the structure of ORB-SLAM2 as laid out in the original paper below.

3.3.1 Datastructures

ORB-SLAM2 consists primarily of two datastructures which is used to build the map. These are the map points corresponding to landmarks and keyframes which correspond to the camera poses. In addition a frame is defined to store the properties of an image and its ORB features.

3.3.1.1 Map point

A point in the map is essentially a keypoint given by an ORB-feature spotted in several frames. It stores its 3D position in the world coordinates X_w and its mean viewing direction n_i. Furthermore it stores the associated ORB descriptor which is defined as the descriptor that has the minimum Hamming distance to all the associated descriptors in the keyframes it is observed. It also defines a maximum and minimum viewing distance according to the ORB features and their scale invariance. The map points are used for localization by using the known 3D location of the map points to refine the pose given by the constant motion model [20].

3.3.1.2 Frame

A frame stores all the properties that can be extracted from any one image, or image pair in the case of stereovision. This includes the extracted ORB features and associated descriptors and bag of words representations. It also stores the visible map points as well as pose matrices for that particular frame [20].

3.3.1.3 Keyframe

A keyframe is essentially a special case of a frame. It has an associated frame and therefore also stores the visible keypoints and descriptors given by the ORB features, regardless of whether or not they are map points. The keyframe also stores the camera pose T_{cw} which is a transformation from the world frame to camera coordinates. The keyframe is also linked to other keyframes with a covisibility graph with keyframes as nodes and the number of covisible map points contained in the weighted edge of the graph. The covisibility graph is used to define a local neighborhood around a keyframe by defining it as the keyframes linked to that keyframe in the covisibility graph and this is then used when tracking the local map. The covisibility graph is also used during the loop closing procedure to distribute the loop closing error along the graph. Instead of using the entire covisibility graph a subset called the essential graph which contains all the nodes but less edges which reduces the computational cost. A keyframe is only linked to another in the essential graph if they have more than 100 covisible map points or if it is included in the

spanning tree. The spanning tree is a different subset of the essential graph which is created by linking each new keyframe to the keyframe it shares the most covisible points with. By constructing the essential graph like this and also including the loop closure edges the essential graph can be used in the loop closing procedure to propagate the error in the loop [20].

3.3.2 Localization

This is the primary thread dealing with tracking the pose of the camera in relation to the local map. In accordance with this it also deals with processing the images to compute new poses. This thread also has the responsibility to make the decision to create a new keyframe based on the quality of the tracking.

Initialization As the initialization procedure is very different for stereo and monocular cameras they will both be described separately.

Monocular For a monocular camera a good initialization procedure is key to be able to project 2D points into 3D space and create an initial set of map points to be used for further tracking. This is achieved by triangulating an initial set of points in the map using the relative pose between two frames. In ORB-SLAM2 the initialization procedure computes two different geometrical models in parallel, one using a homography matrix defined as

$$X_c = H_{cr} X_r \tag{3.1}$$

dealing with planar scenarios and one using a fundamental matrix defined as

$$X_c^T F_{cr} X_r = 0 \tag{3.2}$$

dealing with non-planar scenarios. In more detail the procedure is: First extract the ORB features of the current frame I_c and search for matches in the reference frame I_r so that the correspondences X_c and X_r can be found. If there is not enough correspondences the reference frame is reset. If enough correspondences are found they are used to compute the F_{cr} and H_{cr} matrices in parallel. This is done using a RANSAC scheme. To select the best candidate a score is defined as

$$S = \sum_i \rho(d_{rc}^2(X_c^i, X_r^i, M)) + \rho(d_{cr}^2(X_c^i, X_r^i, M)) \tag{3.3}$$

$$\rho(d^2) = \begin{cases} \Gamma - d^2, & \text{if } d^2 < T \\ 0, & \text{if } d^2 > T \end{cases} \tag{3.4}$$

Where M is the model used (i.e. F or H) and d is the symmetric transfer error between the frames. The threshold T is defined differently for the two models ($T_H = 5.99$ and $T_F = 3.84$). Γ is defined as equal to T_H to make sure that both models score equally for the same distance. After the scores have been computed the best matrix is returned as well as the respective scores for these matrices. Now a model has to be selected based on their scores since while both the fundamental and

homography model can produce a result in either case the homography will produce a more accurate result in the planar case whereas the fundamental matrix will be more accurate in non-planar cases. So a heuristic is defined as

$$R_H = \frac{S_H}{S_F + S_H} \tag{3.5}$$

and the homography is selected if $R_H \geq 0.45$ and the fundamental matrix is selected otherwise [20].

Once a model has been selected it is evaluated by constructing motion hypotheses and evaluating them to see which one has the most points seen in the frame. If there is no clear winner the solution isn't accepted as that means the model has some ambiguity and the solution will be corrupt. In that case the solution is discarded and the initialization is repeated. If a clear solution is found a full bundle adjustment is performed to refine the initial reconstruction.

Stereo In the stereo case the initialization procedure is much simpler since we do not depend on an initialization procedure to compute the 3D coordinates of the initial map points since this information can be directly calculated using the depth information. In this case an initial keyframe is simply created along with the identified map points in the keyframe and the tracking can proceed from there [19].

Pose estimation For each new image, a frame is created and $n = 2000$ ORB features are subsequently extracted at eight scale levels with a scale factor of 1.2. In order to ensure an even distribution of the extracted features the image is divided into a grid and at least five features are extracted from each grid square. If not enough features can be found the threshold is adapted to ensure enough features are extracted. The number of features extracted from each cell also change if some cells are found to contain no or very few features. After the FAST corners have been detected the ORB-descriptor of the feature is also calculated. When the features have been extracted, a constant velocity model is used to predict the current camera pose. Then a search is performed to find matches between the ORB features in the frame and the already existing keypoints in the last frame. The pose is then optimized using the found map points by formulating it as a graph problem and optimizing said graph [20].

Local map tracking After the pose has been refined and we also have an initial set of correspondences to the last frame we can project the local map into the frame to search for map points. The local map contains the set of keyframes that share the map points seen by the current frame as well as their neighbours in the defined covisibility graph. After the local map has been projected into the current frame each map point in the local map is searched in the current frame. This is done according to the following process.

- The map point projection in the current frame is computed. If it lays out of the bounds of the image it is discarded.

- The angle between the mean viewing direction and the current viewing ray is computed. If it is larger than 60^o it is discarded.
- The distance from the camera pose and the map point is computed. If it is outside of the scale invariance region of the map point it is discarded.
- Compute the scale of the map point.
- Compare the descriptor of the map point with the still unmatched ORB features in the frame near the map point projection, associate the map point to the nearest match.

After this the pose is optimized again with all the map points found in the frame.

New keyframes If the tracking is successful it is decided whether or not the current frame should become a new keyframe. This is done according to a specific policy that should be fairly lenient. In ORB-SLAM2 a new keyframe is created if the following criteria are met

- If more than a second has passed since the last relocalization
- And if more than a second has passed since the last keyframe
- Or If local mapping is idle
- Or if the stereo tracking is tracking few points or many close points that aren't tracked by keyframes
- And If current frame tracks less than 90% (mono) or 75% (stereo) of the map points in the reference keyframe

The last condition ensures that the keyframes are visually different and the first condition ensures a good relocalization. The fourth condition is only valid for the stereo case and is used when a big part of the frame is far away from the camera. In such a condition it is necessary to have a sufficient amount of close points to ensure a good translation.

If a new keyframe is to be created the final step is to create it from the current frame and insert it into the mapping. In the stereo case the close keypoints are inserted in the map right away since the 3D positions of these points can be accurately determined whereas for the far stereo and monocular case the map points need to be triangulated first which is done in the mapping thread. The keyframe is then pushed into a queue of new keyframes to be dealt with by the mapping thread [20].

3.3.3 Mapping

After a new keyframe has been created in the tracking thread it is the responsibility of the mapping to incorporate it into the map. This means updating the covisibility graph and creating and triangulating new map points and culling redundant ones. In addition, a local bundle adjustment is performed to refine the local map. Redundant keyframes are also culled in the mapping thread.

New keyframes A new keyframe is first associated with the map points found in the keyframe by computing an updated descriptor and mean viewing direction. Then the keyframe is added to the covisibility graph by using the associated map

points to find covisible keyframes. In addition a bag of words representation is computed.

Map point culling In order to ensure a stable map with few outliers and robust map points the points recently created must pass a few criteria. First they must be seen in at least 25 % of the frames it is predicted to be visible and if more than one keyframe has been inserted since the map point was created it needs to be seen by at least three keyframes. If it passes these tests it can only be removed if it is observed by less than three keyframes at a time. This can happen if keyframes are removed or if the local bundle adjustment classifies an observation as an outlier.

New map points When a new keyframe is added to the map unmatched ORB-feature points in that keyframe are matched with other unmatched points in the neighbouring keyframes in the covisibility graph. If a match is found it is triangulated with respect to the two keyframe poses in order to compute a 3D position in world coordinates. As a check for consistency the map point should have positive depth in both frames and a similar scale and reprojection error. If a point is accurately triangulated it is finally added as a map point in both of the frames that it was triangulated in, as well as to the map. After all the unmatched points have been checked the added map points are projected into other neighboring keyframes to check for matches in those frames as well.

Local bundle adjustment To refine the triangulation and the local map a local bundle adjustment is performed using the newly added keyframe, the neighboring keyframes in the covisibility graphs as well as all their map points. Neighbors of neighbors (i.e. the keyframes that see map points also seen by the neighboring keyframes in the covisibility graph) are also included but are kept fixed during the optimization. During the course of the optimization, a map point observation can be found to be an outlier and is then discarded. This is done in the middle and at the end of the optimization run.

Keyframe culling After the local bundle adjustment has been performed, the local keyframes are checked for redundancy. A keyframe is considered to be redundant if more than 90 % of the map points it sees are seen by other keyframes in the same or finer scale. The advantage of culling the keyframe is that the number of keyframes is then bounded if operating in the same environment. In addition, the complexity of the bundle adjustment also grows with the number of keyframes so keeping the number of keyframes manageable is vital to the performance of the system. As part of the culling procedure the covisibility graph is also updated along with the spanning tree [20].

Loop closing

After a keyframe has been processed by the mapping thread it is pushed to the loop closing thread. The loop closing thread processes the keyframe in parallell with the rest of the system to detect and close loops. Loop closing is essential to the system

as it provides a check on the error accumulated in the loop which can be significant especially in the monocular case where scale drift is a significant source of error.

Loop detection For the place detection, the bag of words similarity score is used to detect loop candidates. The current keyframe is compared to its neighbors in the covisibility graph and the minimum score s_{min} is kept. Then the entire database of keyframes is queried and if a keyframe is not connected with the current keyframe and it has a score lower than s_{min} it is kept as a loop candidate. If it is validated as a loop candidate three times consecutively it is accepted and is then evaluated more closely.

Similarity transform The next step is to compute a Sim(3) similarity transform between the current keyframe and the loop candidate. The result of this transformation will also provide the error accumulated in the loop. In monocular SLAM there are seven degrees of freedom, three translational, three rotational, and one scale dimension. To compute the similarity transform the map points seen by the loop candidate(s) are matched with the map points seen in the current keyframe. This matching is done using the bag of words representation which means that the 3D positions of the respective points can be used to establish a 3D-to-3D correspondence between the current keyframe and the loop candidate. Using these map point matches RANSAC iterations as in [14] are done to compute the similarity transformation. If enough inliers are found the similarity transform is used to perform a search for more matches, this time using the similarity transform and 3D locations to aid the search. After this, the similarity transform is further optimized by minimizing the reprojection error in both keyframes and if successful another matching is performed and if enough matches are found the loop is accepted.

Loop correction After the loop and similarity transform has been computed duplicated map points are fused and new edges are added to the covisibility graph to represent the loop closure. In more detail, the pose of the current keyframe is corrected using the similarity transform and this correction is propagated to all the neighbors in the covisibility graph. The map points are also corrected by projecting all the map points seen by the loop keyframe and its neighbors into the current keyframe and searching for matches and fusing those points. This creates new edges in the covisibility graph which closes the loop in it.

Essential graph optimization As a way to distribute the accumulated error a final optimization is performed over the essential graph which distributes the error over the loop. As part of the correction the map points are corrected in accordance with the correction of the keyframe that observed the map point [20].

Global bundle adjustment As a means to ensure that the optimal loop closure has been performed, a full bundle adjustment of the entire map is performed after each loop closure. Since this is a very computationally costly operation it is performed in a separate thread to allow the system to continue mapping new keyframes. There is also the added challenge of merging the output of this optimization with

the new state of the map. If a new loop closure is detected, the global bundle adjustment is immediately aborted and the loop closure procedure starts again. After the global bundle adjustment finishes the map is updated by propagating the updated poses of the keyframes through the covisibility graph. Non-updated map points are updated according to the correction applied to their reference keyframe [19].

3.3.4 Graph optimization

As part of ORB-SLAM2 there is a lot of bundle adjustment methods and pose optimization which requires a structured way to handle the optimization problem. ORB-SLAM2 uses the library g2o which is a graph optimization library. All optimization problems are therefore defined as graph problems and then solved using the Levenberg-Marquardt algorithm implemented in g2o. A more detailed problem formulation for each optimization problem is presented below.

Bundle adjustment Three types of bundle adjustment are used across ORB-SLAM2. The first one is pose optimization which only includes the current frame and the currently visible map point observations as edges. This is done to optimize the pose provided by the constant motion model and reject outlier observations. The second type is local bundle adjustment which is done in the mapping thread to refine the local map by including all keyframes in the local neighborhood of the current keyframe as well as all map points seen by all those keyframes. The third version is the global bundle adjustment which takes the entire map and optimizes it as a bundle. This means linking all map points with every single keyframe to ensure a good and stable map. This is only done after loop closure. Regardless, the mathematical definition of the problem is the same and the only difference is the size of the graph to be optimized. The problem is defined as optimizing a set of poses defined by the camera orientation matrix $\mathbf{R} \in \mathrm{SO}(3)$ and the translation vector $\mathbf{t} \in \mathbb{R}^3$ as well as a set of matched 3D map points $\mathbf{X}_{w,j} \in \mathbb{R}^3$. The error to be minimized is the reprojection error between the map point $\mathbf{X}_w$ and keypoint $\mathbf{p}_m \in \mathbb{R}^2$ in the monocular case or $\mathbf{p}_s \in \mathbb{R}^3$ in the stereo case. The error term for an edge linking a map point j with a keyframe i is then defined as

$$\mathbf{e}_{i,j} = \mathbf{p}_j - \pi(\mathbf{R_i}\mathbf{X}_j + \mathbf{t}_i) \tag{3.6}$$

where π is the projection function defined as

$$\pi_m([xyz]^T) = \begin{pmatrix} f_x \frac{x}{z} + c_x \\ f_y \frac{y}{z} + c_y \end{pmatrix} \tag{3.7}$$

for the monocular case, and

$$\pi_s([xyz]^T) = \begin{pmatrix} f_x \frac{x}{z} + c_x \\ f_y \frac{y}{z} + c_y \\ f_x \frac{x-b}{z} + c_x \end{pmatrix} \tag{3.8}$$

for the stereo case. f_x and f_y is the focal length in x and y direction respectively and c_x and c_y is the principal point, and b is the baseline. The objective function to

be minimized is then defined as

$$f = \sum_{i,j} \rho_h(\mathbf{e}_{i,j}^T \boldsymbol{\Omega}_{\mathbf{i,j}}{}^{-1} \mathbf{e}_{i,j}) \tag{3.9}$$

where ρ_h is the Huber cost function and $\boldsymbol{\Omega}_{\mathbf{i,j}}$ is the covariance matrix given by the scale at which keypoint $\mathbf{p}_j$ was detected in keyframe i.

Sim3 optimization This optimization is used to refine the similarity transformation $\mathbf{S}_{1,2}$ between keyframe 1 and 2 used to compute the loop error. Given a set of map point matches we want to minimize the reprojection error between a map point i in frame 1 $\mathbf{X}_{1,i}$ and a corresponding keypoint j in frame 2 $\mathbf{p}_{2,j}$ and vice versa by optimizing $\mathbf{S}_{1,2}$ to obtain the optimal similarity transform. In more detail the error of each edge is defined as

$$\mathbf{e_1} = \mathbf{p_{1,i}} - \pi_1(\mathbf{S_{1,2}}, \mathbf{X_{2,j}}) \tag{3.10}$$

$$\mathbf{e_2} = \mathbf{p_{2,j}} - \pi_1(\mathbf{S_{1,2}}{}^{-1}, \mathbf{X_{1,i}}) \tag{3.11}$$

. Here the projection function is defined similarly as for bundle adjustment, with the difference that the map point is transformed using the similarity transform instead of the camera pose. The cost function to be minimized is

$$f = \sum_{i,j} \rho_h(\mathbf{e_1}^T \boldsymbol{\Omega}_{\mathbf{1,i}}{}^{-1} \mathbf{e}_1) + \rho_h(\mathbf{e_2}^T \boldsymbol{\Omega}_{\mathbf{2,j}}{}^{-1} \mathbf{e}_2) \tag{3.12}$$

where $\Omega_{1,i}$ again refers to the covariance matrix of keypoint $\mathbf{p}_i$ in frame 1 and 2 respectively. In this optimization the map points are fixed and we only optimize $\mathbf{S}_{1,2}$ to obtain an accurate similarity transform.

Essential graph optimization This optimization is used to optimize the essential graph so that the loop error is distributed over the loop. The optimization done is using a Sim(3) transformation of each keyframe to account for scale variance. However, the keyframe is converted using the camera poses in SE(3) and using the scale factor $s = 1$. In this optimization only the keyframes are used and the loop closure keyframe is kept fixed. The error of each edge is defined as

$$\mathbf{e}_{i,j} = log_{Sim(3)}(\mathbf{S_{i,j}} \mathbf{S_i} \mathbf{S_j}{}^{-1}) \tag{3.13}$$

where $\mathbf{S_i}$ is the similarity transform of keyframe i and $\mathbf{S_{i,j}}$ is the relative pose constraint between two keyframes. The cost function to be minimized is then defined as

$$f = \sum_{i,j} \mathbf{e}_{i,j}^T \boldsymbol{\Lambda} \mathbf{i}, \mathbf{j} \mathbf{e}_{i,j} \tag{3.14}$$

where $\Lambda_{i,j}$ is the information matrix of the edge which in this case is simply the identity matrix. After the optimization each similarity transform is translated back to the camera using the relation $s\mathbf{t} = \mathbf{t}_{corr}$ leaving the rotation unchanged and changing the translation vector with the optimized scale factor thereby correcting for scale drift accumulated in the loop [20][19].

4

Results

We present results both from the popular KITTI dataset to compare our dockerized implementation compares with the original ORB-SLAM2. We also present results from our own dataset gathered at Lindholmen as well as data gathered from testing with the Chalmers formula student driverless project. Further we investigate the framerates we acquire and the dependency on image resolution for the performance of the algorithm.

4.1 KITTI evaluation

Below we present the results of our dockerized version of ORB-SLAM2 on the KITTI dataset.

4.1.1 Stereo

The trajectory for the sequence named '00' is presented in Figure 4.1. This sequence is gathered at speeds from 15–55 km/h in an urban environment and it contains loop closures at three places in the sequence. Figure 4.3 shows the average of the rotation and translation error over subsequences of different length. The relative average translation and rotation error decreases with distance in this sequence. In addition high speed seems to cause a higher error as seen in Figure 4.2.

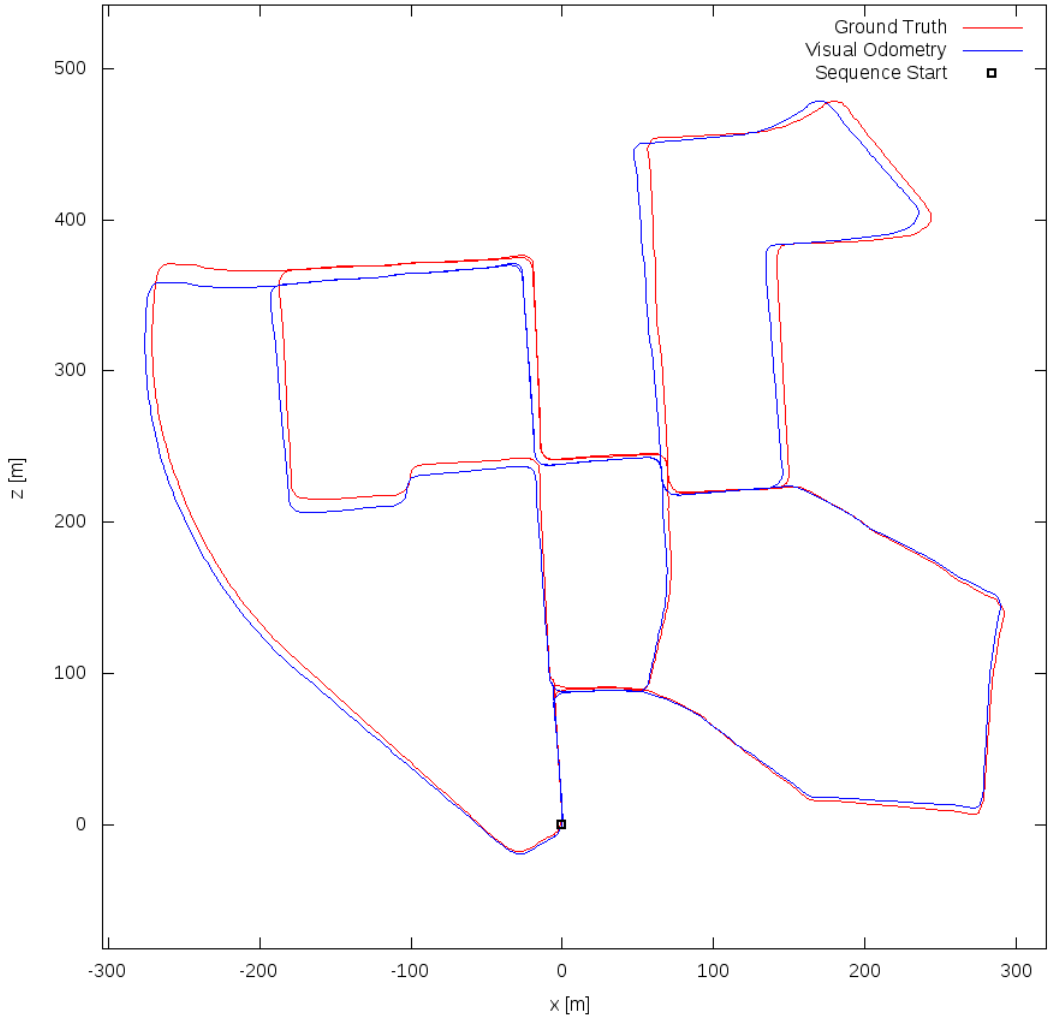


Figure 4.1: Estimated path from ORB2 versus the ground truth for KITTI sequence 00.

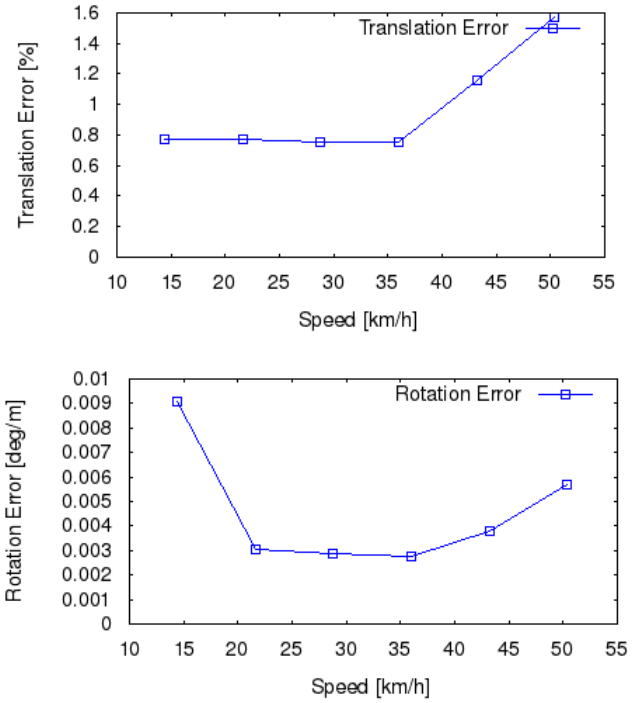


Figure 4.2: Translation and rotation error versus speed for KITTI sequence 00.

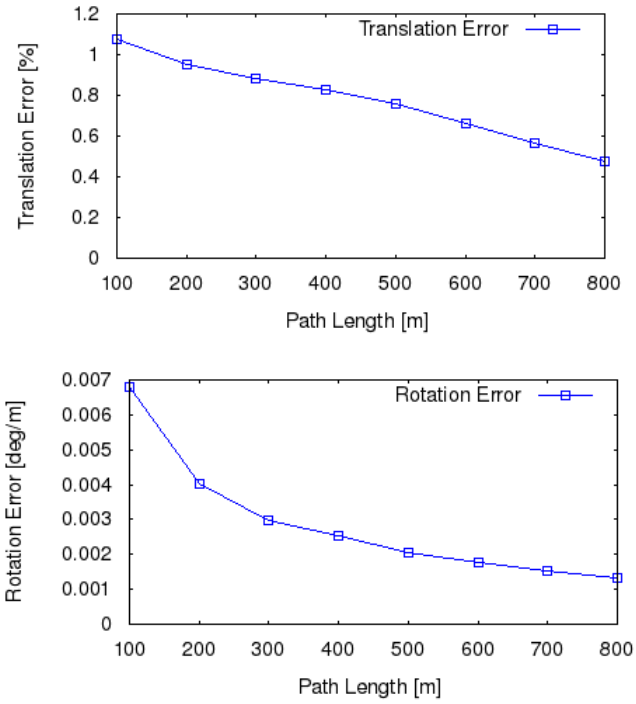


Figure 4.3: Translation and rotation error versus distance traveled for KITTI sequence 00.

The sequence named '01' of the KITTI dataset is gathered on a highway and the speeds range from 40–90 km/h. The trajectory along with ground truth is presented in Figure 4.4. It is a single stretch and as such do not contain any loops. In Figure 4.6 we can instead observe that the relative average translation error increase with distance. The rotation error however still decreases as the distance grows larger. In Figure 4.5 the speed curve is interesting in that it shows high error for the lower and higher range of the speed profile.

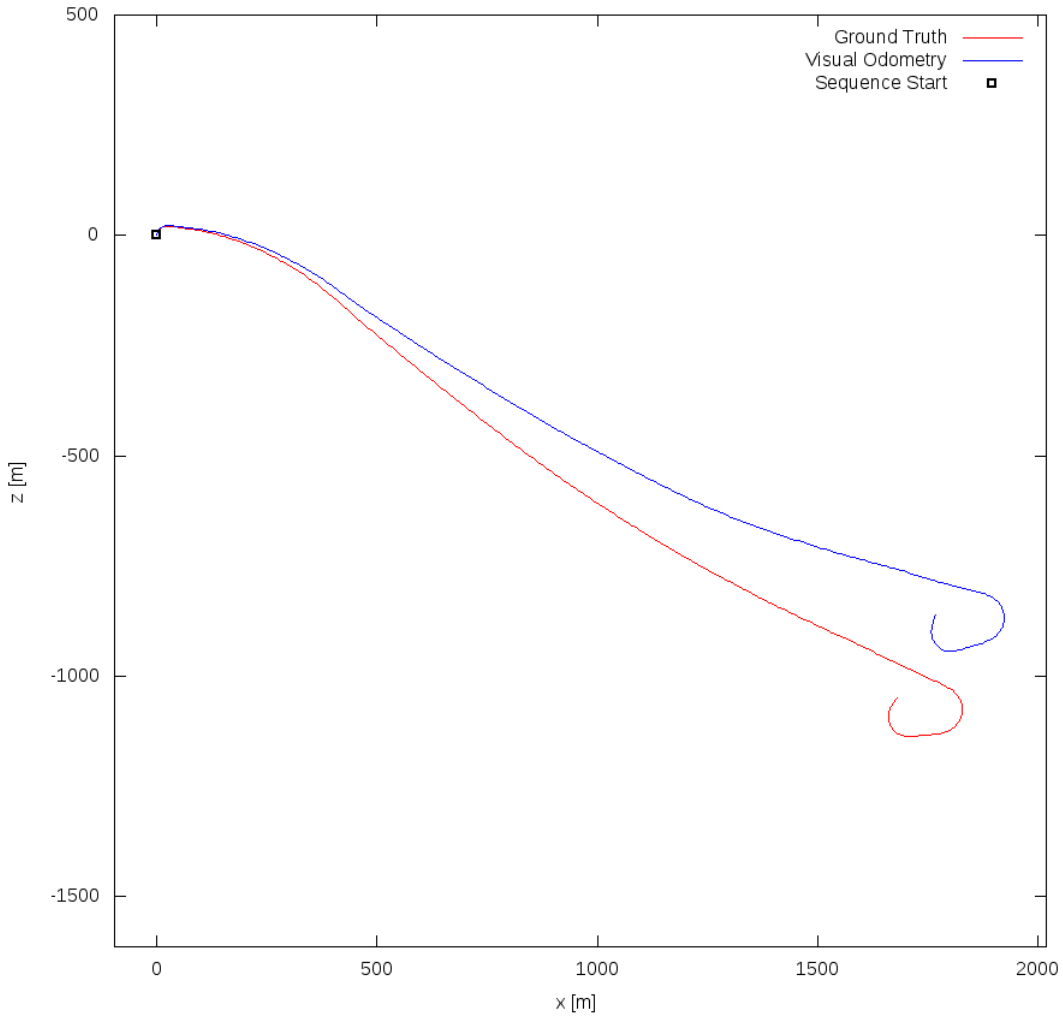


Figure 4.4: Estimated path from ORB2 versus the ground truth for KITTI sequence 01.

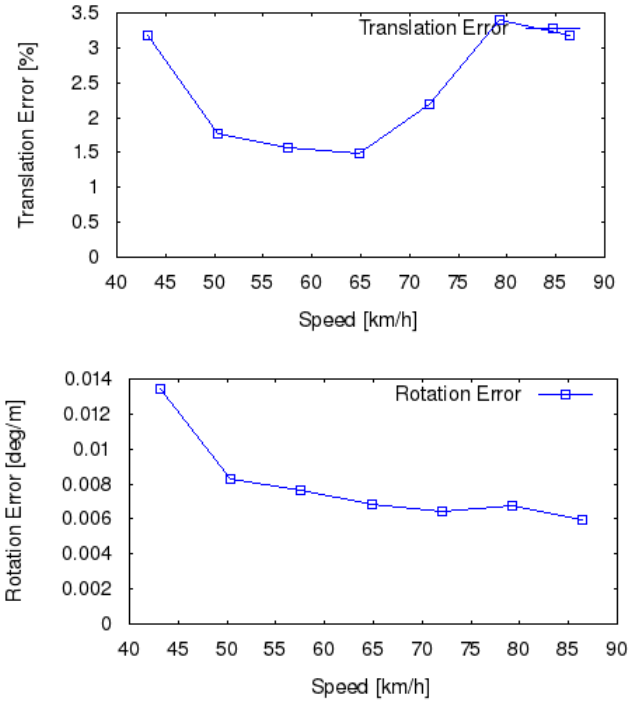


Figure 4.5: Translation and rotation error versus speed for KITTI sequence 01.

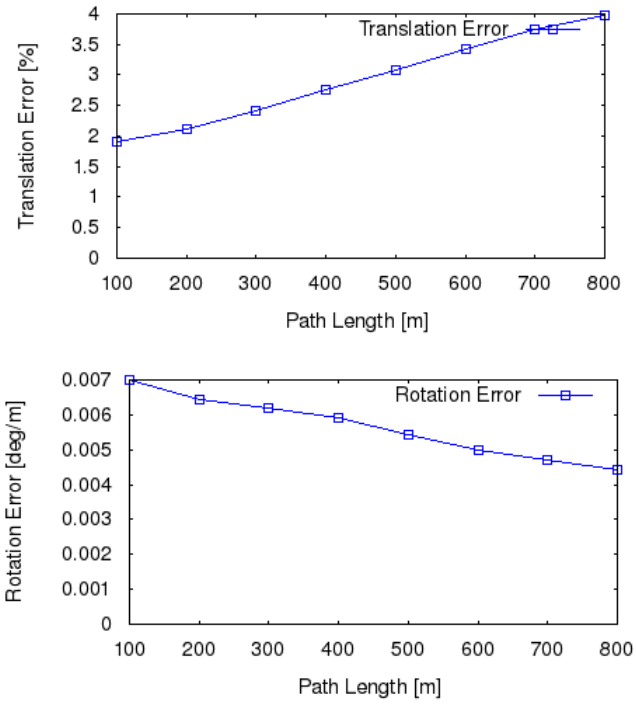


Figure 4.6: Translation and rotation error versus distance traveled for KITTI sequence 01.

The '05' sequence is primarily a suburban environment with small streets and houses close to the sides, it contains two loops. The trajectory along with the ground truth is plotted in Figure 4.7. In this sequence ORB-SLAM2 is able to map the environment very accurately with a very low translation and rotation error. As with sequence '00', in Figure 4.9 we can observe the relative translation error decreasing with distance. The low speeds seem to cause the highest error in this sequence as seen in Figure 4.8.

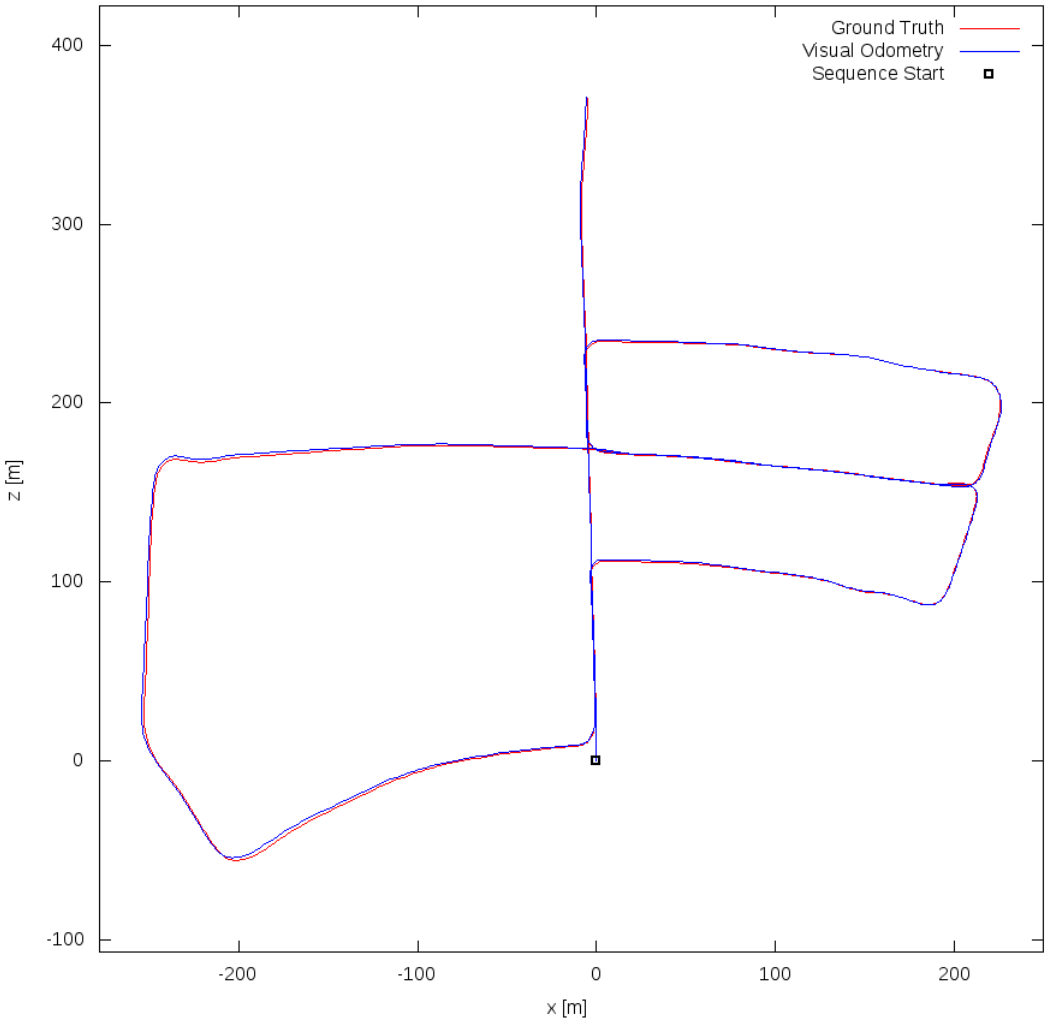


Figure 4.7: Estimated path from ORB2 versus the ground truth for KITTI sequence 05.

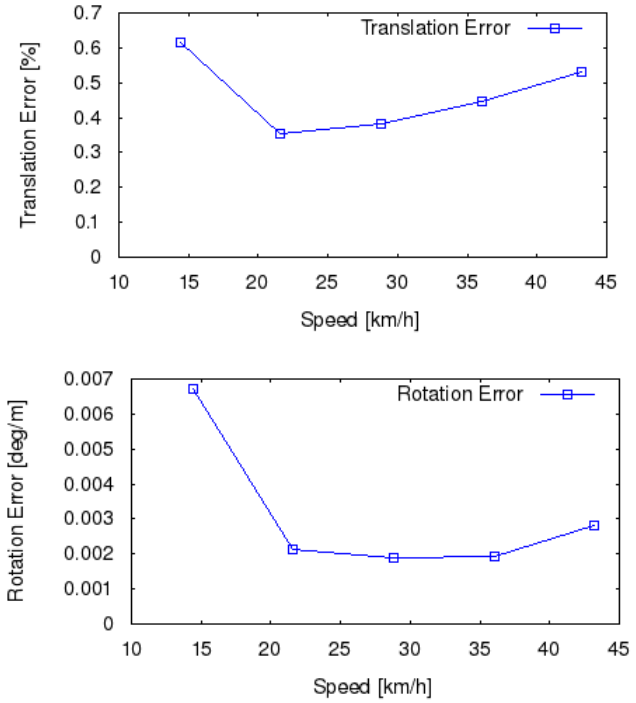


Figure 4.8: Translation and rotation error versus speed for KITTI sequence 05.

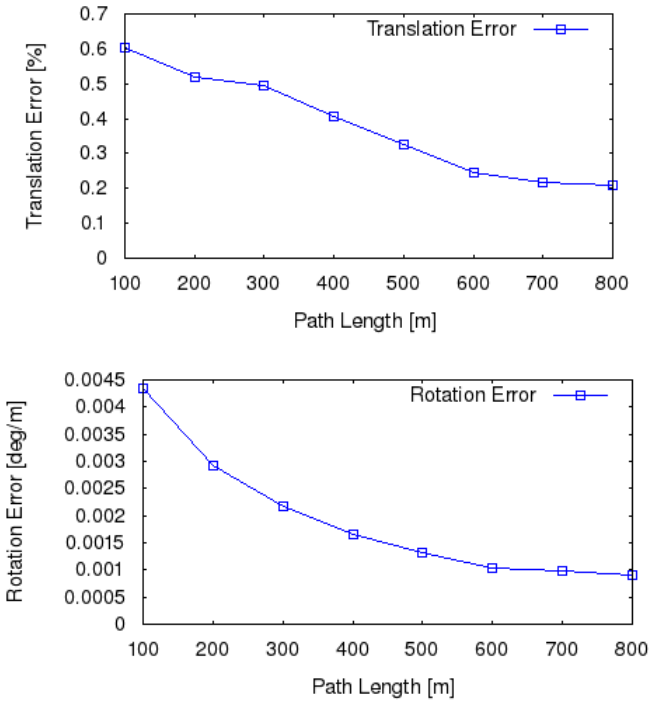


Figure 4.9: Translation and rotation error versus distance traveled for KITTI sequence 05.

KITTI sequence '09' is again a suburban environment with some forested parts and it contains one loop at the very end of the sequence. The trajectory is plotted in Figure 4.10. ORB-SLAM2 do not usually detect the loop due to it occurring so late and the loop closing procedure being so rigorous. The impact of this can be seen in the trajectory as the map drifts further and further as the distance increases which is supported by the relative error dependent on distance plotted in Figure 4.11. This underscores the importance of loop closure for the performance of the system even in the stereo case.

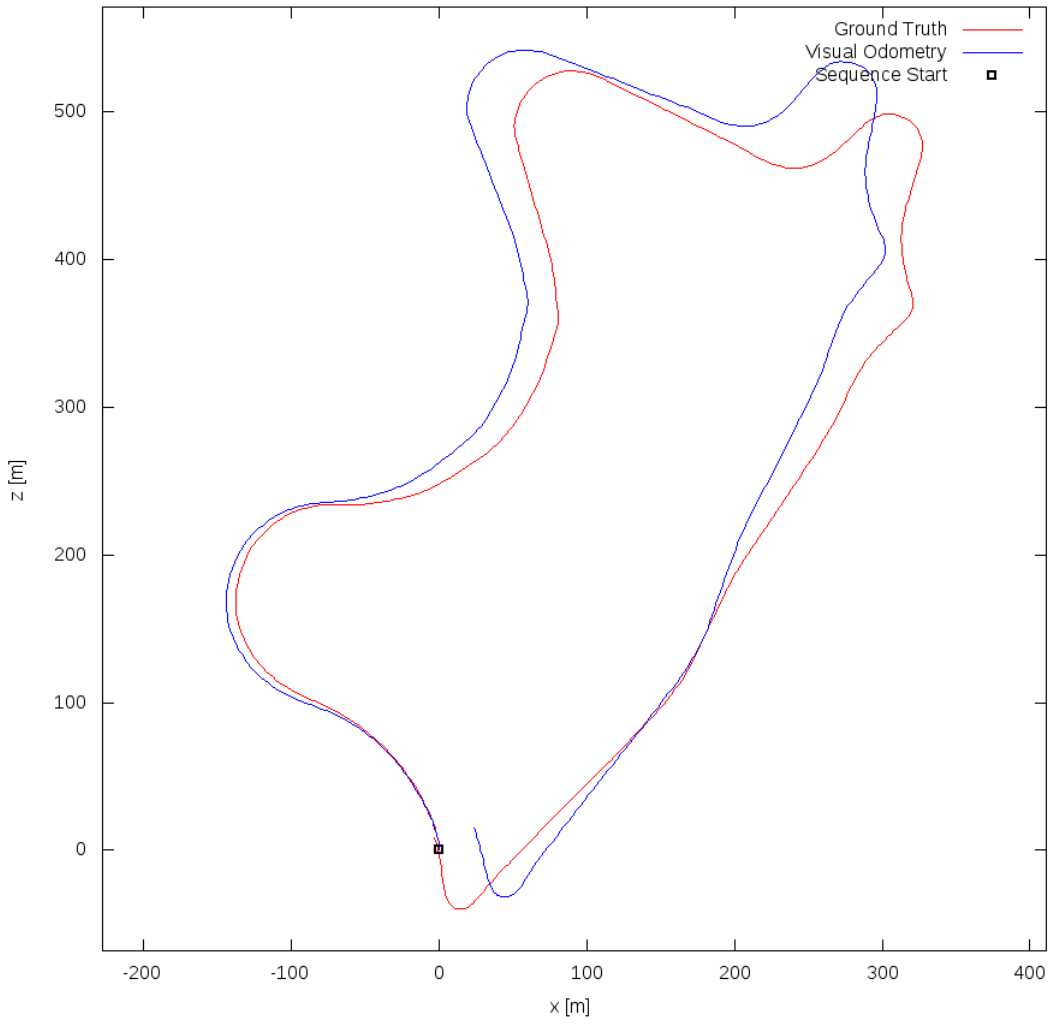


Figure 4.10: Estimated path from ORB2 versus the ground truth for KITTI sequence 09.

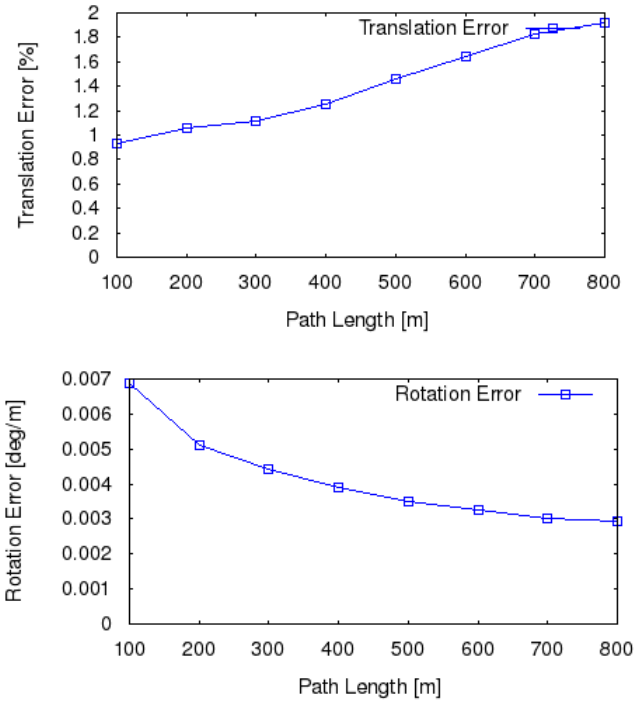


Figure 4.11: Translation and rotation error versus distance traveled for KITTI sequence 09.

4.2 Own dataset evaluation

Here we present the results on our own two gathered datasets.

4.2.1 Monocular

For the monocular case we were unable to map the entire first sequence. The system does initialize but then fails at the 180º turn and due to not revisiting that section is unable to relocalize itself. For the second sequence the system is able to map the entire sequence if the number of features is increased to 3000. The trajectory is plotted in Figure 4.12 along with the ground truth and rescaled with scale parameter s=32.3. Rotation is approximated quite well in this sequence but the motion estimation has a different scale in different parts of the map meaning that the translation error is still quite significant.

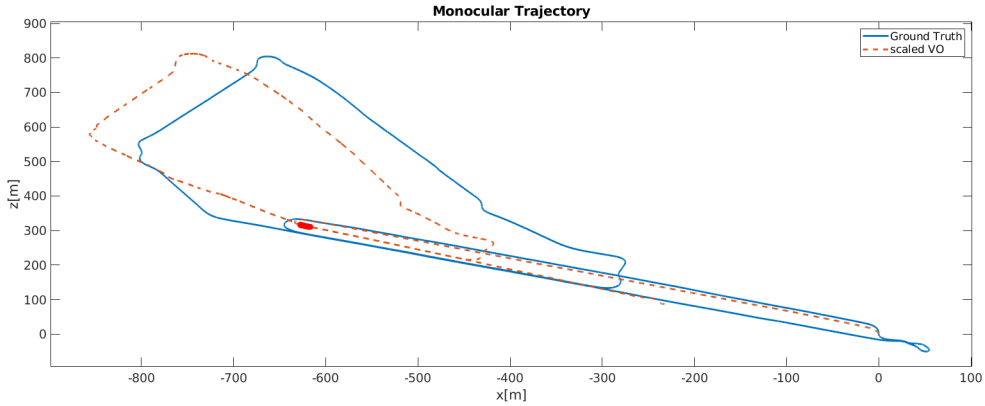


Figure 4.12: Rescaled trajectory (red, dashed line) compared with ground truth (blue, straight line). The rescaling factor is s=32.3 with the scale derived from the first long straight. The main error introduced in this sequence is the overestimation of the motion in the first straight. The scale error is quite large though meaning that the data needs to be rescaled before use.

4.2.2 Stereo

The stereo trajectory along with ground truth is plotted in Figure 4.13 and 4.14 for the first and second dataset respectively. Both are rescaled with scale factor s≈1.3 to compensate for the underestimation of motion which occurs. Both sets have problems with the rotation of the 180^o turn causing rotation errors disrupting the sequence. Due to the fact that the first sequence fails to close the loop the end error is less severe than the second sequence where the loop is closed. This causes the rotation error to propagate further by aligning the the trajectories of the loop. Something else of note is the relocalization occuring at the very end of the second sequence showing the capability of the system to recognize a previously mapped position and relocalize itself in it.

4.3 Performance analysis

We evaluate how the performance of the tracking part of the algorithm depends on image resolution and the number of ORB-features. The runs were all done on the stereo version of the dataset with the camera mounted on the hood. The tested cases and their results are outlined in Table 4.1 with the exception of the case with 1280x720 and 1000 features which failed to map the sequence. The trajectories along with the ground truth of the different cases are plotted in Figure 4.15. With 1280x720 the processing time is larger but the tracking is also slightly better meaning that there exists a tradeoff with regards to processing time and accuracy. Furthermore we can note that the resolution seems to have a larger impact on the processing time than the number of features. Also of note is that the processing time of the tracking part do not increase over time or as the map grows larger which can be seen in Figure 4.16.

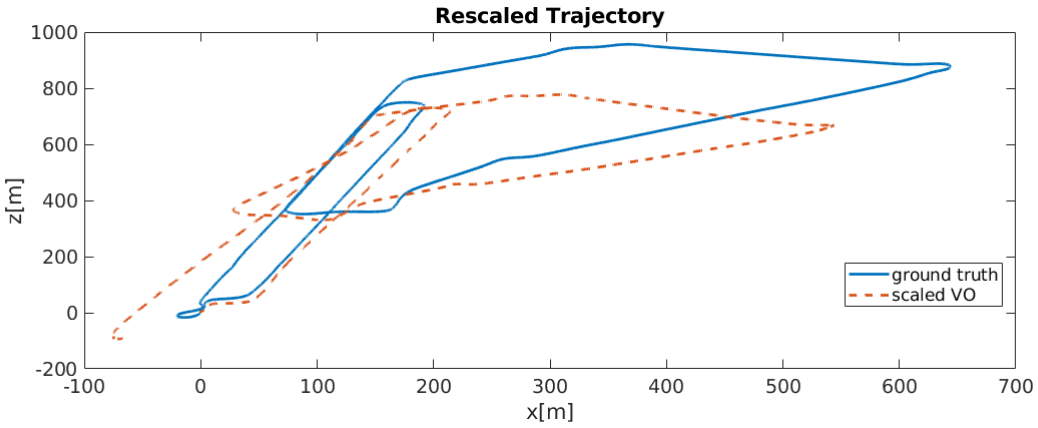


Figure 4.13: Rescaled trajectory (red, dashed line) compared with ground truth (blue, straight line). The rescaling factor is s=1.32 with the scale derived from the first long straight. Even with rescaling the trajectory shown is wrong due to rotation error in the 180^o turn as well as further underestimating the motion in the larger loop.

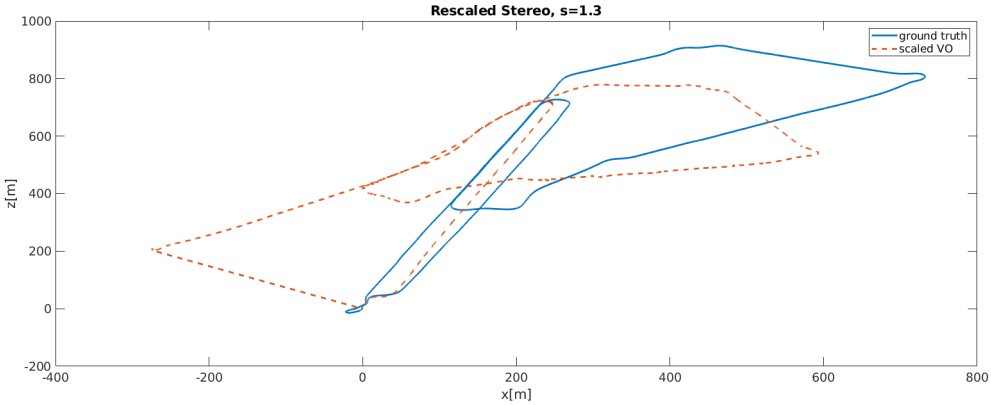


Figure 4.14: Rescaled trajectory (red, dashed line) compared with ground truth (blue, straight line). The rescaling factor is s=1.3 with the scale derived from the first long straight. In this set severe rotation error can be seen in the 180^o turn causing the rest of the trajectory to drift severely. The line at the end is a relocalization after losing tracking at the very end of the sequence. The larger loop is fairly well approximated due to two loop closures.

Case	Mean Time per Frame	Standard Deviation	Map size
1280x720, 2000 features	0.2197 s	0.0248	369718
640x360, 2000 features	0.1360 s	0.0250	205942
640x360, 1000 features	0.1200 s	0.0152	155940

Table 4.1: Comparison with processing time per frame for different image sizes and ORB features

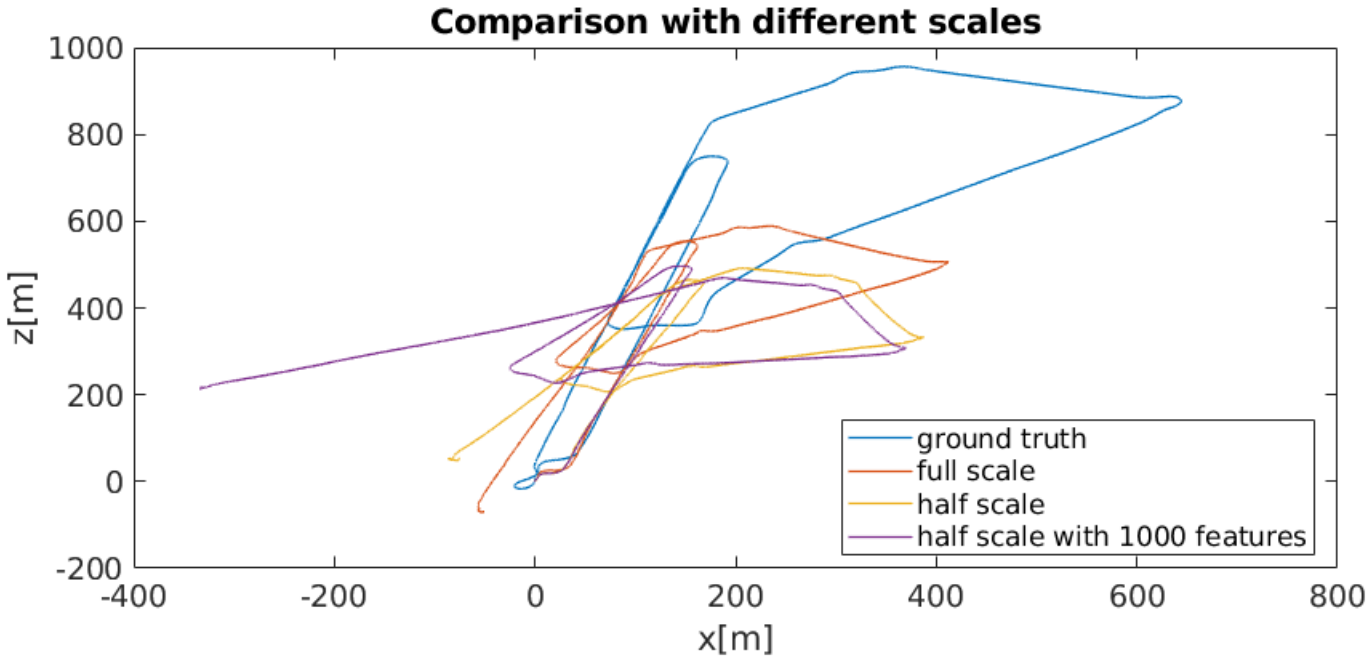


Figure 4.15: Trajectories with the different cases in Table 4.1 along with the ground truth. The 640x360 with 2000 features closes the loop and the others do not. What can be seen is that the gain in processing time comes at the cost of lower accuracy especially in the form of a greater scale error but also in the rotation error.

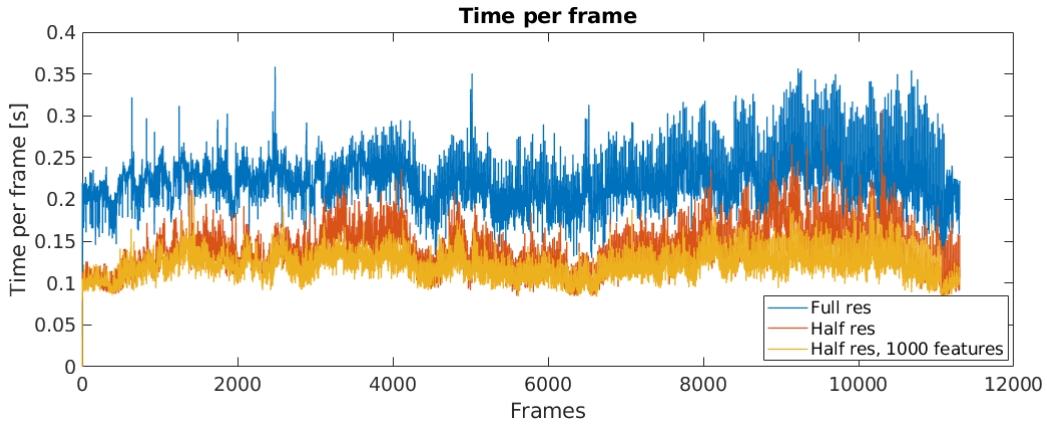


Figure 4.16: Comparison of the processing time per frame over the course of the run for the cases outlined in Table 4.1. Note that it is relatively constant over the time of the run.

4.4 Utility for CFSD

We show the results of our algorithm on data from the Chalmers formula student driverless and investigate how ORB-SLAM2 could aid in the specific usecase defined by the formula student driverless competition. We show the map along with the trajectory in Figure 4.17. The trajectory is consistent over the multiple laps and the cones positions can be discerned as clusters of points lining the trajectory.

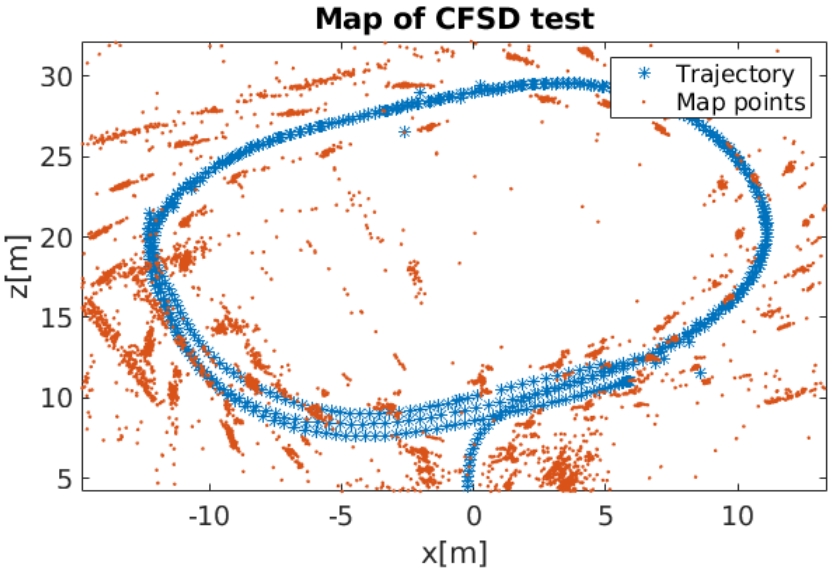


Figure 4.17: The trajectory along with the map gathered from three laps of a simple circle with cones marking the boundary. The trajectory is consistent over multiple laps and the position of the cones can be discerned as clusters on both sides of the trajectory.

5

Discussion

5.1 Performance in automotive applications

Below we detail the results for our dataset and discuss what it means for automotive applications.

5.1.1 KITTI

The KITTI dataset primarily confirms the results of the original paper wherein ORB-SLAM2 is able to map all of the sequences. The only exception being sequence 01 for the monocular case. In addition our results indicate that loop closing is also important for the performance in the stereo case in that the relative translation error seems to grow in the absence of loops but decrease when a loop is possible. We can also discern indications that ORB-SLAM2 has trouble establishing an accurate motion estimate at higher speeds as in sequence 01, however this could perhaps be mitigated with a higher framerate than KITTIs 10 fps. There is also an indication that ORB-SLAM2 has problem with motion estimation in sharper turns. The rotation error is however very good in comparison to other methods on the KITTI leaderboard but the problem might be the estimation of motion in scenes with high parallax.

5.1.2 Gothenburg sets

In our own gathered data the results are worse than in the KITTI dataset. The problem here seems to be an inability to estimate the motion during higher speeds on straights as well as the 180^o turn causing a significant rotation error. The motion error is also not uniform but seems to be larger during certain stretches of road where the ORB-features are fewer and further away. The reason for these larger errors in comparison to the KITTI dataset is most likely due to the lower baseline of our camera, the ZED camera which has a baseline of 12 cm compared to the KITTI setup which has a baseline of 54 cm. A lower baseline causes a larger error when estimating depth, especially at longer distances. Since depth is used to estimate forward motion a larger source of error for the depth estimation would cause a larger error during motion estimation.

5.1.3 Mono or stereo

The main difference between the monocular and stereo case is the scale ambiguity introduced by the monocular camera. Due to the scale ambiguity it is not possible to relate the monocular localization and mapping to metric units, which means that some kind of reference measurement is needed to rescale the values so that they can be related to the ground truth. Without an additional sensor this is very hard to do in real time so we conclude that the monocular system cannot accurately map in real time with respect to metric units without additional sensors. In contrast, stereo is able to resolve the scale ambiguity due to the ability to triangulate the mappoints from a single frame as well as being able to relate the measurement to the baseline in metric units. So stereo could be used in real-time to provide accurate localization and mapping in metric units which is relatable to other measurements. In other regards stereo is generally more accurate and robust since it is able to map several sequences such as sequence 01 in the KITTI set and our dataset where we mounted the camera on the hood which mono fails to do. Stereo also provides more dense mapping than mono due to being able to recover more mappoints. With regards to loop closing both systems depend on it but mono to a much larger extent since the significant error source scale drift is eliminated due to loop closure. However, our results for the KITTI dataset show a more accurate mapping also for the stereo case when a loop is present.

5.1.4 Baseline

To be able to perform accurate 3D reconstruction at higher speeds, for instance for use on the CFSD race car, a stereo camera with a sufficiently large baseline is needed. To determine a sufficient baseline we assume that we need to detect cones and map them at a distance of 15 m and that we need to be able to determine the depth with a precision of less than 1 m at this distance. To calculate the baseline needed for this we rewrite equation 2.8 to

$$d = \frac{fb}{n} \tag{5.1}$$

where d is the depth, f is the focal length in pixels, b is the baseline, and n is the pixel disparity between the two images. Using this equation we can determine at what distance we can achieve sub-meter precision (i.e. at what distance will a change in n produce a difference in depth smaller than one meter). We plot this as a function of the baseline in figure using the focal length of $f = 700$ in figure 5.1. By this we can infer that in order to achieve accurate reconstruction at a distance of 15 meters a baseline of 0.35 m is required. This metric is taken because we believe that sub-meter precision is the minimum required to achieve accurate 3D reconstruction.

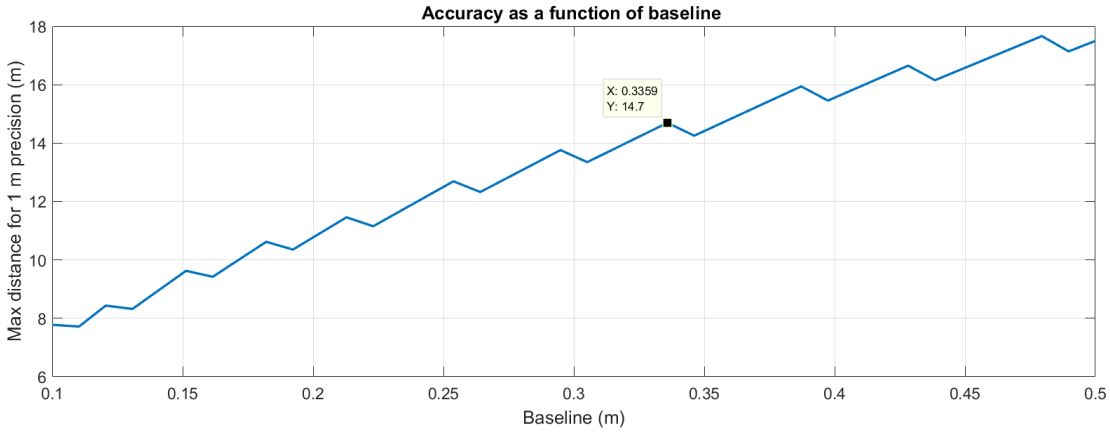


Figure 5.1: Distance of 1 meter precision as a function of the baseline. The edges of the curve are due to the discrete jumps of the function. Note that in order to achieve sub-meter precision at 15 m a 0.35 m baseline is required.

5.2 Further developments

Because the results of the KITTI data and further compounded by our own gathered data shows that motion estimation is a significant source of error, especially during higher speeds and scenes with parallax, an improvement would be to incorporate an external motion sensor into ORB-SLAM2 to better be able to estimate the motion. This could be as simple as a wheel encoder or something more complex like an IMU or GPS receiver. This would also mean that the scaling problem introduced by the monocular case would be resolved by introducing real world measurements. As it is now, the localization measurements given by the monocular SLAM is not relatable to other real world measurements.

5.3 Conclusion

5.3.1 General automotive use

For mapping with a more general purpose, for example mapping a portion of a city, it took us a couple of data gatherings to achieve some good results. Some interesting behaviour was that cloudy days made it quite hard for the algorithm to recognize the translational movement of the vehicle. Because of the distance to the clouds they seem static between frames, and if a big portion of the orb features are extracted from clouds the algorithm can have a problem with calculating the velocity and translation between frames, resulting that the vehicle is standing still. this can also happen on multi-lane roads where you have cars going the same speed as the own vehicle. One can reduce this problem by choosing the camera position wisely.

For highways with long straight lanes, the monocular camera has drawbacks as well, originating from the same behaviour as the clouds. There will not be much change between frames on that type of scenario so a portion of the features will be extracted

with similar position in each frame, resulting in the algorithm finding vehicles to stand still. With a stereo camera the performance on a high way is better due to the triangulation between the image pairs. Due to the observed drifting errors for both mono and stereo, while stereo handles the drifting error better a, loop closing is necessary to eliminate this error. This can be well observed in Figure 4.10 which lacks loop closing, and Figure 4.7 which has loop closing.

One of our aims was to have at least a frequency of 5 Hz, but when running on the minimum frequency the algorithm looses its localization. Our hypothesis is that the low frequency causes the localization to fail due to the big differences between frames, making it difficult to find good features to match. We saw with a data set recorded with twenty frames per second, the algorithm performed much more robustly in general.

The difference in performance between our own data set and the KITTI data set is noticeable, by analyzing the different cases we come to the conclusion that the ZED camera is not the best choice of camera setup for this usecase. We suspect that the short baseline of the ZED camera compared to the KITTI setup creates large errors in the velocity prediction. This is due to the distant map points will not have enough disparity to accurately estimate the depth. So a wider baseline for the camera setup would be preferred when mapping areas were there is a big chance of features being extracted from far away. Our final conclusion is that mono will not be able to handle the task to map in real-time using the ORB-SLAM2, the algorithm needs to be extended to handle map scaling by referred to a metric property, as discussed in section 5.1.3. Stereo vision is a solid solution for real-time mapping, if the user carefully chooses the camera setup as well as having knowledge about what can cause ORB-SLAM2 to fail or have large errors during mapping.

5.3.2 Formula student competition

The main feature that the software embedded in the car is acting on is cones. this is due to the track perimeter consisting of cones for all events in the competition. For ORB-SLAM2 to be a feasible solution the algorithm has to see the cones as potential places to extract corner features from, so the perception subsystem can send out cone positions to the next subsystem. After running ORB-SLAM2 on the own dataset where we set up a circular track on a parking lot, we could see that the cones are defined and has a good amount of features. Then the next question is, how do we manage the local map reconstructed from the image to be able to identify what is a cone, and what is not. The ORB map would then need to be processed with another algorithm that cluster the features for the cones and detect them. In the race car, there is already a nearest neighbour clustering algorithm to detect cones from lidar point clouds. This method could be used for ORB as well, with modifications for how to analyze the features. One main upside of ORB-SLAM2 is that the total system is very lightweight. The race car has a limited power supply and the benefits of only needing one or two cameras combined with a fast enough CPU to perform mapping and localization.

One of the main concerns we saw was that for our setup with the ZED camera the velocity prediction based on depth perception was not accurate enough. As described in the section above we derive this conclusion to be caused by the small baseline of the ZED camera, since distance points will not have enough disparity to accurately estimate the depth. The Lindholmen dataset shows that the distance travelled is not close to the measured GPS readings. If this problem arises during the mapping of the competition track the car would certainly go out of bounds due to the speed, and that steering requests would be based on a track that is not correctly scaled. Therefore, in the situation of the race car where the camera baseline is small, it might be a better idea to implement a traditional SLAM algorithm based on the lidar cone detections. However, ORB2-SLAM could be a great option given an external input that could resolve the scaling problem, for example using an IMU for more accurate localization. In addition we would need a camera with a baseline of 35 cm to achieve accurate 3D reconstruction at the cone detection range achieved with the lidar.

Bibliography

[1] Mohammad O. A. Aqel, Mohammad H. Marhaban, M. I. Saripan, and Napsiah B. Ismail. Review of visual odometry: types, approaches, challenges, and applications. *SpringerPlus*, 5(1):1–26, 2016.

[2] David Arthur and Sergei Vassilvitskii. k-means: the advantages of careful seeding. volume 7-09-, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[3] Christian Berger, Björnborg Nguyen, and Ola Benderius. Containerized development and microservices for self-driving vehicles: Experiences best practices. 2017.

[4] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, pages 778–792, Berlin, Heidelberg, 2010. Springer-Verlag.

[5] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007.

[6] Ethan Eade. Lie groups for 2d and 3d transformations. 05 2017.

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

[8] Martin Fowler and James Lewis. Microservices: a definition of this new architectural term. 2015.

[9] Andrea Fusiello. Elements of geometric computer vision, 2015.

[10] Dorian Gálvez-López and J. D. Tardós. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, October 2012.

[11] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[12] 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.

[13] Kenji Hata and Silvio Savarese. Cs231a course notes 1: Camera models.

[14] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *J. Opt. Soc. Am. A*, 4(4):629–642, Apr 1987.

[15] Sing Bing Kang, Jon Webb, and C Lawrence Zitnick. An active multibaseline stereo system with real-time image acquisition. 11 1999.

[16] R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3607–3613, Shanghai, China, May 2011.

[17] Velodyne LiDAR. Vlp-16, 2017. [Online; accessed 2018-05-12].

[18] Michael Montemerlo, Sebastian Thrun, Daphne Roller, and Ben Wegbreit. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges.

[19] R. Mur-Artal and J. D. Tardos. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. *ArXiv e-prints*, October 2016.

[20] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[21] Albin Pålsson and Markus Smedberg. Investigating simultaneous localization and mapping for agv systems. Master's thesis, 2017. 72.

[22] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443, May 2006.

[23] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, Nov 2011.

[24] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. In *Autonomous robot vehicles*, pages 167–193. Springer, 1990.

[25] STEREOLABS. Meet the zed stereo camera, 2017. [Online; accessed 2018-05-12].

[26] SBG Systems. Ellipse2-n: Miniature ins/gps, 2017. [Online; accessed 2018-05-12].

[27] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment – a modern synthesis. volume 1883, pages 298–372, 2000.

[28] Wikipedia contributors. Pinhole camera model — Wikipedia, the free encyclopedia, 2018. [Online; accessed 2018-04-25].

[29] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. An overview to visual odometry and visual slam: Applications to mobile robotics. *Intelligent Industrial Systems*, 1(4):289–311, Dec 2015.

A

Data Gathering

Additional material about the data gathering is presented below.

GPS trace

The GPS trace is presented in Figure A.1

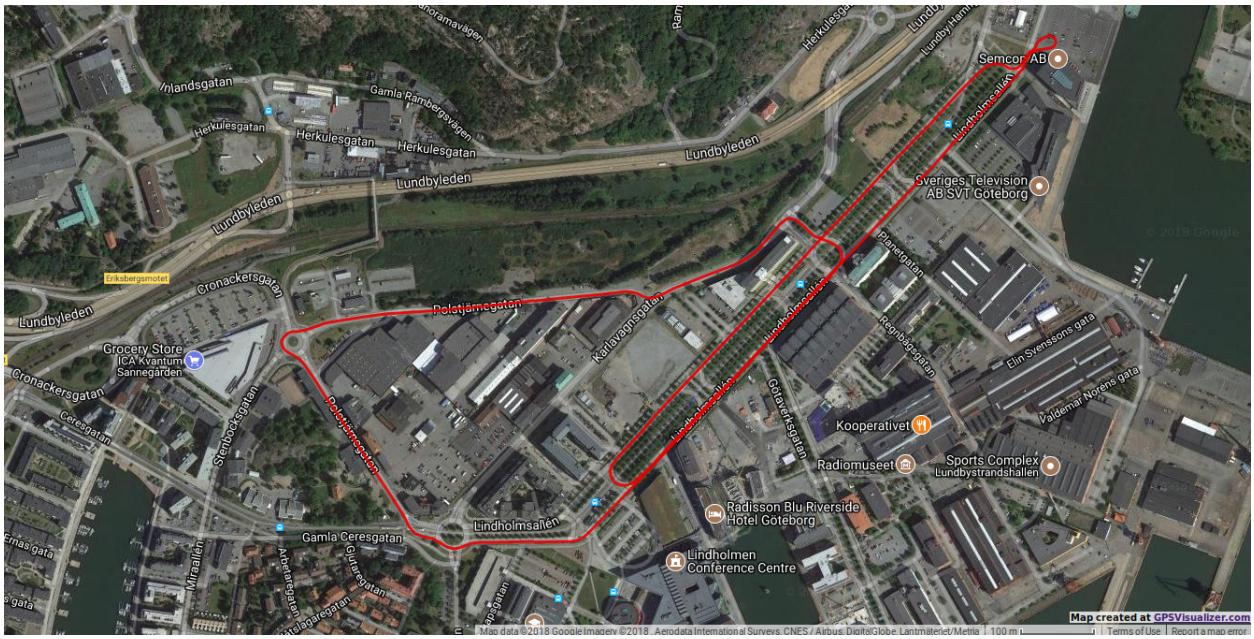


Figure A.1: GPS trace for the dataset at Lindholmen. Plotted through GPSvisualizer.com with map data provided by Google Imagery along with AeroData International Surveys, CNES/Airbus, DigitalGlobe, Lantmäteriet/Metria. Map URL: `http://www.gpsvisualizer.com/display/20180515071129-21046-map.html`

Camera Setup

Roof

Figure A.2: Camera setup for camera position on the roof.

Figure A.3: Camera setup for camera position on the roof.

Figure A.4: Camera setup for camera position on the roof.

Hood

Figure A.5: Camera setup for camera position on the hood.

Figure A.6: Camera setup for camera position on the hood.

Figure A.7: Camera setup for camera position on the hood.