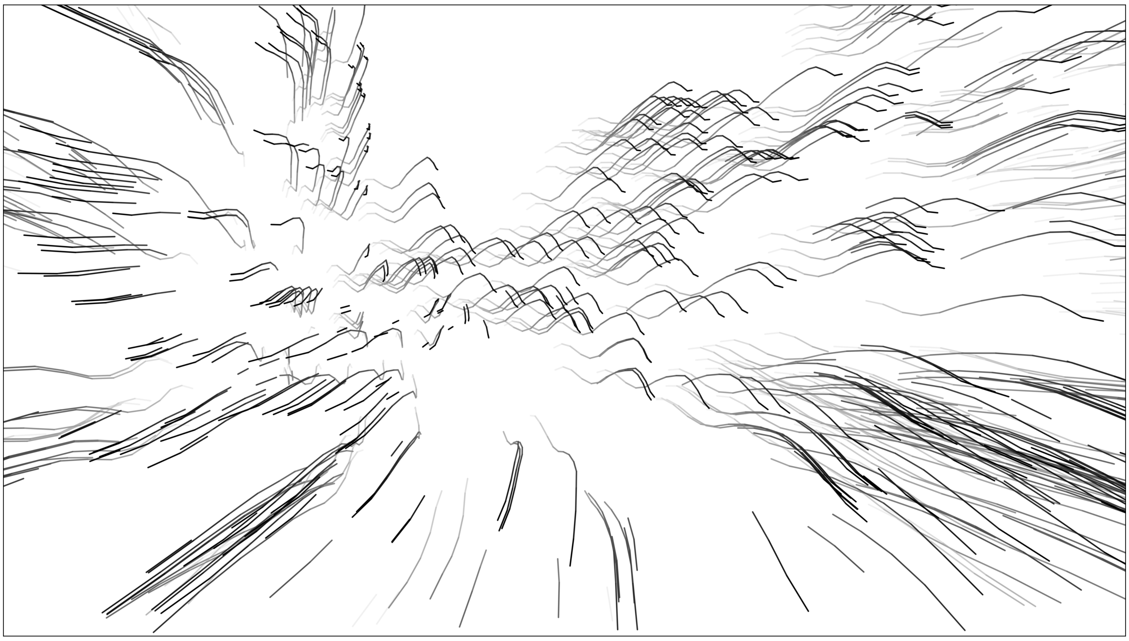
CHALMERS
UNIVERSITY OF TECHNOLOGY



Monocular Simultaneous Localisation and Mapping for Road Vehicles

Master's thesis in Systems, Control and Mechatronics

MATHIAS ERNST, SAMUEL SCHEIDEGGER

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2015

Master's thesis EX027/2015

Monocular Simultaneous Localisation and Mapping for Road Vehicles

MATHIAS ERNST, SAMUEL SCHEIDEGGER

Department of Signals and Systems
Signal processing and Biomedical engineering
Chalmers University of Technology
Gothenburg, Sweden 2015

Monocular Simultaneous Localisation and Mapping for Road Vehicles

MATHIAS ERNST, SAMUEL SCHEIDEGGER

Supervisor: Erik Stenborg, Volvo Car Corporation
Examiner: Lennart Svensson, Department of Signals and Systems

Master's thesis
Department of Signals and Systems
Signal processing and Biomedical engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration of feature points tracked over a sequence of images.

Typeset in LaTeX
Printed by [Name of printing company]
Gothenburg, Sweden 2015

Monocular Simultaneous Localisation and Mapping for Road Vehicles

MATHIAS ERNST, SAMUEL SCHEIDEGGER
Department of Signals and Systems
Chalmers University of Technology

Abstract

Autonomous cars hold great promise for the future of transportation enabling more efficient driving and safer vehicles for both the occupants and other road users. A key aspect to achieving autonomous driving is accurate positioning of the vehicle. One way of meeting this goal is to build a map and simultaneously perform localisation within that map. This problem in known as the Simultaneous Localisation and Mapping (SLAM) problem. To avoid each vehicle having to build its own map, one can also localise the vehicles position within a previously made map, and such a map could be made using SLAM methods.

This thesis explores a few different approaches to solving the SLAM problem and describes the implementation of a graph-based SLAM system using monocular camera data. The system that has been developed uses feature tracking, epipolar geometry and local bundle adjustment for visual odometry, relying on velocity and time measurements to recover the information lost in the scale ambiguity of monocular image data. Topological loop closures are found by using the FAB-MAP 2.0 algorithm. The developed system uses the OpenCV library for feature detection and description, the g2o and CHOLMOD libraries for graph-representation and bundle adjustment, and the OpenFABMAP library for running the FAB-MAP 2.0 algorithm. The algorithm is tested on the public KITTI datasets and is shown to successfully build consistent maps and localise within them.

Keywords: Visual SLAM, Monocular, Visual Odometry, Bundle Adjustment.

Acknowledgements

We would like to thank our supervisors, Erik Stenborg and Anders Karlsson, and our examiner, Lennart Svensson, for support during our work. We also would like to thank the authors of our sister project, Robin Lindholm and Carl-Johan Pålsson, for the interesting discussions, and the other participants in our study group, Lars Hammarstrand, Nicklas Gustafsson, Maryam Fatemi and Malin Lundgren for their useful input. And also, we would like to thank Viktor Lindström for help debugging our code and Mikael Persson for his expert help on the subject. Finally we would like to thank Volvo Car Corporation for the opportunity to perform this work.

Mathias Ernst, Samuel Scheidegger, Gothenburg, June 2015

Contents

List of Figures

List of Tables

List of Acronyms

BA Bundle Adjustment.

BoW Bag-Of-Words.

BRIEF Binary Robust Independent Elementary Features.

BRISK Binary Robust Invariant Scalable Keypoints.

CCD Charge-Coupled Device.

CENSURE Center Surround Extrema.

CMOS Complementary Metal Oxide Semiconductor.

DoF Degrees Of Freedom.

DoG Difference-Of-Gaussian.

EIF Extended Information Filter.

EKF Extended Kalman Filter.

FAB-MAP Fast Appearance-Based Mapping.

FAST Features from Accelerated Segment Test.

GNSS Global Navigation Satellite System.

GPS Global Positioning System.

GPU Graphics Processing Unit.

IAC Image of Absolute Conic.

ICR Instantaneous Centre of Rotation.

IF Information Filter.

IMU Inertial Measurement Unit.

KLT Kanade–Lucas–Tomasi.

LSD-SLAM Large Scale Direct SLAM.

ORB Oriented FAST and Rotated BRIEF.

PTAM Parallel Tracking And Mapping.

RANSAC Random Sample Consensus.

RMS Root Mean Square.

SeqSLAM Sequence SLAM.

SIFT Scale Invariant Feature Transform.

SLAM Simultaneous Localisation And Mapping.

SURF Speeded Up Robust Features.

SVD Singular Value Decomposition.

UKF Unscented Kalman Filter.

V-SLAM Visual Simultaneous Localisation And Mapping.

VCC Volvo Car Corporation.

VO Visual Odometry.

1

Introduction

In 2011, an estimated 1.3 million people where killed in road traffic accidents; by 2020 this is expected to grow to 1.9 million people, annually [1]. In 2015, a study by the U.S. Department of Transportation showed that 94 % of the road traffic accidents in the U.S. were caused by drivers [2]. The number of road traffic accidents could be reduced by identifying when a driver is about to make a mistake that will cause an accident, and take action to prevent the accident. Volvo Car Corporation (VCC) has such a system in production today, in form of the City Safety system, which intends to autonomously brake the car to avoid collisions at low speeds. This concept could be expanded, to further reduce accidents, with other systems, e.g. lane keeping or even fully autonomous vehicles. Considering the vast number of accidents caused by human error, it is clear that autonomous vehicles that are less prone to driver errors has the potential to drastically reduce the number of accidents. Furthermore, autonomous vehicles has the potential to free up billions of hours for people currently spent acting as the human control system of the vehicles

The ever-increasing vehicle ownership and energy demand from the transport sector pose a concern for the environment. The worldwide transport sector, in 2010, stood for 28 % of the total energy consumption and caused 23 % of total energy-related CO_2 emissions with a majority, 72 %, coming from road vehicles [3, pp. 603]. Efforts have been made to develop more energy efficient engines, and to use alternative energy sources. For example, hybrid vehicles, using both internal combustion engines and electric motors, and purely electric vehicles, have been developed. No matter which propulsion system is used, further gains in energy efficiency can be made by adopting efficient driver behaviour. In a fully autonomous vehicle this can be done automatically. The energy consumption can be minimised by route planning, platooning, better control of the power components, and regenerative drive system [4, pp. 5] to a degree not achievable by a human driver.

Autonomous vehicles is a hot research topic worldwide, as they can make traffic safer and more efficient in terms of both energy use and traffic flow and several experimental platforms have been tested. Google has been testing self driving cars since 2009, using expensive sensors such as LIDARs, on public roads [5]. Initially the test were limited to freeways, but later also included more complex city streets. In 2014 Daimler successfully tested a car on a route with a variety of difficult traffic situations, using close-to-production sensor hardware. VCC plans to put 100 cars on the roads of Gothenburg by 2017, that will be self-driving on a number of predetermined roads.

1.1 Problem background

To enable autonomous systems, sensor data input is required. For lane keeping, a system which estimates the road lane and the vehicle's lateral position in the lane is necessary. For a fully autonomous vehicle, a map of the surroundings and other traffic is required [7]. The limited accuracy of Global Navigation Satellite Systems (GNSSes), such as the Global Positioning System (GPS), and the poor performance or non-existent signal in some environments, such as urban canyons or tunnels, make them insufficient for such applications [8]. Therefore, the use of other types of sensors is required to replace or complement GNSSes. Such other sensors could for example be wheel odometry, steering angle sensors, range sensors, such as LIDAR or radar detectors, or cameras. A problem in common for using these types of sensors to estimate position, is that the estimates are incremental and noisy, and thus subject to a drift. One consequence of the drift when building a map is that the representation of the location in the map depends on the measurement noise and the path taken to that location, causing inconsistent maps. To reduce the drift, information from a GPS, which is absolute in the global frame and thus does not suffer from drift, could be fused with sensors that are more accurate in the local frame. Another technique to reduce drift and enforce consistency is to adjust the estimated trajectory when visiting a location with previously known information about the position. To accomplish this, a map in which it is possible to find the ego-pose has to be built, and to build a map, a known position is required. This "chicken or the egg" problem is often referred to as the Simultaneous Localisation And Mapping (SLAM) problem [9, pp. 222].

1.2 Purpose

The purpose of this work is to evaluate how digital cameras mounted on a vehicle can be used to estimate the ego-motion of the vehicle and build a map, in which it is possible to localise the ego pose. This concept is commonly referred to as the Visual Simultaneous Localisation And Mapping (V-SLAM) problem.

A review of the current literature on the subject should be done, and the most common and most promising methods to solve the SLAM problem should be investigated. A solution to the V-SLAM problem should be proposed and evaluated. Further improvements to the system, unresolved issues, and areas and techniques that would be useful to explore further should be pointed out for future work. An evaluation of how the V-SLAM techniques could be deployed for building a large scale map should be considered. Aspects such as the accuracy of the mapping and localisation, the robustness in various road environments, and feasibility should be evaluated. Lastly, avenues for further investigation in the area of V-SLAM should be identified.

1.3 Delimitations

In this project a full 6 Degrees Of Freedom (DoF) SLAM system was built using monochrome monocular image data from a calibrated camera. Vehicle speed measurements were used to resolve the scale ambiguity inherent in monocular camera data. he system does not handle dynamic scenes by classifying moving objects, such as other traffic, and taking this into consideration in the map building. Even though computational complexity of the different algorithms is considered, the system is not intended to run in real-time.

The implementation takes unrectified images and speed measurements as inputs, and produces an estimate of the trajectory and a map in the form of a point cloud of estimated feature positions. For loop closure detections, training data in the form of a set of images from environments similar to the ones to be driven in is also needed.

1.4 Methodology

The thesis work began with a study of the SLAM problem based on the lectures and exercises from the course Robot Mapping[1] held by Cyril Stachniss at Freiburg University. A further literature study regarding V-SLAM in particular was done, covering such topics as camera calibration, projective geometry, image features, Visual Odometry (VO), and appearance based loop closures.

The first implementations of the VO system were made in Matlab, but the chosen solution was later reimplemented in C++ to be able to utilise certain third party libraries. The performance of the system was evaluated on datasets from the KITTI Vision Benchmark Suite[2] [10]. The KITTI datasets are logs from sensors mounted on a car when different paths where driven. The sensors include 4 forward looking cameras, 1 64 layer LIDAR, and a highly accurate combined Inertial Measurement Unit (IMU) and GNSS. Evaluation was performed in the same manner as in the KITTI Benchmark, to enable comparison with other solutions, i.e. checking the average accumulated deviation from ground truth in subsequences of certain lengths.

1.5 Related work

VO, or estimation of the ego-motion from consecutive camera images has previously been studied in the literature, and several different solutions have been proposed. One solution, proposed by Nistér *et al.* in 2004, uses a stereo camera and feature tracking, and achieves successful results from aerial, automotive and handheld platforms [11]. A lane-based solution, LaneLoc, purely for road vehicles, using pre-built maps in which on-line localisation is performed, was initially proposed by Schreiber *et al.* in 2013 [12] and was later used by Ziegler *et al.* in 2014 in Daimler's Bertha

[1] `http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/`

[2] `http://www.cvlibs.net/datasets/kitti/`

project [6]. In 2013 Engel *et al.* proposed a semi-dense direct method, using image gradients, achieving real-time performance on a modern smartphone [13].

Different solutions to the V-SLAM problem has been proposed. In 2003 Davison used an Extended Kalman Filter (EKF) and feature points for mapping and pose estimation. The EKF based solutions were outperformed by a particle filter based solution by Montemerlo *et al.* in 2002 [15], FastSLAM, which was then improved, and named FastSLAM 2.0, by Montemerlo *et al.* in 2003 [16]. A solution inspired by the hippocampus of rodents, RatSLAM, was proposed by Milford *et al.* in 2004 [17]. In 2007, Klein and Murray proposed a solution, Parallel Tracking And Mapping (PTAM), which broke the barrier to real-time direct augmented reality with inexpensive cameras. A large scale real-time V-SLAM solution was proposed by Lim *et al.* in 2014 [19] using feature point matching and graph based SLAM.

Solutions for pure loop closure detection have been proposed. Cummins and Newman in 2008 proposed Fast Appearance-Based Mapping (FAB-MAP) [20], which was improved in 2011 by Cummins and Newman to FAB-MAP 2.0 [21]. Both versions detects loop closures by exploiting highly distinctive feature points. Milford and Wyeth in 2012 proposed Sequence SLAM (SeqSLAM), which uses sequences instead of single images. These loop closure detection algorithms have been used by various implementations for VO, to achieve a full V-SLAM solution, e.g. the semi-dense direct VO solution, by Engel *et al.* in 2013, was improved by Engel *et al.* in 2014 to Large Scale Direct SLAM (LSD-SLAM), a full V-SLAM solution by using FAB-MAP 2.0 for loop closure detection in real-time.

2

Theory

In this chapter, the theory on which the implemented system relies is presented. Starting with the theory on how the camera maps the environment to an image, continuing with how to incrementally estimate the ego-motion from the images using visual odometry, and ending with how SLAM algorithms can be used to localise and build large-scale, consistent maps of the environment. While the final implementation is a graph-based SLAM system, EKF SLAM and the particle filter based FastSLAM algorithm were considered during the study of the SLAM problem and are also presented here.

2.1 Camera

A camera maps reflected light from 3D objects in space onto a 2D image [24, pp. 153]. Typically, the light reflected by the environment is projected, using optical lenses, onto a Complementary Metal Oxide Semiconductor (CMOS) or Charge-Coupled Device (CCD) sensor, which both use the photoelectric effect [25] and analogue to digital converters to digitise the image. To use the images to build a map of the environment, the way the camera projects the environment to the image must be known. The maps considered here are locations of points on the surface of objects in the world. A mathematical camera model is used to represent the camera projection.

2.1.1 Camera model

The basic pinhole camera model projects a point in space onto an image point on a plane, called the image plane or the focal plane. Let the centre of projection, the camera centre, $\mathbf{C}$, be the origin of a Euclidean coordinate system. A point in space, $\mathbf{X} = (X, Y, Z)^T$, is projected to a point, $\mathbf{x} = (x, y)^T$, on the image plane, $Z = f$, where the line connecting the centre of projection and the point in space intersects the image plane, as shown in figure 2.1. It can be seen that the point in space, $\mathbf{X}$, in $\mathbb{R}^3$ is mapped to the image point, $\mathbf{x}$, in $\mathbb{R}^2$, as follows [24, pp. 154]:

$$\begin{pmatrix} X, & Y, & Z \end{pmatrix}^T \mapsto \begin{pmatrix} f \cdot X/Z, & f \cdot Y/Z \end{pmatrix}^T . \tag{2.1}$$

The line which is perpendicular to the image plane and meets the camera centre, is called the principal axis. The point where the principal axis meets the image plane is called the principal point, $\mathbf{p}$, or the optical centre.

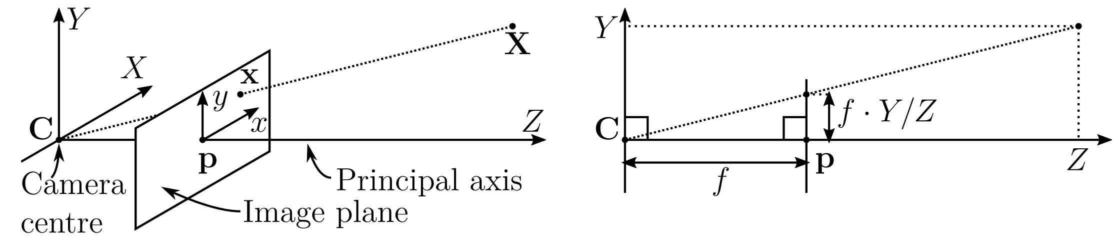


Figure 2.1: Basic pinhole camera model. Shows how the point, **X**, in space is projected onto the image plane, the principal point, **p**, and the camera centre, **C**.

If the world point and image point are represented as homogeneous coordinates, the mapping can be written as a matrix multiplication:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} f \cdot X \\ f \cdot Y \\ Z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{2.2}$$

Hereafter **X** will represent the world point as a homogeneous 4-vector $(X, Y, Z, 1)^T$ and **x** will represent the image point as a homogeneous 3-vector $(x, y, 1)^T$, then (2.2) can be written as

$$\mathbf{x} = P\mathbf{X}_{cam}, \tag{2.3}$$

where P is the 3×4 homogeneous camera projection matrix and $\mathbf{X}_{cam}$ is a point in space in the coordinate system with the origin in the camera centre.

In (2.2) it is assumed that the principal point is in the origin of the image plane coordinate system, which it may not be, so in general the mapping is [24, pp. 155]

$$\left(X, \quad Y, \quad Z\right)^T \mapsto \left(f \cdot X/Z + p_x, \quad f \cdot Y/Z + p_y\right)^T = \left(x_{cam}, y_{cam}\right)^T, \tag{2.4}$$

where x_{cam} and y_{cam} are the image pixel coordinates with the principal point at $\mathbf{p} = (p_x, p_y)^T$, as in figure 2.2. This can be represented as homogeneous coordinates:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} f \cdot X + Z \cdot p_x \\ f \cdot Y + Z \cdot p_y \\ Z \end{pmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{2.5}$$

By expressing K as

$$K = \begin{bmatrix} f & & p_y \\ & f & p_y \\ & & 1 \end{bmatrix}, \tag{2.6}$$

(2.5) can be written as

$$\mathbf{x} = K\left[I \mid \mathbf{0}\right]\mathbf{X}, \tag{2.7}$$

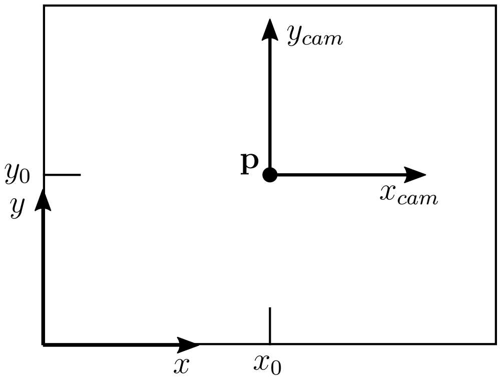


Figure 2.2: The principal point with offset from the image coordinate origin.

where K is called the camera calibration matrix and $[I|\mathbf{0}]$ denotes I and $\mathbf{0}$ stacked horizontally.

This model assumes square pixels. In the case of a camera, with a CCD or CMOS sensor, this is not a necessity. To compensate for this, unequal scale factors can be introduced, and the camera calibration matrix, K, can be generalised as

$$K = \begin{bmatrix} m_x & & \\ & m_y & \\ & & 1 \end{bmatrix} \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} = \begin{bmatrix} \alpha_x & & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix}. \tag{2.8}$$

For added generality, a skew parameter, s, can be added:

$$K = \begin{bmatrix} \alpha_x & s & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix}. \tag{2.9}$$

For normal cameras, $s = 0$. If $s \neq 0$, this can be interpreted as a skewing of the pixels in a sensor array, as if the x- and y-axes of the pixels were not perpendicular. This can occur if the image is a photography of an image itself, where the principal axis is not perpendicular to the plane of the image [24, pp. 164].

In general, the world points will be expressed in a world coordinate system, which usually will not coincide with the camera coordinate system. The relation between the two coordinate systems can be expressed as a rotation and a translation. If $\tilde{\mathbf{X}}$ is an inhomogeneous 3-vector representing a point in space, expressed in world coordinates, the coordinates of the point in the camera coordinate system can be written as $\tilde{\mathbf{X}}_{cam} = R(\tilde{\mathbf{X}} - \tilde{\mathbf{C}})$ [24, pp. 156], where R is a 3×3 rotational matrix representing the orientation of the camera coordinate system and $\tilde{\mathbf{C}}$ the coordinates of the camera origin expressed in world coordinates. In homogeneous coordinates, this can be expressed as

$$\mathbf{X}_{cam} = \begin{bmatrix} R & -R\tilde{\mathbf{C}} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{X}, \tag{2.10}$$

which, with (2.7), gives

$$\mathbf{x} = KR\left[I \mid -\tilde{\mathbf{C}}\right]\mathbf{X}, \tag{2.11}$$

and the camera matrix, P:

$$P = KR\left[I \mid -\tilde{\mathbf{C}}\right]. \tag{2.12}$$

For convenience, the camera centre is usually not expressed in world coordinates, but with $\mathbf{t} = -R\tilde{\mathbf{C}}$, which gives the camera matrix

$$P = K\left[R \mid \mathbf{t}\right]. \tag{2.13}$$

K is often referred to as the internal camera parameters, or intrinsic matrix, and $[R|\mathbf{t}]$ as the external camera parameters, or the extrinsic matrix. Finding K is seen as a part of the calibration of the camera. If K has the form as in (2.9), the model is called finite projective camera and has 11 degrees of freedom [24, pp. 175].

This linear model will only hold for cameras with pinhole lenses. For cameras with optical lenses, i.e. most cameras, the world point will not be collinear with the projected point and the camera centre [24, pp. 189]. The most significant error is a radial distortion caused by the lens, as illustrated in figure 2.3, which becomes more severe with shorter focal length and lesser optical quality. To still be able to use a linear camera model, the image can be corrected to what it would have been with a perfect linear camera. The correction for radial distortion has to be applied as a first step in the image processing [24, pp. 190]. A world point in camera coordinates, $\mathbf{X}_{cam}$, gets linearly projected onto an image plane at $(\tilde{x}, \tilde{y})^T$ in focal length units as

$$\begin{pmatrix}\tilde{x}, & \tilde{y}, & 1\end{pmatrix}^T = \left[I \mid \mathbf{0}\right]\mathbf{X}_{cam}. \tag{2.14}$$

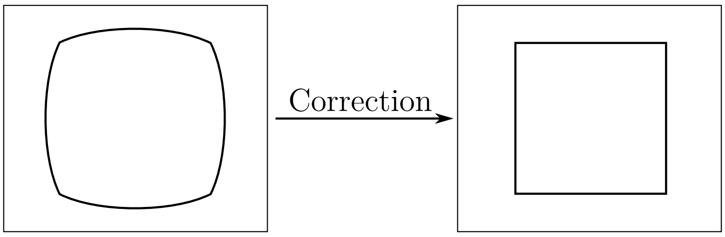


Figure 2.3: Sketch of how a heavily distorted image of a perfect square would be corrected.

The actual projected point $(x_d, y_d)^T$ can be related to the ideal point as

$$\begin{pmatrix}x_d\\ y_d\end{pmatrix} = \mathrm{L}(\tilde{r})\begin{pmatrix}\tilde{x}\\ \tilde{y}\end{pmatrix}, \tag{2.15}$$

where $\tilde{r} = \sqrt{\tilde{x}^2 + \tilde{y}^2}$ is the radial distance from the centre of radial distortion and $\mathrm{L}(\tilde{r})$ is a distortion factor. This can be written in pixel coordinates as

$$\hat{x} = x_c + \mathrm{L}(r)(x - x_c) \quad \hat{y} = y_c + \mathrm{L}(r)(y - y_c), \tag{2.16}$$

where $(x, y)^T$ are the measured coordinates, $(\hat{x}, \hat{y})^T$ are the corrected coordinates, $(x_c, y_c)^T$ the centre of radial distortion and $r = \sqrt{(x - x_c)^2 + (y - y_c)^2}$ the radial distance from the centre of radial distortion. $\mathrm{L}(r)$ is defined only for positive values and $\mathrm{L}(0) = 1$. $\mathrm{L}(r)$ is usually approximated by a Taylor expansion $\mathrm{L}(r) = 1 + \kappa_1 r + \kappa_2 r^2 + \kappa_3 r^3 + \ldots$. Determining the coefficients, κ_n and the centre of radial distortion, $(x_c, y_c)^T$, is usually considered as part of the calibration of the camera.

2.1.2 Camera calibration

A camera is calibrated when the image can be corrected to linear projection and K is known. In this case, the projection of a world point on the image plane can be related to the direction to the world point from the camera centre, and can be written $\tilde{\mathbf{X}} = \lambda \mathbf{d}$. This would then map to the image point as $\mathbf{x} = K[I|\mathbf{0}](\lambda \mathbf{d}^T, 1)^T = K\mathbf{d}$, up to a scale, and the direction of an image point would thus be $\mathbf{d}$ [24, pp. 208].

The angle between two image points can, according to the cosine formula for the angle between two vectors, be written as

$$\begin{aligned}\cos\theta &= \frac{\mathbf{d}_1^T\mathbf{d}_2}{\sqrt{\mathbf{d}_1^T\mathbf{d}_1}\sqrt{\mathbf{d}_2^T\mathbf{d}_2}} = \frac{(K^{-1}\mathbf{x}_1)^T(K^{-1}\mathbf{x}_2)}{\sqrt{(K^{-1}\mathbf{x}_1)^T(K^{-1}\mathbf{x}_1)}\sqrt{(K^{-1}\mathbf{x}_2)^T(K^{-1}\mathbf{x}_2)}} \\ &= \frac{\mathbf{x}_1^T(K^{-T}K^{-1})\mathbf{x}_2}{\sqrt{\mathbf{x}_1^T(K^{-T}K^{-1})\mathbf{x}_1}\sqrt{\mathbf{x}_2^T(K^{-T}K^{-1})\mathbf{x}_2}}.\end{aligned} \tag{2.17}$$

This shows that the angle between the direction of two points can be measured if K is known. It can also be shown that a scene plane defined by an image line, $\mathbf{l}$ and the camera centre, with the normal, $\mathbf{n}$, can be related by K, such that $\mathbf{n} = \mathbf{l}K$ [24, pp. 209].

To find the parameters of the intrinsic matrix, K, the geometry of points projected from infinitely far away is used. Points on the plane at infinity, $\boldsymbol{\pi}_\infty$, can be written as $\mathbf{X}_\infty = (\mathbf{d}^T, 0)^T$, and are projected onto an image plane with $H = KR$ as

$$\mathbf{x} = KR\left[I \mid -\tilde{\mathbf{C}}\right]\begin{pmatrix}\mathbf{d}\\0\end{pmatrix} = KR\mathbf{d} = H\mathbf{d}. \tag{2.18}$$

This projection is independent of the camera position, $\mathbf{C}$, and only depends on the internal parameters of the camera and rotation, and thus the calibration can be made independent of the position of the camera.

A conic is a curve described by a second degree equation in the plane [24, pp. 30]:

$$ax^2 + bxy + cy^2 + dx + ey + f = 0, \tag{2.19}$$

or in homogeneous coordinates

$$ax_1^2 + bx_1x_2 + cx_2^2 + dx_1x_3 + ex_2x_3 + fx_3^2 = 0. \tag{2.20}$$

This can be written in matrix form as

$$\mathbf{x}^T C\mathbf{x} = 0, \tag{2.21}$$

where

$$C = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}. \tag{2.22}$$

Under the point transformation (2.18), a conic is transformed as $C' = H^{-T}CH^{-1}$, since $\mathbf{x}^T C\mathbf{x} = \mathbf{d}^T H^{-T}CH^{-1}\mathbf{d} = \mathbf{d}^T C'\mathbf{d}$.

Points on the absolute conic, Ω_∞, a conic on a plane at infinity, $\boldsymbol{\pi}_\infty$, satisfies [24, pp. 81]

$$\left.\begin{matrix} x_1^2 + x_2^2 + x_3^2 \\ x_4 \end{matrix}\right\} = 0, \tag{2.23}$$

which gives

$$\begin{pmatrix} x_1, & x_2, & x_3 \end{pmatrix} I \begin{pmatrix} x_1, & x_2, & x_3 \end{pmatrix}^T = 0. \tag{2.24}$$

Thus, the C that describes Ω_∞ has to be the identity matrix and Ω_∞ is also a conic of purely imaginary points. The conic, Ω_∞, maps to $\boldsymbol{\omega} = H^{-T}CH^{-1} = H^{-T}IH^{-1} = (KR)^{-T})I(KR) = (KK^T)^{-1}$. Thus, if $\boldsymbol{\omega}$ is determined, K is also determined. $\boldsymbol{\omega}$ is called the Image of Absolute Conic (IAC).

It can be shown that all circles intersect Ω_∞, and that the support plane for a circle, $\boldsymbol{\pi}$, intersect $\boldsymbol{\pi}_\infty$ in a line and this line intersects Ω_∞ in two points [24, pp. 28]. These two points are the circular points of $\boldsymbol{\pi}$. The projected circular points lie on $\boldsymbol{\omega}$, and are also vanishing points of $\boldsymbol{\pi}$.

The vanishing line of a plane $\boldsymbol{\pi}$ is the line where $\boldsymbol{\pi}$ intersects $\boldsymbol{\pi}_\infty$. This line is the same for all parallel planes [24, pp. 82]. The line can be constructed by two vanishing points of a plane, the points where two parallel, projected, lines on the plane converge, as in figure 2.4.

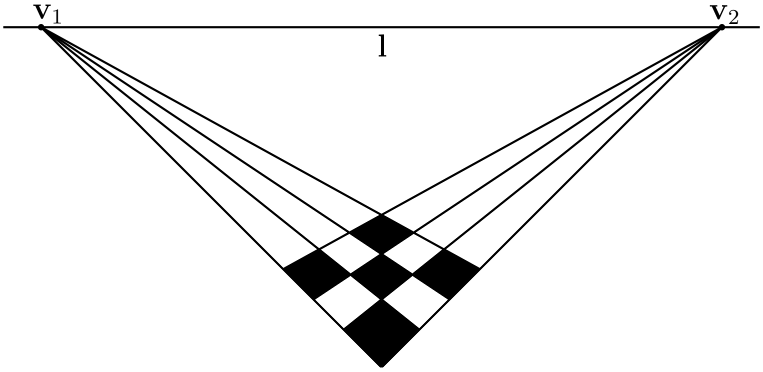


Figure 2.4: The projected parallel lines on a plane converges to the vanishing points, $\mathbf{v}_1$ and $\mathbf{v}_2$, and constructs the vanishing line, $\mathbf{l}$.

Using the result from (2.17) gives

$$\cos\theta = \frac{\mathbf{v}_1^T \boldsymbol{\omega} \mathbf{v}_2}{\sqrt{\mathbf{v}_1^T \boldsymbol{\omega} \mathbf{v}_1}\sqrt{\mathbf{v}_2^T \boldsymbol{\omega} \mathbf{v}_2}}, \tag{2.25}$$

which gives that the vanishing points of two perpendicular directions satisfy [24, pp. 219]

$$\mathbf{v}_1^T \boldsymbol{\omega} \mathbf{v}_2 = 0. \tag{2.26}$$

Writing the IAC as

$$\boldsymbol{\omega} = \begin{bmatrix} \omega_1 & \omega_2 & \omega_4 \\ \omega_2 & \omega_3 & \omega_5 \\ \omega_4 & \omega_5 & \omega_6 \end{bmatrix}, \tag{2.27}$$

it can be shown that cameras with zero skew will put a constraint on $\boldsymbol{\omega}$ such that

$$\omega_2 = 0 \tag{2.28}$$

and a camera with square pixels in addition give a constraint such that

$$\omega_1 = \omega_3. \tag{2.29}$$

Representing $\boldsymbol{\omega}$ as a 6-vector $\mathbf{w} = (\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6)^T$, constraints in (2.26), (2.28) and (2.29) can be written in the form of $\mathbf{a}^T\mathbf{w} = 0$.

For (2.26), with $\mathbf{v} = (v_1, v_2, v_3)^T$ and $\mathbf{u} = (u_1, u_2, u_3)^T$ as vanishing points of two perpendicular directions, $\mathbf{a}$ can be written as [24, pp. 225]

$$\mathbf{a} = \Big(v_1u_1, \quad v_1u_2 + v_2u_1, \quad v_2u_2, \quad v_1u_3 + v_3u_1, \quad v_2u_3 + v_3u_2, \quad v_3u_3\Big)^T, \tag{2.30}$$

for (2.28), $\mathbf{a}$ can be written as

$$\mathbf{a} = \Big(0, \quad 1, \quad 0, \quad 0, \quad 0, \quad 0\Big)^T \tag{2.31}$$

and for (2.29), $\mathbf{a}$ can be written as

$$\mathbf{a} = \Big(1, \quad 0, \quad -1, \quad 0, \quad 0, \quad 0\Big)^T. \tag{2.32}$$

By stacking $\mathbf{a}^T$ from n of these constraints in a matrix, A, of size $n \times 6$ and solving $A\mathbf{w} = \mathbf{0}$ using Singular Value Decomposition (SVD) [24, pp. 593], gives a least-squares solution for $\boldsymbol{\omega}$. $\boldsymbol{\omega}$ can be decomposed into the intrinsic camera matrix, K, using matrix inversion and Cholesky factorisation [24, pp. 582]. A minimum of five independent constraints are required, which can be achieved by using multiple constraints from (2.26).

This calibration assumes that the scene is projected to the image plane linearly. If this is not the case, the image has to be corrected as in (2.16). To do this the coefficients of $\mathrm{L}(r)$ and the centre of radial distortion have to be determined. This is usually done by an iterative minimisation of a cost function on the projection of a known pattern, e.g. a chessboard pattern or a Tsai grid [24, pp. 192]. Complete algorithms are proposed by Heikkilä and Silvén in 1997 [26] and Zhang in 2000 [27].

There are different tools for camera calibration available. Some of the most popular ones [28] are the Camera Calibration Toolbox for Matlab[1] and tools available in the OpenCV library [29].

[1] `http://www.vision.caltech.edu/bouguetj/calib_doc/`

2.2 Visual Odometry

VO is the process of estimating the ego-motion based on consecutive camera images. The classical approach is to detect distinctive projected world points, the points between images, and use the pixel coordinates of the points to determine the ego-motion. These methods are usually called feature based methods.

2.2.1 Feature based methods

To relate the pose of two camera views, epipolar geometry can be used. To denote camera matrices, projected points and other quantities related to the second view, $'$ will be used. All cameras are assumed to project world points onto the image plane linearly.

Epipolar geometry

Epipolar geometry is independent of the scene structure, and only depends on the internal and external parameters of the two cameras, i.e. the camera matrix from (2.12). A world point, $\mathbf{X}$, is projected onto the image plane of two views at $\mathbf{x}$ and $\mathbf{x}'$. The two camera centres, $\mathbf{C}$ and $\mathbf{C}'$, the world point and the projected points will be coplanar [24, pp. 239], as can be seen in figure 2.5. Call this plane, $\boldsymbol{\pi}$, the epipolar plane. Knowing that the projected points have to be on the epipolar plane, the search for a matching point for $\mathbf{x}$ in the second view, $\mathbf{C}'$, is limited to the line where the image plane intersects the epipolar plane, the epipolar line, as is shown in figure 2.5. The epipoles, $\mathbf{e}$ and $\mathbf{e}'$, are the points where the baseline, the line between the two camera centres, intersects each image plane.

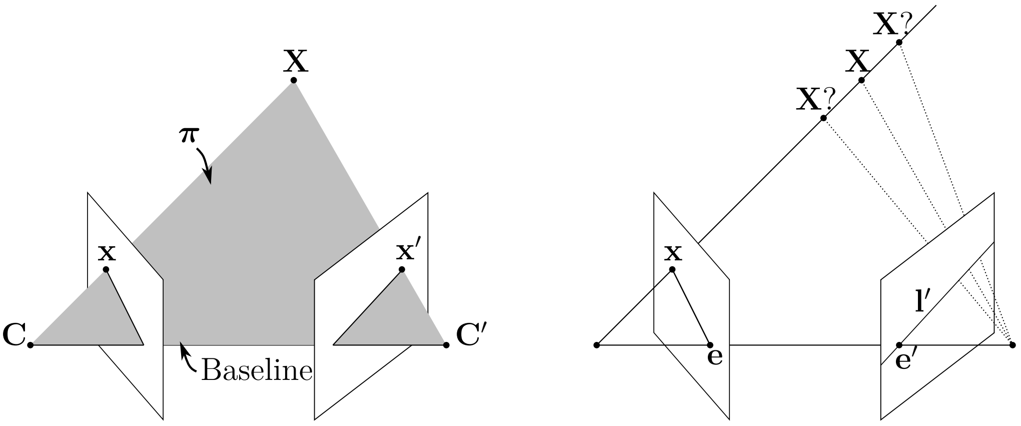


Figure 2.5: The left figure is showing the two camera view centres, $\mathbf{C}$ and $\mathbf{C}'$, the world point, $\mathbf{X}$, the projected image points, $\mathbf{x}$ and $\mathbf{x}'$, and the epipolar plane $\boldsymbol{\pi}$. The right figure shows the epipolar points, $\mathbf{e}$ and $\mathbf{e}'$, the line of the second view, $\mathbf{l}'$, and how all world points in the direction of $\mathbf{X}$ must be projected onto this line.

The fundamental matrix, F, is a central part of epipolar geometry. The fundamental matrix relates the two views as $\mathbf{x}'^T F\mathbf{x} = 0$, where $\mathbf{x}$ and $\mathbf{x}'$ is the projection of a world point, $\mathbf{X}$, in the first and second view, respectively. The fundamental matrix is a 3×3 matrix of rank 2 [24, pp. 239]. From the fundamental matrix the relative pose between two views can be calculated, up to a scale. The fundamental matrix is independent of scene structure, but can be calculated from scene points [24, pp. 239].

In a pair of images, any world point, $\mathbf{X}$, projected onto one of the images, $\mathbf{x}$, the matching point, $\mathbf{x}'$, on the other image has to be on the corresponding epipolar line, $\mathbf{l}'$. Thus there is a mapping $\mathbf{x} \mapsto \mathbf{l}'$ [24, pp. 242].

According to (2.7) the world point, $\mathbf{X}$, is projected to an image point, $\mathbf{x}$, as $\mathbf{x} = P\mathbf{X}$. The ray, back-projected from $\mathbf{x}$ to $\mathbf{X}$, can be written as [24, pp. 162]

$$\mathbf{X}(\lambda) = P^{+}\mathbf{x} + \lambda\mathbf{C}, \tag{2.33}$$

where P^{+} the pseudo-inverse of P, $\mathbf{C}$ is the null-vector of P and also the camera centre, defined by $P\mathbf{C} = 0$, and λ is a parametrisation of the ray. The camera centre is back projected onto the line at $\mathbf{C}$, when $\lambda \to \infty$, and as the world point, $P^{+}\mathbf{x}$, when $\lambda = 0$. These two points are projected in the second view as $P'\mathbf{C}$ and $P'P^{+}\mathbf{x}$. These two points lie on the epipolar line of the second view, $\mathbf{l}'$, thus $\mathbf{l}' = (P'\mathbf{C}) \times (P'P^{+}\mathbf{x})$. As the point $P'\mathbf{C}$ is the epipole of the second view, $\mathbf{e}'$, this can be written as $\mathbf{l}' = [\mathbf{e}']_{\times}(P'P^{+})\mathbf{x} = F\mathbf{x}$, which gives

$$F = [\mathbf{e}']_{\times}(P'P^{+}), \tag{2.34}$$

and the map $\mathbf{x} \mapsto \mathbf{l}'$

$$\mathbf{l}' = F\mathbf{x}. \tag{2.35}$$

$[\mathbf{e}']_{\times}$ denotes the skew-symmetric of $\mathbf{e}'$.

The skew-symmetric matrix of a vector $\mathbf{a} = (a_1, a_2, a_3)^T$ is defined as [24, pp. 581]

$$[\mathbf{a}]_{\times} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}, \tag{2.36}$$

which relates to the cross product as

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_{\times}\mathbf{b}. \tag{2.37}$$

To find a method for calculating F from image coordinates alone, consider two cameras in the coordinate system of the first camera, $P = K[I|\mathbf{0}]$ and $P' = K'[R|\mathbf{t}]$, then [24, pp. 244]

$$P^{+} = \begin{bmatrix} K^{-1} \\ \mathbf{0}^T \end{bmatrix} \quad \mathbf{C} = \begin{pmatrix} \mathbf{0} \\ 1 \end{pmatrix} \tag{2.38}$$

and, as the epipole of the second camera, $\mathbf{e}'$, is the projection of the first camera centre, $P'\mathbf{C}$,

$$\begin{aligned} F &= [P'\mathbf{C}]_{\times}P'P^{+} = [K'\mathbf{t}]_{\times}K'RK^{-1} = K'^{-T}[\mathbf{t}]_{\times}RK^{-1} = K'^{-T}R[R^T\mathbf{t}]_{\times}K^{-1} \\ &= K'^{-T}RK^T[KR^T\mathbf{t}]_{\times}. \end{aligned} \tag{2.39}$$

Then, as [24, pp. 244]

$$\mathbf{e} = P\begin{pmatrix}-R^T\mathbf{t}\\1\end{pmatrix} = KR^T\mathbf{t} \quad \text{and} \quad \mathbf{e}' = P'\begin{pmatrix}\mathbf{0}\\1\end{pmatrix} = K'\mathbf{t}, \tag{2.40}$$

$$\begin{aligned}F &= [\mathbf{e}']_\times K'RK^{-1} = K'^{-T}[\mathbf{t}]_\times RK^{-1} = K'^{-T}R[R^T\mathbf{t}]_\times K^{-1}\\ &= K'^{-T}RK^T[\mathbf{e}]_\times.\end{aligned} \tag{2.41}$$

If $\mathbf{x}$ and $\mathbf{x}'$ correspond, $\mathbf{x}'$ will lie on the epipolar line $\mathbf{l}' = F\mathbf{x}$, thus $\mathbf{x}'^T\mathbf{l}' = 0$ [24, pp. 245], which gives

$$\mathbf{x}'^T F\mathbf{x} = 0. \tag{2.42}$$

Writing $\mathbf{x} = (x, y, 1)^T$ and $\mathbf{x}' = (x', y', 1)^T$ and denoting f_{nm} as the elements of F, (2.42) gives rise to the linear equation

$$x'xf_{11} + x'yf_{12} + x'f_{13} + y'xf_{21} + y'yf_{22} + y'f_{23} + xf_{31} + yf_{32} + f_{33} = 0, \tag{2.43}$$

which can be expressed by a vector multiplication as

$$\begin{pmatrix}x'x, & x'y, & x', & y'x, & y'y, & y', & x, & y, & 1\end{pmatrix}\mathbf{f} = 0, \tag{2.44}$$

where $\mathbf{f}$ is a 9-vector of the elements in F in row-major order. From a set of n points, a set of linear equations can be written as

$$A\mathbf{f} = \begin{bmatrix}x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1\end{bmatrix}\mathbf{f} = \mathbf{0}. \tag{2.45}$$

For an exact solution to exist for this, A has to be of rank 8, thus a minimum of 8 points are required to solve this equation. Since there is noise in the image coordinates from quantisation errors and other sources, more points can be used to find a better estimate of $\mathbf{f}$. In this case A may be of rank 9. A solution can be found by SVD, $A = UDV^T$, where the last column of V will be the singular vector, corresponding to the smallest singular value of A, and also the least-squares solution of $\mathbf{f}$, which minimises $\|A\mathbf{f}\|$, subject to $\|\mathbf{f}\| = 1$ [24, pp. 280]. The solution for $\mathbf{f}$ can then be decomposed to F.

The fundamental matrix is a singular matrix of rank 2, and many applications of it relies on that fact [24, pp. 280]. Solving the linear equations of from (2.45) does not guarantee this, and steps to enforce this should be performed. The fundamental matrix obtained from the linear equations can be modified to satisfy this by replacing F with F' that minimises the Frobenius norm $\|F - F'\|$ subject to $\det F' = 0$. This can be obtained by the SVD of $F = UDV^T$, where D is a 3×3 diagonal matrix with the singular values of F in descending order on the diagonal. Replacing the smallest singular value in D with 0, calling it D', and constructing $F' = UD'V^T$, then F' will minimise the Frobenius norm $\|F - F'\|$ [24, pp. 281].

Image coordinates can come in different formats, some define the upper left corner of the image as the origin and others define the centre of the image as the origin. It can be shown that this, as well as the magnitude of the pixel coordinates, influence the result when retrieving the fundamental matrix using the 8-point algorithm described above [24, pp. 281]. This is due to the difference in magnitude of the elements of A which are two elements multiplied, like $x'x$ and $x'y$, compared to single elements, like x and y. The condition number of A will be higher when the magnitude of the elements differs more, which will lead to bigger errors when enforcing rank 2 on the fundamental matrix [30]. Therefore it is considered an essential step to normalise the image points [24, pp. 281]. A suggested normalisation $\hat{\mathbf{x}}_i = T\mathbf{x}_i$ and $\hat{\mathbf{x}}'_i = T'\mathbf{x}'_i$ is to construct the transformation T and T' such that it will place the points with the centroid at the origin and with the Root Mean Square (RMS) distance equal to $\sqrt{2}$ [24, pp. 282]. Then perform the steps of the 8-point algorithm with the normalised points and retrieve the normalised fundamental matrix $\hat{F}'$. Finally retrieve the fundamental matrix corresponding to the original data as $F = T'^T\hat{F}'T$ [24, pp. 282]. This is called the normalised 8-point algorithm for F.

A specialisation of the fundamental matrix is the essential matrix, E. The fundamental matrix can be seen as a generalisation of the essential matrix, where the assumption of a calibrated camera is removed.

Consider a camera matrix $P = K[R|\mathbf{t}]$, and a world point projected to an image point as $\mathbf{x} = P\mathbf{X}$. If the intrinsic matrix, K, is known, its inverse can be applied to the image point which gives $\hat{\mathbf{x}} = K^{-1}\mathbf{x}$. This gives $\hat{\mathbf{x}} = [R|\mathbf{t}]\mathbf{X}$, where $\hat{\mathbf{x}}$ is the image point in normalised coordinates [24, pp. 257]. The camera matrix $K^{-1}P = [R|\mathbf{t}]$ is called the normalised camera matrix [24, pp. 257]. A pair of normalised camera matrices would then, according to (2.39), result in an essential matrix as

$$E = [\mathbf{t}]_\times R = R[R^T\mathbf{t}]_\times. \tag{2.46}$$

The essential matrix is defined as

$$\hat{\mathbf{x}}'^T E\hat{\mathbf{x}} = 0. \tag{2.47}$$

Substituting $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}'$ with the corresponding points $\mathbf{x} \leftrightarrow \mathbf{x}'$ gives $\mathbf{x}'^T K'^{-T}EK^{-1}\mathbf{x} = 0$, which compared to $\mathbf{x}'^T F\mathbf{x} = 0$ gives

$$E = K'^T FK. \tag{2.48}$$

Both the translation, $\mathbf{t}$, and the rotation, R, in the essential matrix $E = [\mathbf{t}]_\times R$ has 3 DoF, however, the essential matrix has only five due to a scale ambiguity.

Nistér in 2004 proposed an efficient solution to determine the essential matrix with five correspondences [31], which is the minimal set to fully solve the 5 DoF essential matrix.

It can be shown that the essential matrix is a matrix where two of its singular values are equal and the third is equal to zero [24, pp. 258]. The two equal singular values can be chosen freely, due to the scale ambiguity, and are often set to 1. It can also be shown that for a given essential matrix, with the SVD $E = U\text{diag}(1, 1, 0)V^T$, and a first camera matrix $P = [I|\mathbf{0}]$, there are four possible solutions for the second camera matrix [24, pp. 259]:

$$\begin{aligned} P_1' &= \left[UWV^T \mid \mathbf{u}_3\right] \quad P_2' = \left[UWV^T \mid -\mathbf{u}_3\right] \\ P_3' &= \left[UW^TV^T \mid \mathbf{u}_3\right] \quad P_4' = \left[UW^TV^T \mid -\mathbf{u}_3\right], \end{aligned} \tag{2.49}$$

where

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.50}$$

Of the four different solutions, only one is correct. The different solutions can be interpreted as is shown in figure 2.6. To determine which of the solutions is correct, the respective translation and rotation can be used to triangulate the positions of the points used to determine the essential matrix. The solution which renders the points in front of both cameras is the correct one, top left in figure 2.6.

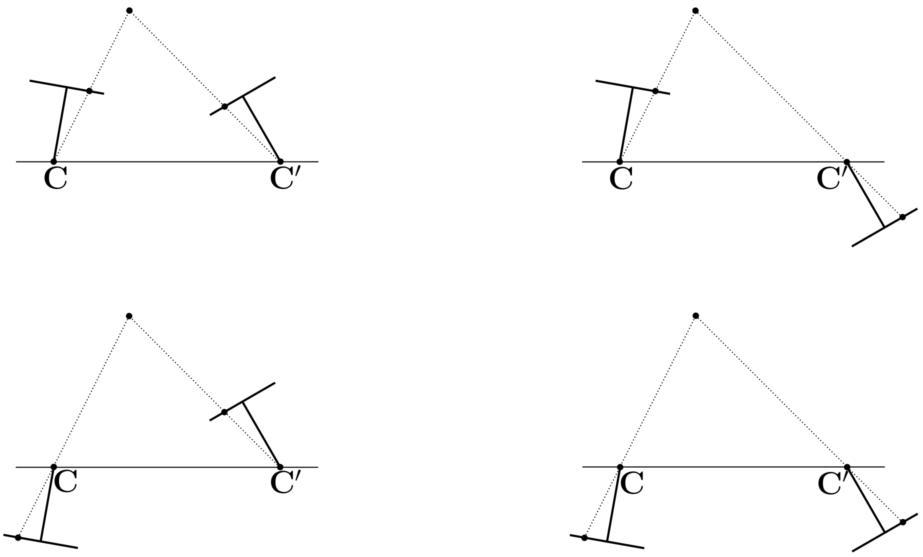


Figure 2.6: Illustration of how the four different choices of camera matrices for a given essential matrix can be interpreted.

A world point is projected through two cameras onto their image planes as $\mathbf{x} = P\mathbf{X}$ and $\mathbf{x}' = P'\mathbf{X}$. These equations can be combined into a form $A\mathbf{X} = \mathbf{0}$. This form can be obtained by the cross product $\mathbf{x} \times (P\mathbf{X}) = \mathbf{0}$, which written out gives [24, pp. 312]

$$\begin{aligned} x(\mathbf{p}^{3T}\mathbf{X}) - (\mathbf{p}^{3T}\mathbf{X}) &= 0 \\ y(\mathbf{p}^{3T}\mathbf{X}) - (\mathbf{p}^{2T}\mathbf{X}) &= 0 \\ x(\mathbf{p}^{2T}\mathbf{X}) - y(\mathbf{p}^{1T}\mathbf{X}) &= 0, \end{aligned} \tag{2.51}$$

where $\mathbf{p}^{iT}$ denotes the i:th row of P. Combined with the analogue for $\mathbf{x}' \times (P'\mathbf{X}) = \mathbf{0}$, this can be written in the form $A\mathbf{X} = \mathbf{0}$ with

$$A = \begin{bmatrix} x\mathbf{p}^{3T} - \mathbf{p}^{1T} \\ y\mathbf{p}^{3T} - \mathbf{p}^{2T} \\ x'\mathbf{p}'^{3T} - \mathbf{p}'^{1T} \\ y'\mathbf{p}'^{3T} - \mathbf{p}'^{2T} \end{bmatrix}. \tag{2.52}$$

This can be solved using SVD, $A = UDV^T$, where the last column of V will correspond to the least-squares solution of $\mathbf{X}$ [24, pp. 312].

A more efficient algorithm for obtaining $\mathbf{t}$ and R is proposed by Nistér in 2004 [31].

Feature detection and description

To use the methods described, points in the image planes of successive images that are projections of the same point on a physical object must be identified. There are two different, generally used approaches to find image points corresponding to the same world point: feature matching and feature tracking.

Feature matching is to independently detect interesting feature points in two images, construct a feature descriptor for each image point, which should describe the points in a unique way, but still be invariant for different views, and then match the feature points between the two images using their descriptors.

The feature detectors can be divided into two groups: corner detectors and blob detectors. A corner is defined as a point where two or more edges intersect [32]. A blob is a pattern in the image which differs from its immediate neighbourhood in terms of intensity, colour and texture, and is not an edge nor a corner [32]. Desired properties for feature detectors are localisation accuracy, the pixel coordinate of where the feature is detected should be precise; repeatable, the same feature should be detected in different images; computationally efficient; robust to noise, compression artifacts and blur; distinctive, to be able to match the feature accurately across different images, and invariance to both photometric and geometric changes [32]. Corner detectors are in general faster to compute and give a more accurate image position, but finds less distinctive features than blob detectors. Blob detectors are usually less invariant to changes in scale and viewpoint [32]. The choice of corner detector depends on computational constraints, environment type and motion baseline, and should be carefully considered [32].

There are many feature detectors available. Commonly used corner detectors are Harris [33], Features from Accelerated Segment Test (FAST) [34] and Shi-Tomasi [35] and blob detectors are Scale Invariant Feature Transform (SIFT) [36], Speeded Up Robust Features (SURF) [37] and Center Surround Extrema (CENSURE) [38]. Feature detectors usually work in two steps. The first step is to apply a feature-response function on the entire image [32]. For instance, Harris uses the corner response function and SIFT uses the Difference-Of-Gaussian (DoG) operator [32]. The second step is to localise all local extrema points on the output of the first step,

	Corner Detector	Blob Detector	Rotation Invariant	Scale Invariant	Affine Invariant[1]	Repeatability	Localisation Accuracy	Robustness	Efficiency
Harris	×		×			+++	+++	++	++
Shi-Tomasi	×		×			+++	+++	++	++
FAST	×		×	×		++	++	++	++++
SIFT		×	×	×	×	+++	++	+++	+
SURF		×	×	×	×	+++	++	++	++
CENSURE		×	×	×	×	+++	++	+++	+++

Table 2.1: Properties and performance of feature detectors [32].

[1] Not truly affine invariant, but found to be invariant up to certain changes in viewpoint.

by applying nonmaxima suppression [32]. To achieve invariance to scale, the feature detector is often applied to images of different scale, however this will lead to loss of accuracy in the pixel coordinate due to the lower resolution and is computationally heavy [38]. In CENSURE, another approach, where the Laplacian across scale feature-response is used, which calculates the features at every pixel at all scales, and achieves real-time performance [38]. A summary of properties and performance of different feature detectors is given in table 2.1.

For each detected feature point a compact descriptor based on the region around each point is calculated. The simplest feature descriptor is a descriptor of the appearance of the point, i.e. the intensity of each pixel in a region around the feature point. Then the sum of squared differences or the normalised cross correlation can be used to compare the descriptors [32]. However, these descriptors are not very robust to changes in orientation, scale and viewpoint [32]. More elaborate descriptors are the SIFT descriptor [36], the SURF descriptor [37], Binary Robust Independent Elementary Features (BRIEF) [39], Oriented FAST and Rotated BRIEF (ORB) [40] and Binary Robust Invariant Scalable Keypoints (BRISK) [41]. In common for these descriptors is that they generate a vector, usually 64 or 128 elements long. SIFT and SURF produce vectors of real numbers, while, as their name indicate, BRISK, ORB and BRIEF produce binary vectors.

The SIFT descriptor algorithm works by dividing the region around the feature point into a 4×4 grid, for which each of eight histograms of gradients with different orientations are calculated and concatenated into the 128 element descriptor vector [32]. It is desired for a descriptor to be invariant to rotation. In SIFT, this achieved by calculating the image gradient orientation of a region around the feature point and rotating the calculation of histograms accordingly [36].

The set of features from two images can then be exhaustively matched, using a similarity measure on the feature descriptors. For SIFT and SURF the Euclidean distance can be used [32], and for BRIEF, ORB and BRISK the Hamming distance can be used [39][40][41]. The complexity of exhaustive matching is $\mathcal{O}(n^2)$, and becomes impractical when the number features becomes large [32]. Binary descriptors easily outperforms floating point descriptors in speed of matching, as the Hamming distance of a binary vector can be calculated extremely fast on a modern CPU in the form of a bitwise XOR operation [41]. The matching process can also be speeded up by using an indexing structure, such as a search tree or a hash table [32]. If there is a known motion model where other sensors, like an IMU, can be used to estimate a motion, given the first image point, the epipolar line of the second image, where the matching point should recur, can be calculated as shown in section 2.1.2, and the search for a matching feature point can be limited [32].

Feature tracking is a different approach to find matching points. The tracker is initialised with a set of feature points, which are to be tracked. Often the Kanade–Lucas–Tomasi (KLT) [35] feature tracker is used [32]. These methods require that the images are taken in close proximity [32]. The KLT tracker estimates the displacement of a point by estimating an affine image transformation of regions in the image around a feature point, feature windows. The affine image motion assumes a point, $\mathbf{x} = (x, y)^T$, in the first image, I_1, will move to a point, $A\mathbf{x} + \mathbf{d}$, in the second image I_2, and that

$$I_2(A\mathbf{x} + \mathbf{d}) = I_1(\mathbf{x}), \tag{2.53}$$

where $\mathbf{d}$ is the feature windows displacement, $A = I + D$, and where

$$D = \begin{bmatrix} d_{xx} & d_{xy} \\ d_{yx} & d_{yy} \end{bmatrix}. \tag{2.54}$$

The displacement, $\mathbf{d}$, and the deformation matrix, D, are then found by minimising [35]

$$\epsilon = \iint_W [I_2(A\mathbf{x} + \mathbf{d}) - I_1(\mathbf{x})]^2 w(\mathbf{x})\, \mathrm{d}\mathbf{x}, \tag{2.55}$$

where W is the feature window and $w(\mathbf{x})$ is a weighting function. The weighting function can e.g. be equal to 1 or a Gaussian-like function, to emphasize the central area of the feature window [35]. The integral is linearised using truncated Taylor expansion and then minimised using the Newton-Raphson method [35].

RANSAC

Both feature matching and feature tracking will result in incorrect point correspondences between images, outliers, which will bias the result if included in estimation of the motion. Random Sample Consensus (RANSAC) [42] has been established as the standard method for outlier rejection [32]. The idea behind RANSAC is to estimate a number of hypothesis models by repeatedly sampling a randomly selected minimum set of correspondences and count the total number of correspondences in consensus with the estimated hypothesis [32]. The matching correspondences of the hypothesis generating the most matches will be selected as inliers. An example of

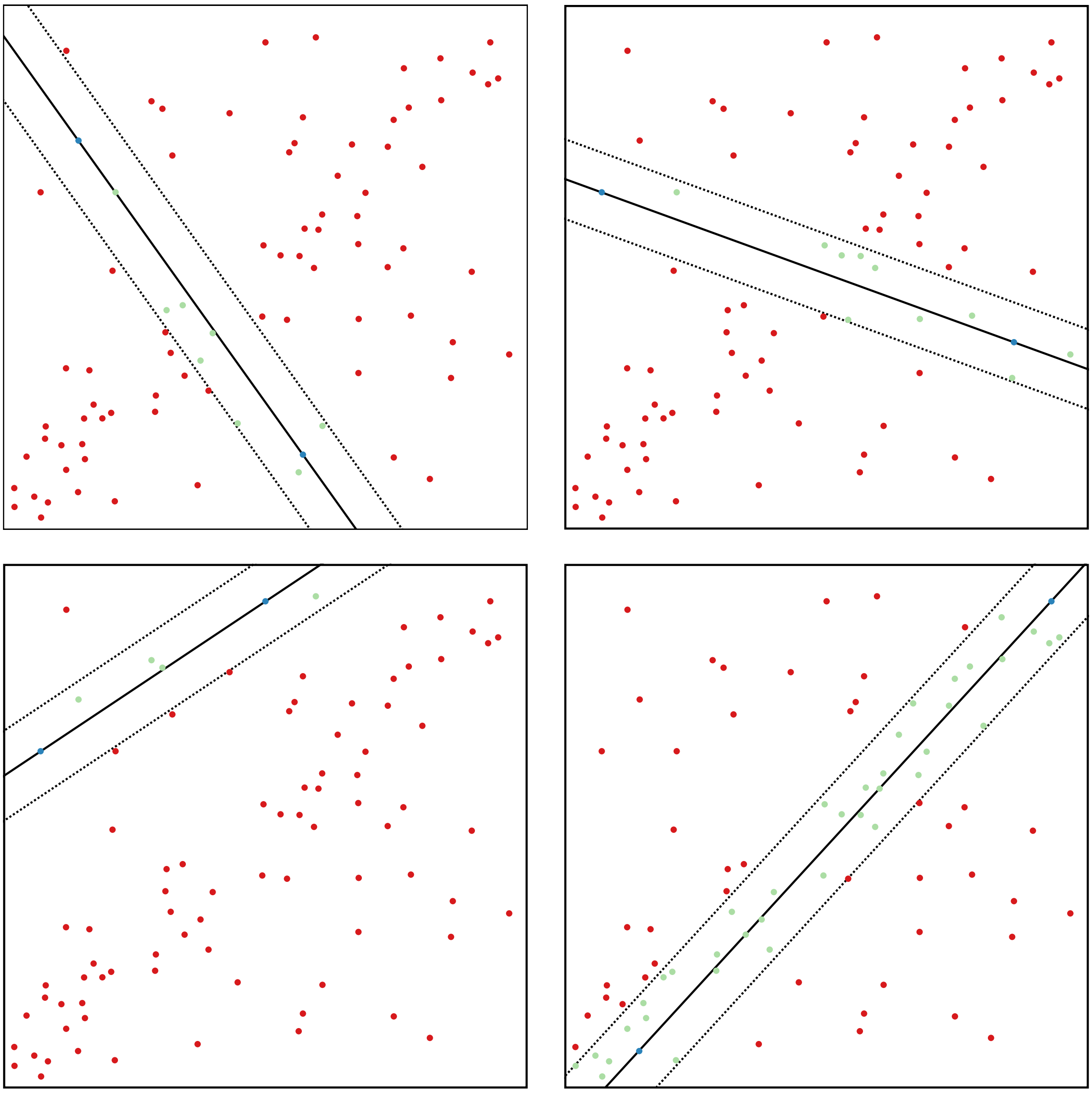


Figure 2.7: Illustration of how RANSAC would work to estimate a line. The figures shows two randomly selected points in blue, rejected outliers in red and the inliers in green for four different hypothesises. Here the inliers of the hypothesis in the bottom right image will be selected as the final inliers as they are more than in the other hypothesises.

how the RANSAC sampling process, to estimate a line in a plane, could look like is illustrated in figure 2.7.

The number of samples required for finding a correct hypothesis, N, with a probability of success, P, is determined by the percentage of outliers, ϵ, and the number points in the minimal set, s [32]:

$$N = \frac{\log\left(1 - P\right)}{\log\left(1 - (1 - \epsilon)^{s}\right)}. \tag{2.56}$$

As can be seen in figure 2.8, the number of required samples increases exponentially in the number of minimal points in the set, thus it is desired for the minimum set required to make a hypothesis to be as small as possible.

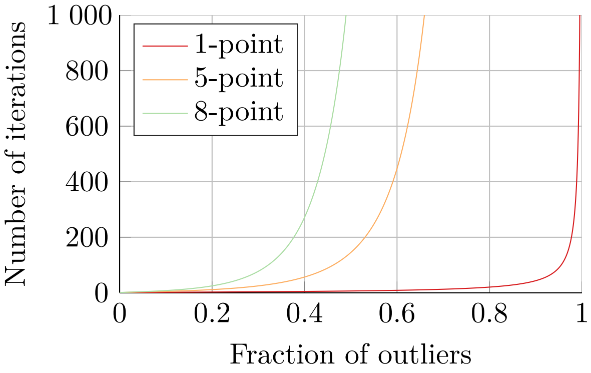


Figure 2.8: A plot of the number of required iterations of RANSAC for a 99 % probability of success at different percentage of outliers with different number of minimal points.

RANSAC can be used with the normalised 8-point algorithm described previously, with 8 points in the minimal set. To calculate a distance to the remaining points, the distance to the epipolar line can be used [32]. Often the Sampson distance [43], which is a first order approximation of the geometric distance, is used, as the geometric error is quite complex in nature [24, pp. 98][32]. The Sampson approximation of a geometric distance is $\epsilon^T(JJ^T)^{-1}\epsilon$, where ϵ is the Sampson cost and J the Jacobian of the Sampson cost. The Sampson approximation of the epipolar line becomes [24, pp. 287]

$$\frac{(\mathbf{x}'^T F\mathbf{x})^2}{JJ^T}. \tag{2.57}$$

From the explicit expression of $\mathbf{x}'^T F\mathbf{x}$ in (2.43), and the definition of J [24, pp. 287],

$$JJ^T = (F\mathbf{x})_1^2 + (F\mathbf{x})_2^2 + (F^T\mathbf{x}')_1^2 + (F^T\mathbf{x}')_2^2, \tag{2.58}$$

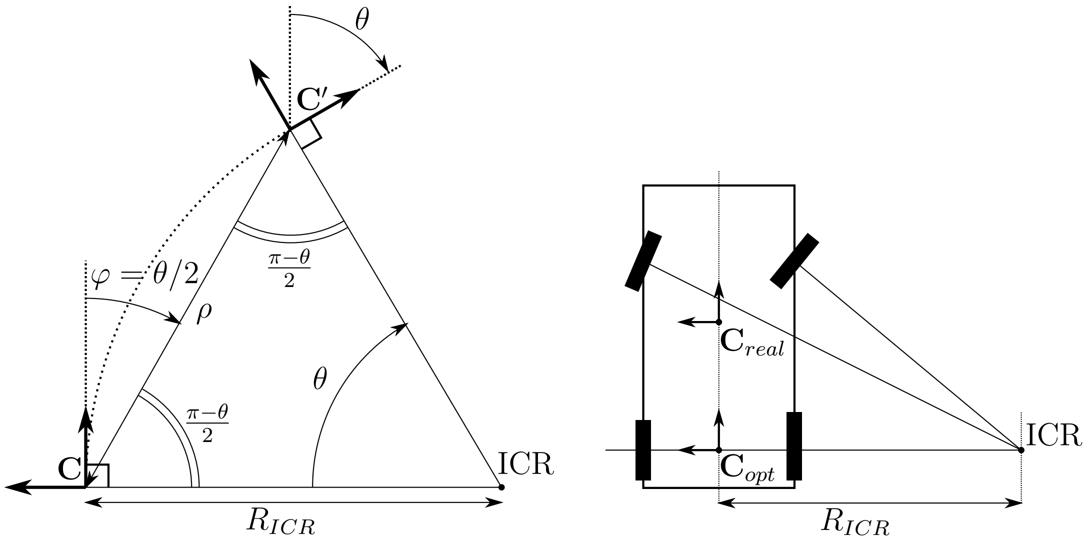
where $(F\mathbf{x})_j$ represents the jth element of the vector $F\mathbf{x}$. Thus, the Sampson cost function for a point correspondence distance to the epipolar line, given a fundamental matrix is

$$\frac{(\mathbf{x}'^T F\mathbf{x})^2}{(F\mathbf{x})_1^2 + (F\mathbf{x})_2^2 + (F^T\mathbf{x}')_1^2 + (F^T\mathbf{x}')_2^2}. \tag{2.59}$$

As it is desired for the minimal set of corresponding points to be as small as possible, to reduce the number of required RANSAC iterations, algorithms requiring less correspondences than the normalised 8-point algorithm, like the highly efficient 5-point algorithm proposed by Nistér in 2004, could be used. Using the 5-point algorithm, which requires 5 correspondences, compared to the normalised 8-point algorithm, which requires 8, would reduce the number of required iterations from 1177 to 145, with a 99 % success rate and 50 % outliers, according to (2.56).

In 2009 Scaramuzza *et al.* proposed a solution for RANSAC for on-road vehicles, only requiring 1 point [44]. With 1 correspondence, according to (2.56), the number of required RANSAC iterations with a 99 % success rate and 50 % outliers is 7. The algorithm requires only 1 correspondence, as a restricted motion model, assuming circular motion, is used [44]. As can be seen in figure 2.9a, a planar motion can be described by three parameters, the yaw angle θ and the polar coordinates $(\rho, \varphi)^T$

[44]. Since, when using a monocular camera, the scale factor is unknown, ρ can be set to 1 [44]. Assuming the camera to move along a circumference, perpendicular to the Instantaneous Centre of Rotation (ICR), as in figure 2.9a, gives $\varphi = \theta/2$, thus, only θ has to be estimated, which can be done with a 1 point correspondence [44]. The Ackermann steering configuration, which is common in automobiles [45, pp. 37], ensures an ICR will exist, located in the center of the rear axle [45, pp. 68]. Thus, the above assumptions made about the motion in the 1 point RANSAC are reasonable for cars, if the camera is placed as $\mathbf{C}_{opt}$ in figure 2.9b [44]. However, in practice, the steering angle of cars is small and thus the radius of the curvature is big, which allows relaxation of constraint on the placement of the camera to the principal axis of the camera being perpendicular to the rear-wheel axle [44].



(a) Description of circular motion.

(b) The Ackermann steering principle.

Figure 2.9: Illustrations of how a circular motion can be used describe the motion of a vehicle using the Ackermann steering principle.

The circular motion can be described by a rotational matrix R and a translation matrix $\mathbf{t}$ as

$$R = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad \mathbf{t} = \rho \cdot \begin{pmatrix} \sin\varphi \\ 0 \\ \cos\varphi \end{pmatrix}, \tag{2.60}$$

if rotation is about the y-axis and translation in the xz plane is considered. It is known from (2.47) that the essential matrix, E, relates two corresponding image points, $\mathbf{x} = (x, y, 1)^T$ and $\mathbf{x}' = (x', y', 1)^T$, as $\mathbf{x}'E\mathbf{x} = 0$, and from (2.46), that $E = [\mathbf{t}]_\times R$. Using (2.60), (2.46), and the constraint $\varphi = \theta/2$, an expression for the essential matrix can be obtained:

$$E = \rho \cdot \begin{bmatrix} 0 & -\cos\frac{\theta}{2} & 0 \\ \cos\frac{\theta}{2} & 0 & \sin\frac{\theta}{2} \\ 0 & \sin\frac{\theta}{2} & 0 \end{bmatrix}. \tag{2.61}$$

Inserting this into (2.47) gives

$$\sin\frac{\theta}{2}(y'+y)+\cos\frac{\theta}{2}(y'x-x'y)=0, \tag{2.62}$$

and the rotation angle can then be obtained:

$$\theta=-2\arctan\left(\frac{y'x-x'y}{y'+y}\right). \tag{2.63}$$

Bundle Adjustment

Bundle Adjustment (BA) is a method for minimising the reprojection error of features detected in an image by adjusting the camera pose estimate and feature 3D position estimates [24, pp. 434]. Consider a set of camera poses, described by camera matrices, P_i, and a set of 3D feature points, $\mathbf{X}_j$, detected at image coordinates, $\mathbf{x}_{i,j}$. The reprojection error is then the distance between the detected feature point in the image, $\mathbf{x}_{i,j}$, and the image coordinates of the feature reprojected onto the image given the camera pose and the 3D feature position, $P_i\mathbf{X}_j$. Since this generally can not be solved exactly due to noisy measurements, the objective is to find estimates of the camera matrices and feature point positions, $\hat{P}_i$ and $\hat{\mathbf{X}}_j$, respectively, that minimise the total reprojection error. Assuming Gaussian noise on the measurements the maximum likelihood, estimates can be found by minimising the sum of the squared reprojection errors:

$$\min_{\hat{P}_i,\hat{\mathbf{X}}_j}\sum_{i,j\in C} d(\hat{P}_i\hat{\mathbf{X}}_j,\mathbf{x}_{i,j})^2, \tag{2.64}$$

where $d(\mathbf{a},\mathbf{b})$ is the geometric distance in the image between homogeneous image points, $\mathbf{a}$ and $\mathbf{b}$, and C is the set of pairs i,j, for which feature $\mathbf{X}_j$ has been seen from pose i, i.e. $\mathbf{x}_{i,j}$ exists. BA needs a good initialisation [24, pp. 435]. This estimate can be provided by triangulating new features when they have been seen in two consecutive frames whose poses are estimated using epipolar geometry, as detailed above.

It has been shown that BA is the optimal non-linear least-squares SLAM algorithm [46], however, the complexity of minimising (2.64) with the Levenberg-Marquardt algorithm is cubic in the number of poses and features [46][24, pp. 435], which makes in impractical to optimised the full graph online. A way of reaping the benefits of the optimality of BA, while maintaining constant complexity, making it possible to use BA for VO, is to partition the sequence of images by windowing the latest n frames, and only performing BA for the corresponding poses and the features in those images [24, pp. 453]. This is called local BA.

2.2.2 Direct methods

Direct methods optimises the geometry directly on the image intensities, which enables the possibility of using all information in the image [23]. By this, direct methods circumvents the limitation of feature based methods, that “only information that conforms to the feature type can be used” [23]. Direct methods have higher accuracy and robustness, compared to feature based methods, in particular in areas with few feature points, and in addition provides more information about the geometry of the environment [23].

Direct methods for visual odometry has been well established for RGB-D and stereo cameras [23], such as the solution by Klose *et al.* in 2013 [47], but solutions have been proposed for monocular cameras by Pizzoli *et al.* in 2014 [48] and Stühmer *et al.* in 2010 [49]. All these solutions are computationally demanding and requires a state-of-the-art Graphics Processing Unit (GPU) to run in real-time [23]. In 2013 Engel *et al.* proposed a semi-dense solution for direct visual odometry for a monocular camera [13], which significantly reduces the computational complexity, compared to previous work, but does not build a globally consistent map including loop closures [23]. The approach of the method is to spend computations where the information gain is maximised [13]. This is done by by calculating a semi-dense inverse depth map only for the regions of the image with non-negligible gradient [13]. This solution for visual odometry was incorporated in a full solution for the SLAM problem by Engel *et al.* in 2014, which corrects accumulated scale drift and closes loops [23].

2.3 Simultaneous Localisation And Mapping

The SLAM problem consists of building a map and localising within that map simultaneously. There are several kinds of maps that can be used to represent the surroundings. Maps consisting of the locations of discrete point-like landmarks, i.e. point clouds, are well suited to be used with camera data [50]. In 2001, Dissanayake *et al.* showed that it is possible to build accurate maps with only relative measurements, assuming Gaussian noise, linear models and static landmarks [51]. More specifically they proved that the only lower limit on the variance in the estimates of the pose and landmarks is the variance of the initial pose.

The SLAM problem can be formulated as calculating, or approximating, the probability distribution

$$p(x_{1:t}, m | z_{1:t}, u_{1:t}, x_0), \tag{2.65}$$

where $x_{1:t}$ is the trajectory of poses x_1 to x_t, m is the map, i.e. the position of the observed landmarks, $z_{1:t}$ are all measurements, $u_{1:t}$ are all the motion commands issued and x_0 is the initial pose [9, pp. 310] [52]. Using a motion model appropriate for the given system, the motion commands can be translated into an odometry and u is indeed sometimes referred to as odometry. The initial position is usually considered to be arbitrary, x_0 can be set to the origin with zero uncertainty.

An equally important variant is the online SLAM problem of finding

$$p(x_t, m | z_{1:t}, u_{1:t}, x_0), \tag{2.66}$$

where only the current pose is sought together with the map, given all measurements and motion commands [9, pp. 309]. Factorising the right-hand side of (2.66) by applying Bayes theorem, omitting x_0 for brevity, gives

$$p(x_t, m|z_{1:t}, u_{1:t}) = \frac{p(x_t, m|z_{1:t-1}, u_{1:t})p(z_t|x_t, m)}{p(z_t|z_{1:t-1}, u_{1:t})}. \tag{2.67}$$

The observation model, $p(z_t|x_t, m)$, is the probability of observing z_t at time t given a location x_t and a map. $p(x_t, m|z_{1:t-1}, u_{1:t})$ is the motion prediction, given the new controls, which can be written as

$$p(x_t, m|z_{1:t-1}, u_{1:t}) = \int p(x_{t-1}, m|z_{1:t-1}, u_{1:t-1})p(x_t|x_{t-1}, u_t)\, dx_{t-1}. \tag{2.68}$$

This clearly factors the prediction into the motion model, $p(x_t|x_{t-1}u_t)$, and the previous posterior, $p(x_{t-1}, m|z_{1:t-1}, u_{1:t-1})$.

The main difference between VO using BA and V-SLAM is that V-SLAM in general tries to create a globally consistent map, as opposed to the local consistency of VO [28], and thereby reducing the drift of the ego-motion. A way of ensuring the global consistency of the map is by detecting the return to a previously visited location, a loop closure.

2.3.1 Loop closure

Because of the drift inherent in successively building a map by incremental, imperfect odometry, the map representation will never be perfectly aligned when returning to a previously visited location. To make sure the map is globally consistent, the SLAM algorithm needs to explicitly connect the new location with the previously visited location to which it corresponds, and adjust the path in between to take the new connection into consideration in order to make the parts of the map involved consistent. The process thus has two basic steps: loop detection and loop closing.

It is important to note that while loop closures are very useful in a SLAM system, introducing erroneous loop closures can cause catastrophic errors in the map, and ego-motion estimates that are hard to recover from, if past data associations can not be undone. It is thus very important not to introduce false positives when closing loops. A common way of measuring the performance of a loop detector is the recall at 100 % precision, i.e. the proportion of all the true loop closures detected without accepting any false positives. With good VO, only occasional loop closures are needed to straighten out the map and trajectory.

In the case of monocular V-SLAM, there are three general categories of loop detection [53]: map-to-map, image-to-map and image-to-image.

Map-to-map loop detection consists of matching the features of two submaps considering the feature descriptor as well as the relative geometric position of the features. This method has been used in an EKF SLAM system by Clemente *et al.* in 2007 [54], and in a Graph-based SLAM system by Eade and Drummond in 2008 [55]. When the submaps have too few features in common to match, 100 % precision cannot be maintained [53].

Image-to-map loop detection aims to find correspondences between the latest image and features in a map. This method is used in a Graph-based SLAM system by Lim *et al.* in 2014 [19] and in EKF SLAM with submaps by Williams *et al.* in 2008 [56]. Lim *et al.* in 2014 used a vocabulary tree to find feature correspondences with earlier feature point locations and when enough features match geometric verification is performed using RANSAC to reject outliers [19].

Image-to-image loop detection, or appearance only based loop detection, works by matching the current image to previously seen images. One such algorithm is FAB-MAP 2.0, by Cummins and Newman in 2011 [21], further details of which is given below. Another variant of the image-to-image approach is matching a recent sequence of images to previous sequences, as implemented in SeqSLAM by [22] in 2012 [22]. In the case of vehicles travelling on roads, it is clear that sequences of images from a forward looking camera can provide useful information for resolving problems with visual aliasing, since the vehicles in the same location will move along similar paths. However, the procedure used by Milford and Wyeth in 2012 [22], of taking the absolute difference between down-sampled and patch normalised images is highly sensitive to changes in view-point orientation [57]. A representation of the image differences that is less susceptible to such variations could improve the performance.

One limitation in common for the image-to-image systems examined here is that only a topological loop closure is achieved. If a geometric loop closure is sought the relative pose between the recent image and the found match must be computed.

FAB-MAP

FAB-MAP 2.0 [21], by Cummins and Newman in 2011, is the state of the art method for loop closure detection [22][58]. It builds on FAB-MAP [20] by Cummins and Newman in 2008. Both use the Bag-Of-Words (BoW) representation of images. The words consist of feature descriptor vectors detected in the image, that have then been quantised with respect to a vocabulary, i.e. the descriptor is replaced by the closest word, in L_2 sense, in the vocabulary. The vocabulary is built by clustering the descriptors found in a training set of images and choosing the centres of the clusters as the vocabulary words. The set of descriptors that are represented by each word corresponds to the Voronoi region in the descriptor space around the word. An image is then a “bag” of the vocabulary words found in the image. The point of this procedure is to get a discrete representation of the visual words which can be indexed. This allows for simple enumeration of the vocabulary words appearing in each image as well as the creation of an inverted index which describes which location each word appears in.

A common strategy for building a vocabulary is to initialise the clustering with uniformly distributed random words. Cummins and Newman in 2011 showed that a radius-based initialization that requires the initial guesses to be separated by a minimum distance gives improved recall at a given precision [21].

The loop detection is done by matching the BoW description of the latest image to those previously seen, taking into account the uniqueness of the words, and their co-visibility. FAB-MAP can either assume the features are independent of each other, the naive Bayes approximation, or each feature can be conditioned on at most one other feature, resulting in a tree structured graphical model. The naive Bayes approach has one unique solution, the probability of observing a feature is proportional to its frequency in the training data. There are many different tree structures for a given number of nodes, the one that most closely approximates the original distribution is a Chow-Liu tree [20]. Word occurrence are highly correlated between words, and accounting for the correlation in this approximate manner is shown to improve results [20].

The probabilistic model, underlying FAB-MAP, is based on unobservable "scene elements", e_q, which give rise to visual word observations, z_q. These are binary variables that take the value 1 when the scene element is in view or when a word describing that scene element is detected, respectively. The detection of words is characterised by two parameters corresponding to the probabilities $p(z_q = 1|e_q = 1)$ and $p(z_q = 1|e_q = 0)$ which are the true and false positive probabilities. These two parameters together with a training data set is needed to initialise, i.b build vocabulary and tree, and run the FAB-MAP algorithm. The observation derived from an image at time k is $Z_k = z_1, ..., z_v$, with v being the number of words in the vocabulary. A location L_i is modelled as the belief about the presence of scene elements $p(e_1 = 1|L_i), \ldots, p(e_1 = 1|L_i)$. This allows a formulation of the belief about the current location very similar to a recursive Bayesian estimation. The probability for the ith location being the current location given the observations Z_1 to Z_k can be written

$$p(L_i|Z_{1:k}) = \frac{p(Z_k|L_i, Z_{1:k-1})p(L_i|Z_{1:k-1})}{p(Z_k|Z_{1:k-1})}, \tag{2.69}$$

where $p(L_i|Z_{1:k-1})$ is the location prior, $p(Z_k|L_i, Z_{1:k-1})$ the observation likelihood, and $p(Z_k|Z_{1:k-1})$ a normalising term. Commonly, the normalising term does not have to calculated the in Bayesian estimation because the probability function covers all possible outcomes, however, in (2.69), the outcomes only cover previously visited locations, and the possibility that the measurement is taken from an entirely new location still remains. To determine this, the normalising term must also be evaluated [20].

FAB-MAP will give a probability for each previous location of it being a loop closure with the latest image, and those above a certain threshold will be accepted as a loop closure. FAB-MAP 2.0 also performs a verification of the epipolar geometry between the potential loop closures with RANSAC and weighs this into the ranking.

2.3.2 EKF SLAM

EKF SLAM is based on the EKF. The EKF applies the classic Kalman filter to a non-linear model by linearising the model, using the first order Taylor expansion. This form of SLAM was the first to be implemented. It is an online form of SLAM and uses feature-based maps. As EKF SLAM is based on the EKF it assumes that the measurement noise and the motion model uncertainty has a Gaussian distribution. EKFs tend to handle large uncertainties in the posterior poorly, since the linearisation may introduce errors too large to handle [9, pp. 312]. The EKF SLAM algorithm is also limited to only using positive information, it cannot take into account the absence of landmarks [9, pp. 313].

The state vector in EKF SLAM consists of the current pose, an n-dimensional vector, x_t, and the landmark positions, m. Call the combined M-dimensional state vector $y_t = (x_t^T, m^T)^T$. EKF SLAM calculates an approximation to the posterior $p(y_t|z_{1:t}, u_{1:t})$. This is done by following the standard EKF procedure outlined below.

Initialise the mean and covariance of y_t in the origin with zero variance for the pose state and infinite variance for the landmarks,

$$\mu_0 = \begin{pmatrix} 0 & \dots & 0 \end{pmatrix} \quad \text{and} \quad \Sigma_0 = \begin{bmatrix} 0 & 0 \\ 0 & \infty \cdot I \end{bmatrix} \tag{2.70}$$

for the mean and covariance matrix respectively. Σ_0 is an $M \times M$ matrix in which the diagonal zero is $n \times n$, and $\infty \cdot I$ is a matrix with ∞ on the diagonal. Then the predicted mean and covariance is calculated:

$$\bar{\mu}_t = g(\mu_{t-1}, u_t), \tag{2.71}$$

where $\bar{\mu}$ is the predicted mean, μ_{t-1} is the previous estimate of the mean, and $g(\mu_{t-1}, u_t)$ is the motion model.

$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t \tag{2.72}$$

Here, $\bar{\Sigma}_t$ is the covariance of the prediction, $G_t = \frac{\partial g(\mu_{t-1}, u_t)}{\partial y_{t-1}}$ is the Jacobian of the motion model, and R_t is the motion noise. This is how the covariance propagates in the linear model [9, pp. 314].

The following assumes that the data association problem is solved by a feature tracker, loop closure algorithm, or other method so that correspondences of the features are known and stored in a correspondence vector, c_t, where the elements c_t^i are the index of the ith landmark seen at time t, $1 < i < N_t$, with N_t the number of landmarks at time t. If the landmark has never been seen before, add the estimated mean and covariance, of the landmark position, to the end of the state vector. In the case of bearing only measurement, there is no sensible estimate of the position of the landmark until it has been seen multiple times, and the position can be triangulated.

The mean and covariance can then be updated by calculating the estimated measurement for each individual landmark seen at time t, $\hat{z}_t^i = h_j(\bar{\mu}_t)$, where h_j is the measurement function and $j = c_t^i$, that is the expected measurement given what is known about the landmarks so far. For a camera, this is the pixel coordinates of the landmark, when projecting it to the camera of the current pose. For each landmark the update is then carried out:

$$\begin{aligned} K_t^i &= \bar{\Sigma}_t^{i-1} H_t^{iT} (H_t^i \bar{\Sigma}_t^{i-1} H_t^{iT} + Q_t)^{-1} \\ \bar{\mu}_t^i &= \bar{\mu}_t^{i-1} + K_t^i (z_t^i - \hat{z}_t^i) \qquad\qquad , \\ \bar{\Sigma}_t^i &= (I - K_t^i H_t^i) \bar{\Sigma}_t^{i-1} \end{aligned} \tag{2.73}$$

where K_t^i is the Kalman gain, and H_t^i is the Jacobian of the measurement function. Finally, setting $\mu_t = \bar{\mu}_t^{N_t}$, and $\Sigma_t = \bar{\Sigma}_t^{N_t}$, gives the estimate of the mean and covariance for the entire state.

One of the drawbacks of EKF SLAM is the poor scaling with the number of landmarks. The complexity of EKF SLAM is $\mathcal{O}(l^2)$ for l landmarks. Since the number of landmarks increases with the distance driven, squared complexity means the computational problem becomes intractable for large maps.

There are several variants of Kalman filter based SLAM algorithms, among them other variants of the basic Kalman filter that has been extended to handle nonlinear filtering such as the Unscented Kalman Filter (UKF). There are also similar filters that, unlike the EKF and other direct descendants of the Kalman filter, uses covariance matrices to quantify the uncertainty, use the inverse of the covariance matrix, the information matrix. These are called Information Filters (IFs). The Extended Information Filter (EIF) is one such filter extended to handle the nonlinear case by linearisation, as in the EKF [9, pp. 75]. IFs trade simple mean calculation and complex covariance calculation in Kalman filters, for complex mean calculation and simple information matrix calculation [9, pp. 73]. One major benefit of using IFs is that, unlike covariance matrices, the information matrices in SLAM are usually sparse, since there is only odometry between consecutive poses, and only a subset of the landmarks are seen from each pose. An information matrix is only non-zero when there is a connection between the corresponding pose-pose or pose-landmark pair [9, pp. 79]. Sparse problems can be solved significantly faster than dense ones.

2.3.3 FastSLAM

FastSLAM is the result of applying particle filters to solve the SLAM problem. A particle filter represents a probability distribution by a finite number of random samples of the distribution, instead of a parametric description of a certain probability function, such as the mean and covariance parameters used to represent Gaussian distributions [9, pp. 85]. For this reason particle filters are sometimes called nonparametric filters. While the sample based representation used in particle filters is only approximate, it can represent a much broader class of distributions, than the commonly used Gaussian distribution, e.g. multimodal distributions. Non-linear transformations of distributions are also easily applied to sets of samples without the need for linearisation, to generate a set of samples in the resulting distribution [9, pp. 97].

To maintain performance, particle filters need a certain density of particles in the state space, to assure that some particles are close to the correct state [9, pp. 112]. As the “volume” of the state space grows exponentially with its dimension and the dimension of the state space grows with the number of landmarks, using a particle filter to estimate the entire state, including landmark positions, would quickly become intractable. Fortunately, since there are no measurements between landmarks, the correlations between landmarks arise solely through the uncertainty in the state [9, pp. 437]. Thus, if the poses are known, the landmarks are independent of each other. Each individual particle assumes that the trajectory is known (the uncertainty is instead described by the spread of the particles pose estimates), and can therefore estimate the position of the landmarks independently of each other. This can be described as factorising the SLAM posterior (2.65) to

$$p(x_{1:t}, m | z_{1:t}, u_{1:t}, c_{1:t}) = p(x_{1:t} | z_{1:t}, u_{1:t}, c_{1:t}) \prod_{n=1}^{N} p(m_n | x_{1:t}, z_{1:t}, c_{1:t}), \tag{2.74}$$

for a trajectory $x_{1:t}$, a map m of N landmarks with the correspondences $c_{1:t}$, and odometry and measurements, $u_{1:t}$ and $z_{1:t}$. Thus, each landmark can be independently estimated with a parametric filter like the EKF. This greatly reduces the pose states, whose posterior distribution is estimated with particles, to a constant number of states, 6 in the case of 3D poses. The resulting particle filter is called a Rao-Blackwellised particle filter. In FastSLAM, each particle has an estimate of the pose, and estimates the location of the landmarks, with an EKF for each landmark [9, pp. 439]. A particle with index $[k]$ in FastSLAM thus contains an estimate of its trajectory, $x_{1:t}^{[k]}$, and a set of parametric, specifically Gaussian, estimates for all the landmarks seen so far, $\{\mu_1^{[k]}, \Sigma_1^{[k]}, \mu_N^{[k]}, \Sigma_N^{[k]}\}$. The FastSLAM algorithm iterates over each of the M particles and does a motion prediction and measurement update.

The motion prediction update differs between FastSLAM 1.0 and 2.0. In the 1.0 algorithm the proposal distribution sampled is simply the motion prediction $p(x_t | x_{t-1}, u_t)$, while the 2.0 version samples a distribution $p(x_t | x_{1:t-1}, u_{1:t}, z_{1_t})$. The benefit of the 2.0 approach is that the proposal distribution better covers the target distribution [9, pp. 451]. If the control and odometry are poor predictors of the posterior, few of the samples drawn from the proposal distribution will lie in the regions of high probability density in the target distribution, the posterior. This means that the distribution of particles diverge from the posterior. A way of coping with this effect is resampling [9, pp. 447], which is done as a last step.

Before resampling the measurement update is carried out, which consists of doing an EKF update, of the mean and covariance, of each measured feature, in each particle. If a feature has not been seen the estimate remains unchanged.

When resampling, M new particles are drawn with replacement from the current set of particles, which then become the new set of particles used to represent the posterior. The probability of a particle being drawn is proportional to its importance weight. The importance weight, $w_t^{[k]}$, is the quotient of the target distribution and the proposal [9, pp. 448]

$$\begin{aligned} w_t^{[k]} &= \frac{\text{target}}{\text{proposal}} = \frac{p(x_{1:t}^{[k]} | z_{1:t}, u_{1:t}, c_{1:t})}{p(x_{1:t} | z_{1:t-1}, u_{1:t}, c_{1:t-1})} \\ &= \frac{\eta\, p(z_t | x_t^{[k]}, z_{1:t-1}, u_{1:t}, c_{1:t}) p(x_{1:t}^{[k]} |, z_{1:t-1}, u_{1:t}, c_{1:t})}{p(x_{1:t} | z_{1:t-1}, u_{1:t}, c_{1:t-1})}, \\ &= \eta\, p(z_t | x_t^{[k]}, z_{1:t-1}, u_{1:t}, c_{1:t}) = \eta\, p(z_t | x_t, c_t) \end{aligned} \tag{2.75}$$

which is the measurement likelihood. η is a normalising factor that can be ignored, since the probability of drawing a particle is only proportional to $w_t^{[k]}$. Resampling is done as a last step in FastSLAM, after the measurement update.

One of the disadvantages of particle filter is the inflexibility under loop closure. When the particles are resampled, fewer and fewer estimates of "old" poses will be represented among the particles. Since only the estimates still represented by one of the particles are available for the filter to choose from, it might be that the "locally" best approximations no longer contains the "globally" best match, when including a loop closure.

2.3.4 Graph-based SLAM

Graph-based SLAM has a very intuitive structure: each pose and each landmark is a node in a graph, and between the nodes there are edges that represent measurements. Typically there are edges between two successive pose nodes, and between a pose node and each node corresponding to a landmark seen from that pose, as is illustrated in figure 2.10. These measurements form constraints on the location of the poses and landmarks involved. Since no real measurement is exact these constraints will not be perfectly consistent. The goal of graph-based SLAM is to find a configuration of the nodes which minimise this inconsistency.

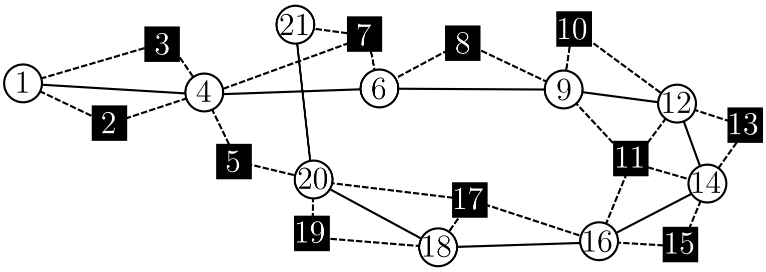


Figure 2.10: Basic structure of the graph in graph-based SLAM. White circles are pose nodes, black squares are landmarks and solid and dashed edges are pose-to-pose and pose-to-landmark constraints respectively. At 20 and 21 loop closures are detected when old landmarks 5 and 7 are observed.

Graph-based SLAM system can be split into a front-end and a back-end. The front-end builds a graph from the measurements and finds loop closures when they arise, setting the constraints of the graph. The back-end takes the graph built by the front-end and optimises it with respect to the constraints in the graph. The front-end deals directly with the sensor data, and thus is highly dependent on the kind of sensors used, the method of finding loop closures and the specifics of the implementation, while the back-end works on the more general and abstract representation of the graph.

Each edge in the graph corresponds to an error function. The type of error function depends on what type of measurement the edge represents. A pose-pose edge could be the Euclidean distance between two poses or the 6 DoF relative transform between the nodes, or a pose-landmark edge could represent the reprojection error in an image in which the landmark appears. The error is the difference between the actual measurement and the expected measurement, given the current values in the graph:

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j). \tag{2.76}$$

Here, $\mathbf{e}_{ij}$ is the error associated with the edge between node $\mathbf{x}_i$ and node $\mathbf{x}_j$, and $\mathbf{z}_{ij}$ is the measurement between nodes with index i and j. $\hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ is the expected measurement for the pair of nodes $\mathbf{x}_i$, $\mathbf{x}_j$. A common way of finding the parameters that best explains a set of measurements is minimising the square of the errors over the set of possible parameters, which results in the maximum likelihood estimate. Assuming the measurements are affected independently by Gaussian noise, this is equivalent to finding the node poses that minimises the negative log-likelihood of the observations [59]. Let $\mathbf{x} = (\mathbf{x}_1^T, ..., \mathbf{x}_T^T)^T$ be a vector of poses $\mathbf{x}_i$, and Ω_{ij} be the information matrix of the measurements from node i to node j. The likelihood of the observations given the poses and landmarks is

$$p(\mathbf{z}|\mathbf{x}) = \prod_{i,j \in C} \eta\, e^{-(\mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij})^T \Omega_{ij} (\mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij})} = \prod_{i,j \in C} \eta\, e^{-\mathbf{e}_{ij}^T \Omega_{ij} \mathbf{e}_{ij}}, \tag{2.77}$$

where C is the set of all pairs i, j, for which a measurement exists, and Ω_{ij} is the information matrix. Taking the logarithm of this gives

$$\ln \left[\prod_{i,j \in C} \eta\, e^{-\mathbf{e}_{ij}^T \Omega_{ij} \mathbf{e}_{ij}} \right] = \sum_{i,j \in C} \left(\ln \eta - \mathbf{e}_{ij}^T \Omega_{ij} \mathbf{e}_{ij} \right) = A - \sum_{i,j \in C} \mathbf{e}_{ij}^T \Omega_{ij} \mathbf{e}_{ij}, \tag{2.78}$$

where A is a constant that does not depend on the node states. Maximising the right hand side in this, finding the maximum likelihood estimate, is equivalent to minimising

$$F(\mathbf{x}) = \sum_{i,j \in C} \mathbf{e}_{ij}^T \Omega_{ij} \mathbf{e}_{ij}, \tag{2.79}$$

which is the sum of squared error terms weighted by their information matrix. This can be written as finding a solution to $\mathbf{x}^*$:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} F(\mathbf{x}). \tag{2.80}$$

With a good initial guess, $\check{\mathbf{x}}$, of the poses and landmark positions, the Gauss-Newton or Levenbarg-Marquardt algorithms can be used for for minimising F [50]. These algorithms rely on iteratively refining $\check{\mathbf{x}}$ by linearising the error functions, and solving the resulting linear minimisation problem to find a better state vector.

Simplifying the notation for $\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ to $\mathbf{e}_{ij}(\mathbf{x})$ and $\mathbf{e}_{ij}(\check{\mathbf{x}})$ to $\check{\mathbf{e}}_{ij}$, the linear approximation of the error function around the initial guess can be written as its first order Taylor expansion,

$$\mathbf{e}_{ij}(\mathbf{x}) = \mathbf{e}_{ij}(\check{\mathbf{x}} + \Delta\mathbf{x}) \approx \check{\mathbf{e}}_{ij} + J_{ij}\Delta\mathbf{x}, \tag{2.81}$$

where J_{ij} is the Jacobian of $\mathbf{e}_{ij}(\mathbf{x})$ in $\check{\mathbf{x}}$. The terms in (2.79) can be expressed as

$$\begin{aligned} \mathbf{e}_{ij}^T \Omega_{ij} \mathbf{e}_{ij} &\approx (\check{\mathbf{e}}_{ij} + J_{ij}\Delta\mathbf{x})^T \Omega_{ij} (\check{\mathbf{e}}_{ij} + J_{ij}\Delta\mathbf{x}) \\ &= \check{\mathbf{e}}_{ij}^T \Omega_{ij} \check{\mathbf{e}}_{ij} + 2\check{\mathbf{e}}_{ij}^T \Omega_{ij} J_{ij} \Delta\mathbf{x}_{ij} + \Delta\mathbf{x}^T J_{ij}^T \Omega_{ij} J_{ij} \Delta\mathbf{x} \\ &= c_{ij} + 2\mathbf{b}_{ij}^T \Delta\mathbf{x}_{ij} + \Delta\mathbf{x}_{ij}^T H_{ij} \Delta\mathbf{x}_{ij}, \end{aligned} \tag{2.82}$$

where $c_{ij} = \check{\mathbf{e}}_{ij}^T \Omega_{ij} \check{\mathbf{e}}_{ij}$, $\mathbf{b}_{ij}^T = \check{\mathbf{e}}_{ij}^T \Omega_{ij} J_{ij}$, and $H_{ij} = J_{ij}^T \Omega_{ij} J_{ij}$. Now the approximation of $F(\mathbf{x})$ close to $\check{\mathbf{x}}$ can be written as

$$\begin{aligned} F(\check{\mathbf{x}} + \Delta\mathbf{x}) &\approx \sum_{i,j \in C} c_{ij} + 2\mathbf{b}_{ij}^T \Delta\mathbf{x}_{ij} + \Delta\mathbf{x}_{ij}^T H_{ij} \Delta\mathbf{x}_{ij} \\ &= c + 2\mathbf{b}^T \Delta\mathbf{x} + \Delta\mathbf{x}^T H \Delta\mathbf{x}. \end{aligned} \tag{2.83}$$

The quadratic form in this has a global minima if H is positive semi-definite. H is an information matrix [50], thus at least positive semi-definite [60]. The global minima occurs when the gradient of the quadratic form is $\mathbf{0}$, that is

$$2\mathbf{b} + H\Delta\mathbf{x} = \mathbf{0}. \tag{2.84}$$

Thus, solving $H\Delta\mathbf{x}^* = -2\mathbf{b}$ gives $\Delta\mathbf{x}^*$. Setting the new initial guess to $\check{\mathbf{x}} + \Delta\mathbf{x}^*$, and iterating until some criteria of convergence is reached will yield an approximation of $\mathbf{x}^*$.

In V-SLAM it is more common to use the Levenberg-Marquardt algorithm for finding $\Delta\mathbf{x}^*$. It is very similar to Gauss-Newton, the difference being that instead of solving (2.84),

$$2\mathbf{b} + (H + \lambda I)\Delta\mathbf{x} = \mathbf{0} \tag{2.85}$$

is solved. Here, λ is a parameter to be chosen for each iteration. If λ is “large enough”, $H + \lambda I$ will have full rank and be positive definite. This is helpful in the V-SLAM context, since H can lack positive definiteness due to scale ambiguity.

Solving (2.85) requires inverting $H + \lambda I$. Matrix inversion has complexity of $\mathcal{O}(n^3)$, which quickly becomes intractable for matrices of the sizes relevant to graph-based SLAM with landmarks. Luckily, H, is sparse and symmetric, stemming from the fact that each landmark is only seen from a few poses and the only pose-pose edges are between consecutive poses, as well as the terms $J_{ij}^T \Omega J_{ij}$ being symmetric, as is illustrated in figure 2.11. There are very efficient solvers for sparse symmetric systems, the complexity of which depend on the structure of H.

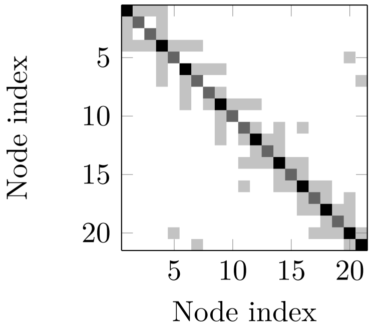


Figure 2.11: An illustration of the sparseness and symmetry of the information matrix, H, corresponding to the graph in figure 2.10. Only the shaded elements are non-zero, black elements are pose nodes, dark grey corresponds to landmarks and middle grey the edges between these. Loop closures manifest as off-diagonal elements at the symmetric pairs $\langle(20, 5), (5, 20)\rangle$ and $\langle(21, 7), (7, 21)\rangle$.

When tracking features in the images, sometimes erroneous associations are obtained, resulting in outliers when triangulating or adding edge constraints to the graph. These can have a large detrimental impact on the final result, since the square of an already large error can overwhelm the smaller, inlier errors. Seen in a Gaussian light, the outliers are deemed very unlikely measurements, so unrealistically large adjustments to the other estimates are made to accommodate the outlier. To lessen this effect one can use a robust error function. These usually mimic the squared error function for small errors to come close to a Gaussian maximum likelihood estimate, but grow slower than quadratically for larger errors to lessen the influence of the outliers. Such a robust error function is the Huber error, which, at a set magnitude of the error, grows linearly instead of quadratically, as is shown in figure 2.12.

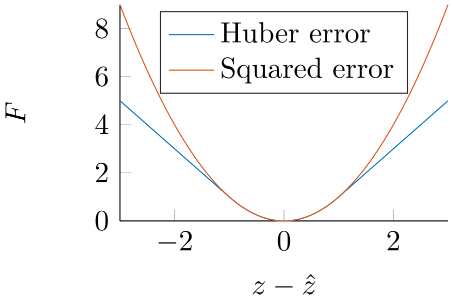


Figure 2.12: Comparison of the squared error function and the Huber error.

3

Implementation

To evaluate the performance of the methods described in chapter 2, a solution for the 6 DoF SLAM problem has been implemented. The rest of this chapter details the implementation of the solution.

The implementation is written in C++ and makes use third-party libraries. Non-primitive data types, mathematical methods and feature point related methods from the OpenCV library [29] are used. OpenCV was chosen since it is a widely used, cross-platform library for computer vision tasks with a C++ interface and includes most of the functionality that was provided by Matlab in the first implementation of the VO system. For handling the graph structure and optimisation the g2o library [60] is used as it is specifically tailored to graph-based SLAM problems and bundle adjustment. For loop closure detection OpenFABMAP [58], as provided in OpenCV, is used. OpenFABMAP is an open source implementation of FAB-MAP, the loop closure method for V-SLAM that was chosen for this implementation. Tunable parameters are presented in appendix A.

3.1 Visual Odometry

The implementation of the VO assumes known camera calibration parameters. The system is divided into two parts. The first takes raw camera images, rectifies the images and tracks features in the consecutive images. This enables the use of epipolar geometry for an initial estimate of the incremental ego-motion and triangulation of the detected features. The second part takes the initial estimate and further refines it with windowed bundle adjustment.

3.1.1 Features and tracking

To extract corresponding feature points between two consecutive images a KLT tracker [35] is used. The tracker is initialised with a specified number, *KLTNPoints*, FAST features [34], with an intensity threshold *KLTFASTThreshold*, in the first image and tracked in the second image. Points for which no corresponding point is found in the second image are removed. RANSAC, using the normalised 8-point algorithm, is used to remove outliers with a Sampson distance to the epipolar line greater than *KLTRANSAC8PointThreshold*. The number of RANSAC iterations is specified in *KLTRANSAC8PointIterations*. If points are lost, new FAST features are added in second image to later be tracked in the next image. VO gives better results when the features are evenly distributed in the image [32]. To achieve an even distribution, the image is divided into a grid of *KLTNWin* fields, with margins specified in *KLTWinWidthMargin* and *KLTWinBottomMargin*, where no grid fields are made to avoid creating features that are likely to disappear out of view in the next image. The number of feature points are counted in each field, and new FAST feature points will be added only in the fraction, *KLTWinThreshold*, of fields containing the least number of feature points.

As suggested by Scaramuzza *et al.* in 2009, if the distance between the matched points is less than *KLTMotionDistanceThreshold* pixels for a fraction equal to *KLT-MotionRatioThreshold* of the points, no motion is assumed, and the second image is discarded. This is repeated until an image not fulfilling this condition is found.

The fundamental matrix is calculated using the feature points remaining after tracking and RANSAC with the normalised 8-point algorithm described in section 2.2.1. Using a known camera calibration matrix, the essential matrix is obtained according to (2.48). From the essential matrix, two possible rotational matrices and two possible unit length translation vectors between the two frames can be obtained, as shown in (2.49). The equation shows that the two possible translation vectors are parallel, in opposite directions. To determine the correct translation vector, the speed of the vehicle, which can be obtained from wheel odometry or an IMU, is used. The speed is also used to fix the scale ambiguity of the monocular camera. This is done by setting the length of the translation vector to the distance travelled between the two images, according to the speed. To find the correct rotational matrix, the matrix describing a rotation less than 180° in all axes is selected.

The above steps are repeated for each consecutive pair of frames. The rotational matrix, R, and the translation, $\mathbf{t}$, are combined and expressed as a relative homogeneous transformation matrix:

$$T = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{3.1}$$

The current pose, expressed as a homogeneous transformation matrix, is found by right-multiplying the previous relative transformation matrices up to the current state. The first pose is initialised in the origin of the camera coordinate system, with the identity matrix.

3.1.2 Bundle Adjustment

Each pose is added as a pose node, *VertexCam*, in a g2o [60] graph, with an *SBACam* estimate. An *SBACam* estimate contains a fixed camera calibration matrix and an extrinsic camera matrix, a pose, to be optimised. The camera calibration matrix is previously known and set accordingly, and the pose is set to the pose obtained from the estimated essential matrix.

For each new feature point tracked across two consecutive frames, a point node, *VertexSBAPointXYZ*, is added to the graph. The point node has an estimate of a position in $\mathbb{R}^3$ to be optimised. The initial estimate is obtained by triangulating the point from the first two poses it is seen from. For each pose where the point is tracked, a projection edge, *EdgeProjectP2MC*, is added. The projection edge connects a pose node with a point node. The measurement is the pixel coordinate of the observed point in the image from the relevant pose. The error function is the reprojection error of the point to the image according to (2.11). The information matrix of the measurement is set to a diagonal matrix $\sigma^{-2} \cdot I$, where σ^2 is *PixelVar*. To fix the scale ambiguity of the monocular camera, a distance edge, *EdgeSBAScale*, is added between each consecutive pose node. The measurement is the geometric distance between the two poses, and the error function is the offset between the estimate and the measurement. The measurement is set to the distance between the poses according to the speed of the vehicle and the information to σ^{-2}, where σ^2 is *DistanceVar*.

Between each consecutive pose node, a pose-pose edge, *EdgeSBACam*, is created and added to the graph. These edges are not considered during the incremental VO, but are instead used when loop closures are detected to avoid doing costly BA optimisation on the entire graph. They are updated until the poses they connect fall out of the BA window. The measurement is the pose to pose transformation between the connected nodes, expressed in a 6-vector with Euclidean coordinates describing the transformation and a unit quaternion with real part omitted (assumed positive) describing the rotation, as $\mathbf{x} = (x, y, z, q_0, q_1, q_2)^T$. Using homogeneous transformation matrices, the error is calculated as

$$T_{err} = T_{meas}^{-1} T_1^{-1} T_2. \tag{3.2}$$

The information matrix is set to a diagonal matrix with $(\sigma_t^{-2}, \sigma_t^{-2}, \sigma_t^{-2}, \sigma_r^{-2}, \sigma_r^{-2}, \sigma_r^{-2})^T$ on the diagonal, where σ_t^2 is set to *TranslVar* and σ_r^2 is set to *RotVar*.

For each iteration, a graph with the projection edges and distance edges connecting the latest *BAWindow* number of pose nodes, are optimised *BAIterations* times with the Levenberg–Marquardt algorithm, *OptimizationAlgorithmLevenberg*, using a Huber error function and a solver based on sparse Cholesky factorisation, *LinearSolverCholmod*.

3.2 Simultaneous Localisation And Mapping

To get a full SLAM implementation, the estimate of the trajectory and map from the bundle adjusted VO is again adjusted when loop closures are detected to ensure that the map is consistent. This is divided into detecting loops using OpenFABMAP and finding the geometric relation between the current and old pose in a step similar to the first part of the VO.

3.2.1 FAB-MAP

The FAB-MAP 2.0 [21] algorithm is set up using a vocabulary and Chow-Liu tree [20], which have been created offline, as well as a training data set. The probability of a feature being detected, $p(z = 1 \mid e = 1)$, is set to *FMPzGe*, and the probability of a false positive, $p(z = 1 \mid e = 0)$, is set to *FMPzGNe*.

For each new image, a set of Star feature points, from the OpenCV library [29], which is a variant of the CENSURE [38] feature detector, is extracted. The threshold of the detector is dynamically adjusted, in maximum *FMStarIterations* iterations, to give between *FMStarNMin* and *FMStarNMax* feature points, and is initialised at *FMStarThreshold*. For the set of feature points, a SURF based BoW descriptor is extracted, and compared to the BoW descriptors previously added, using the FAB-MAP algorithm. If the normalised probability that the compared image matches a previously added image is above *FMMatchThreshold*, a loop closure hypothesis is added. If no match is found, the BoW descriptor, a key frame, is added to the test data. For a loop closure to be added, two loop closures in a row have to be made to the same key frame and be at least *LCDistThreshold*.

3.2.2 Geometric loop closure

For each new key frame, each frame following a key frame, and for each frame from which a loop closure is detected by FAB-MAP, FAST features, with an intensity threshold *LCFASTthreshold*, and BRISK feature descriptors are extracted from the according image and saved.

When a loop closure from the current image to a previous image is detected, the extracted feature points are exhaustively matched using the corresponding BRISK descriptors. Matches with a Hamming distance greater than *LCBRISKDistanceThreshold* are discarded. The remaining feature points are filtered with a normalised 8-point RANSAC with *LCRANSAC8PointIterations* iterations with a threshold for the Sampson distance to the epipolar line for inliers at *LCRANSAC8PointThreshold*. If the number of matched feature points between the current image and the previous image is greater than *LCNMatchedTreshold*, projection edges, *EdgeProjectP2MC*, between the corresponding pose node and feature points are added. The information matrix is set to a diagonal matrix, $\sigma^{-2} \cdot I$, where σ^2 is *PixelVar*. The requirement on the number of matches works as a firewall condition to rule out incorrect loop closures on the geometric consistency of the feature points. This is also used in the original implementation of FAB-MAP 2.0 [21] by Cummins and Newman in 2011, but is not implemented in OpenFABMAP [58].

The first time a previous image is used in a loop closure, the FAST feature points extracted from that image are matched against feature points extracted from the image after the previous image, using the same algorithm as described above. As the relative translation and rotation between the two images is known from the VO, the matched feature points can be triangulated. The feature points for which no match can be found in the following image are removed. For each triangulated feature point, a point node, *VertexSBAPointXYZ*, is created and the estimate obtained from the triangulation is set. Projection edges, *EdgeProjectP2MC*, are added between the corresponding pose nodes and point nodes. The information is set to a diagonal matrix $\sigma^{-2} \cdot I$, where σ^2 is *PixelVar*. As is described in section 2.1.1, a set of matched feature points between two views describes the direction to the other camera centre. Thus, adding a third view which relates to the same world points, where the distance between two of them is known, will fully determine the triangle described by the three camera centres, and fix the scale ambiguity of the monocular camera.

The pose-pose edges between all pose nodes, created in the BA step, and the loop closure projection edges described above, are optimised *LCIterations* times with the Levenberg–Marquardt algorithm, *OptimizationAlgorithmLevenberg*, using a Huber error function and a solver based on sparse Cholesky factorisation, *LinearSolverCholmod*.

4

Results

To evaluate the implementation, four different datasets from the publicly available KITTI dataset [10] are used. The datasets have different characteristics in terms of environment, traffic and speed to assess performance in realistic driving situations. The characteristics of the different datasets are summarised in table 4.1. The parameters used in the evaluation are the same for all datasets and presented in appendix A.

The result of each dataset is divided into all possible subsequences of lengths (100, 200, 300, ... , 800) m, and evaluated against highly accurate GPS measurements. The translation error is measured in percent and the rotation in degrees per travelled meter. The performance of the implementation is evaluated both with purely incremental VO and a complete SLAM system, with loop closures. A summary of the results is shown in table 4.3.

4.1 Visual Odometry

The results of the evaluation of the datasets are shown in figures 4.1 to 4.4. Each figure consists of four subfigures. The top left shows the trajectory, starting in the origin, estimated by the visual odometry system and the ground truth measured by GPS, visualising the consistency of the map created. Top right is a plot of the deviation of the estimate from ground truth in each frame and spatial dimension, showing the drift increasing over time. The bottom subfigures show the average relative error in translation and rotation, left and right respectively. The translation error is measured as the distance between the endpoints of the estimated subsequence and the corresponding ground truth subsequence when the starting poses of the subsequences are aligned, divided by the length of the subsequence. The rotation error is measured as the difference between the orientations of the final pose in the

#	Environment	Traffic	Speed	Frames	Distance	Overlapping distance	Name
1	Residential	+	+	4529	3.73 km	620 m	2011_10_03_drive_0027
2	Residential	+	+	4318	4.66 km	330 m	2011_09_29_drive_0071
3	Highway	++	+++	1175	2.60 km	0 m	2011_10_03_drive_0042
4	Rural road	++	++	434	7.86 km	0 m	2011_09_26_drive_0028

Table 4.1: Properties of the datasets used for evaluation of the implemented algorithms.

VO estimate and the ground truth when their starting poses are aligned. This gives an indication of the quality of the visual odometry; the specific measure used was chosen to enable comparison to the KITTI benchmark.

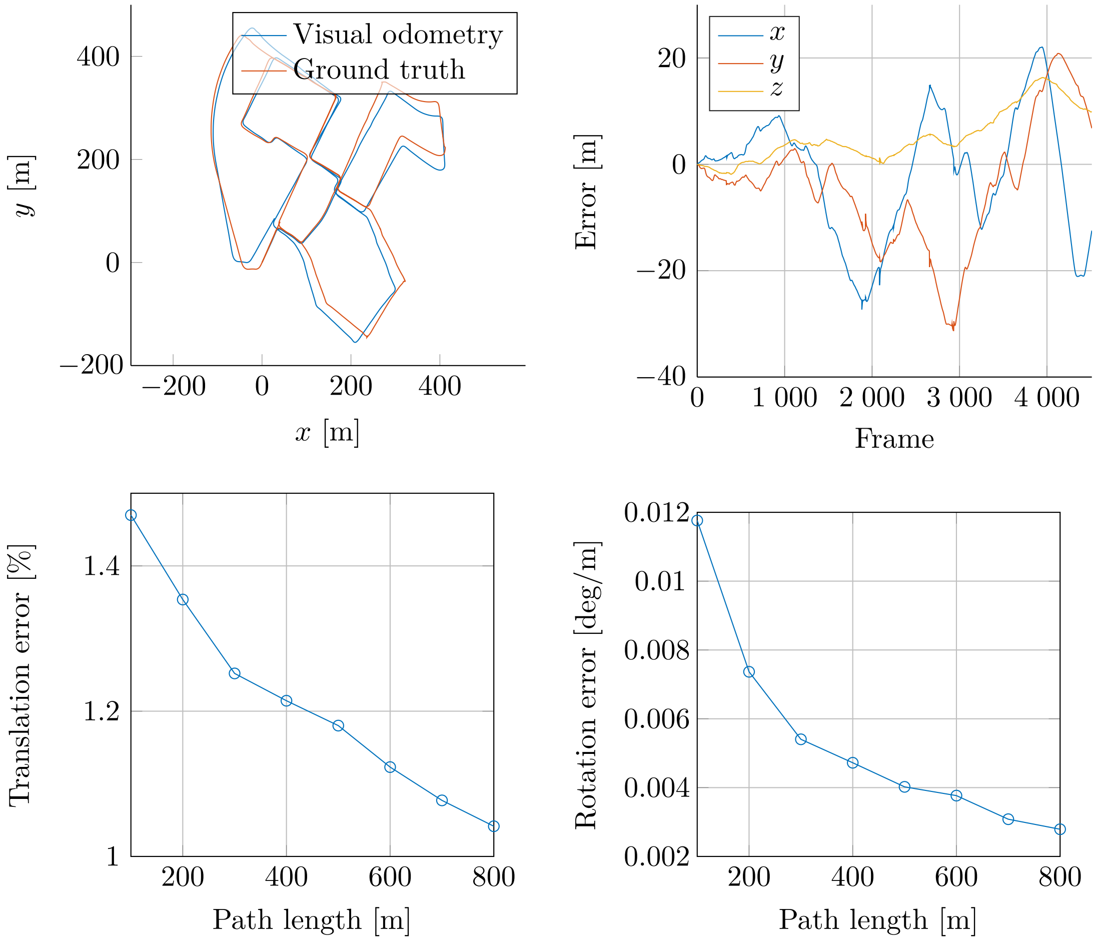


Figure 4.1: Plots of evaluation data for dataset 1 using VO. Average translation error: 1.22 %. Average rotation error: 0.00550 deg/m.

The execution time of the VO implementation is shown in figure 4.5. The total mean execution time for one step is 0.0990 s.

4.2 SLAM

For the OpenFABMAP 2.0 [58] algorithm a vocabulary, Chow-Liu tree and training data, consisting of randomly selected images from the KITTI datasets not occurring in the evaluation datasets, were used. The vocabulary was created using a cluster size of 0.45, and, from the training set of 1000 images, a vocabulary of 5191 words was generated.

Only dataset 1 and 2 are evaluated using the full SLAM algorithm, as the others do not contain any overlapping paths, and thus will render the same results as the VO algorithm. The results of the evaluation of the datasets are shown in figure 4.6 and figure 4.7. The figures show the same measures as used in section 4.1. Comparing

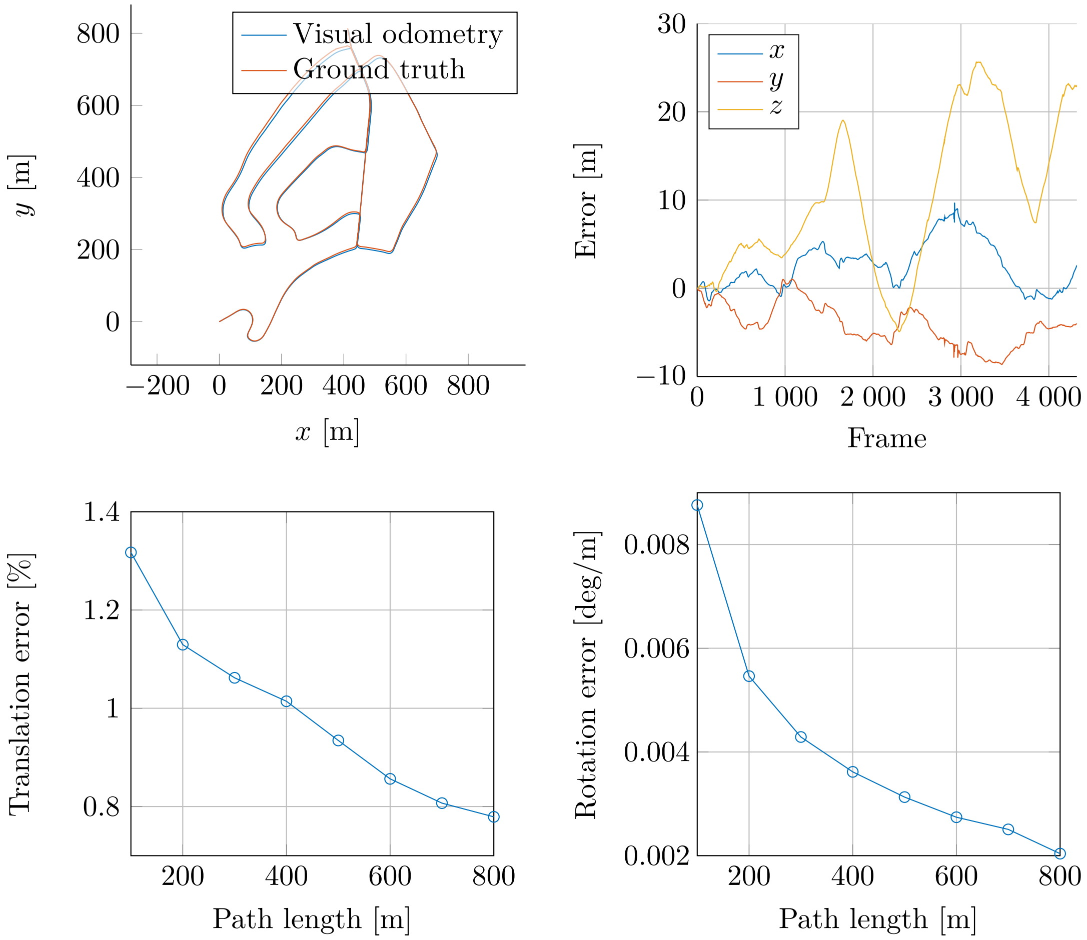


Figure 4.2: Plots of evaluation data for dataset 2 using VO. Average translation error: 0.995 %. Average rotation error: 0.00415 deg/m.

the trajectory plots to the ones from the VO figures, it can be seen that closing loops does enforce consistency as trajectories along the same stretch of road overlap. The drift plot also shows a decrease in deviation from ground truth compared to the VO estimate; especially note that the error becomes very small at the end when returning to the start of the sequence.

The execution time of the full SLAM algorithm is shown in figure 4.8. The median execution time for one step is 0.157 s and the mean execution time is 0.422 s. The steps that take considerably longer are steps where a loop closure is added and the graph is optimised.

#	Detected by FAB-MAP	Two in a row	Successful loop closures	Loop closures per meter
1	505	317	297	0.48 lc/m
2	126	42	40	0.12 lc/m

Table 4.2: Number of detected loop closures before and after the two firewall conditions.

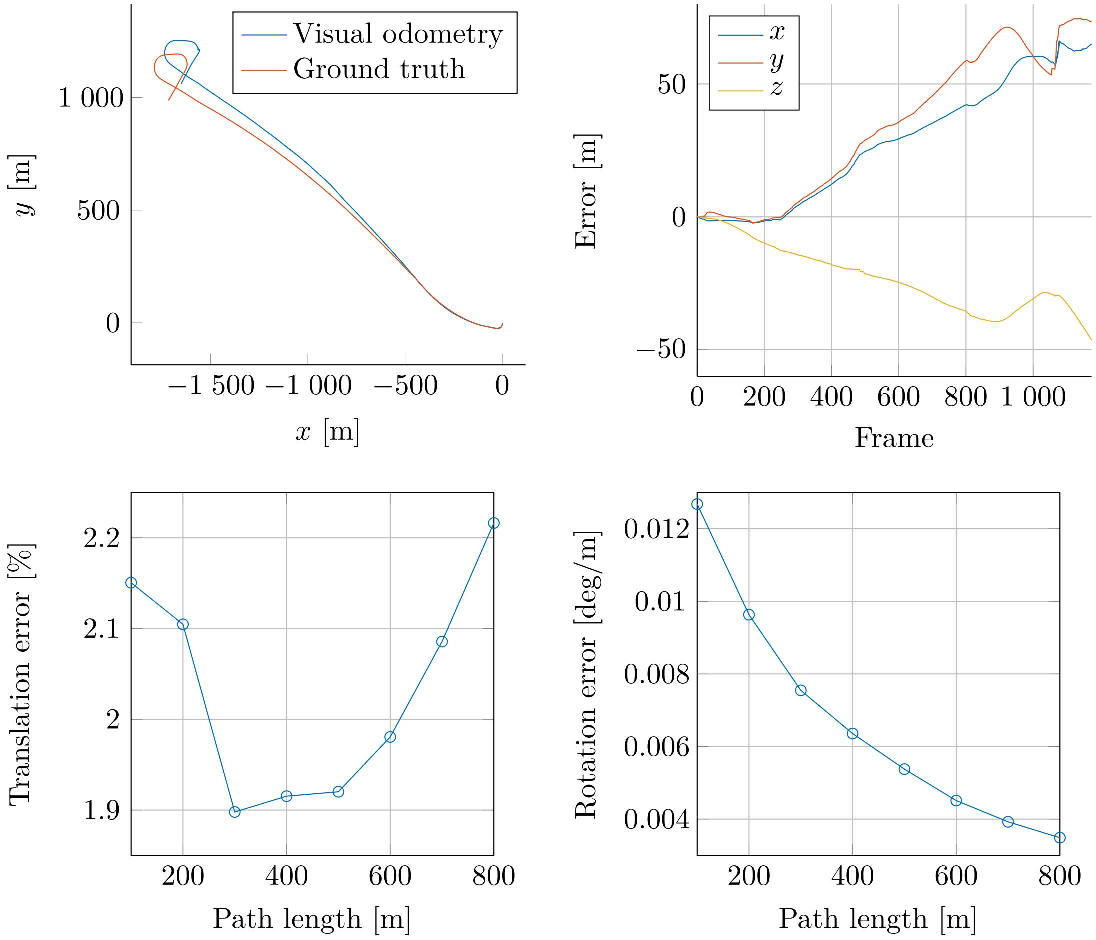


Figure 4.3: Plots of evaluation data for dataset 3 using VO. Average translation error: 2.03 %. Average rotation error: 0.00709 deg/m.

From the trajectory plots it seems a large part of the remaining difference between the estimate and the ground truth is due to a mismatch in orientation of the two trajectories. To investigate this, the trajectory estimated by the SLAM algorithm was aligned with the ground truth by modifying the initial pose to minimise the squared distance between points corresponding to the same frame in the two trajectories. This was done both with the initial position locked, i.e. exactly the same starting location in estimate and ground truth, and only changing the orientation of the estimate, and by full 6 DoF changes.

The results of the alignment can be seen in figures 4.9 and 4.10, showing that a majority of the deviation from ground truth can be removed by aligning the maps, and most of the improvement comes from adjusting the orientation.

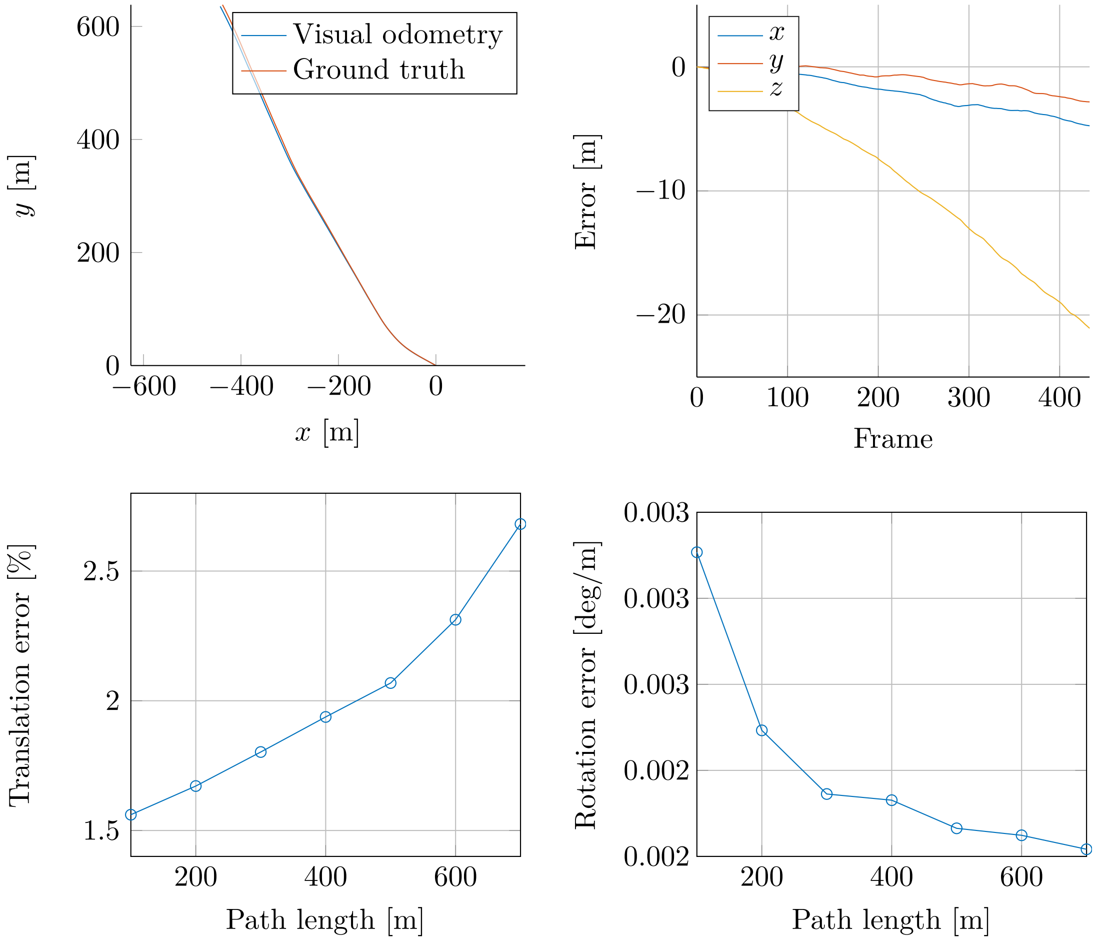


Figure 4.4: Plots of evaluation data for dataset 4 using VO. Average translation error: 1.82 %. Average rotation error: 0.00250 deg/m.

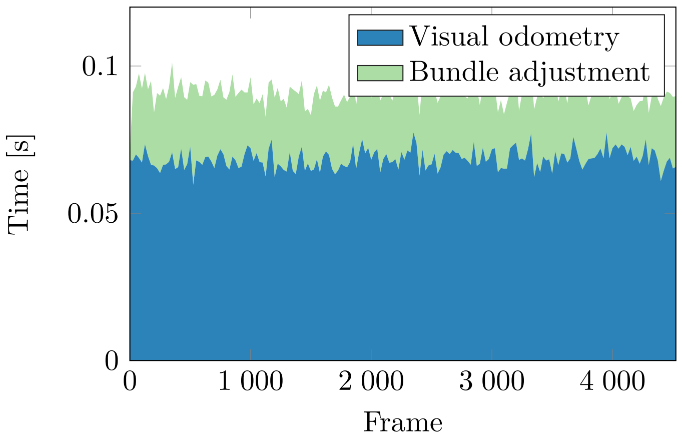


Figure 4.5: Execution time for every 25th step of the different parts of the VO for dataset 1.

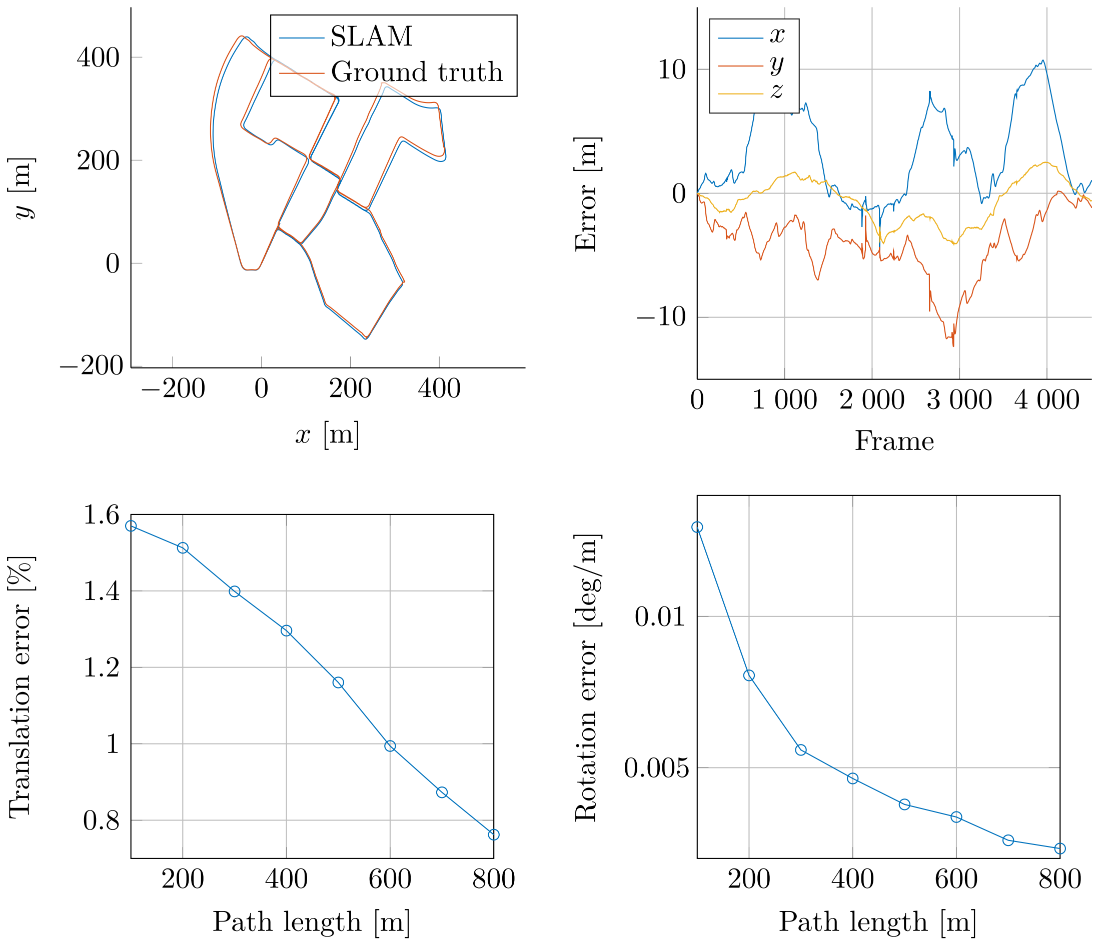


Figure 4.6: Plots of evaluation data for dataset 1 using the full SLAM solution. Average translation error: 1.21 %. Average rotation error: 0.00558 deg/m.

	Visual Odometry		SLAM		
#	Translation error	Rotation error	Translation error	Rotation error	Successful LC
1	1.22 %	0.00550 deg/m	1.21 %	0.00558 deg/m	279 (0.48 lc/m)
2	0.995 %	0.00415 deg/m	1.00 %	0.00405 deg/m	40 (0.12 lc/m)
3	2.03 %	0.00709 deg/m	-	-	-
4	1.82 %	0.00250 deg/m	-	-	-
Mean	1.21 %	0.00505 deg/m	1.11 %	0.00482 deg/m	

Table 4.3: Results of the evaluation of the datasets.

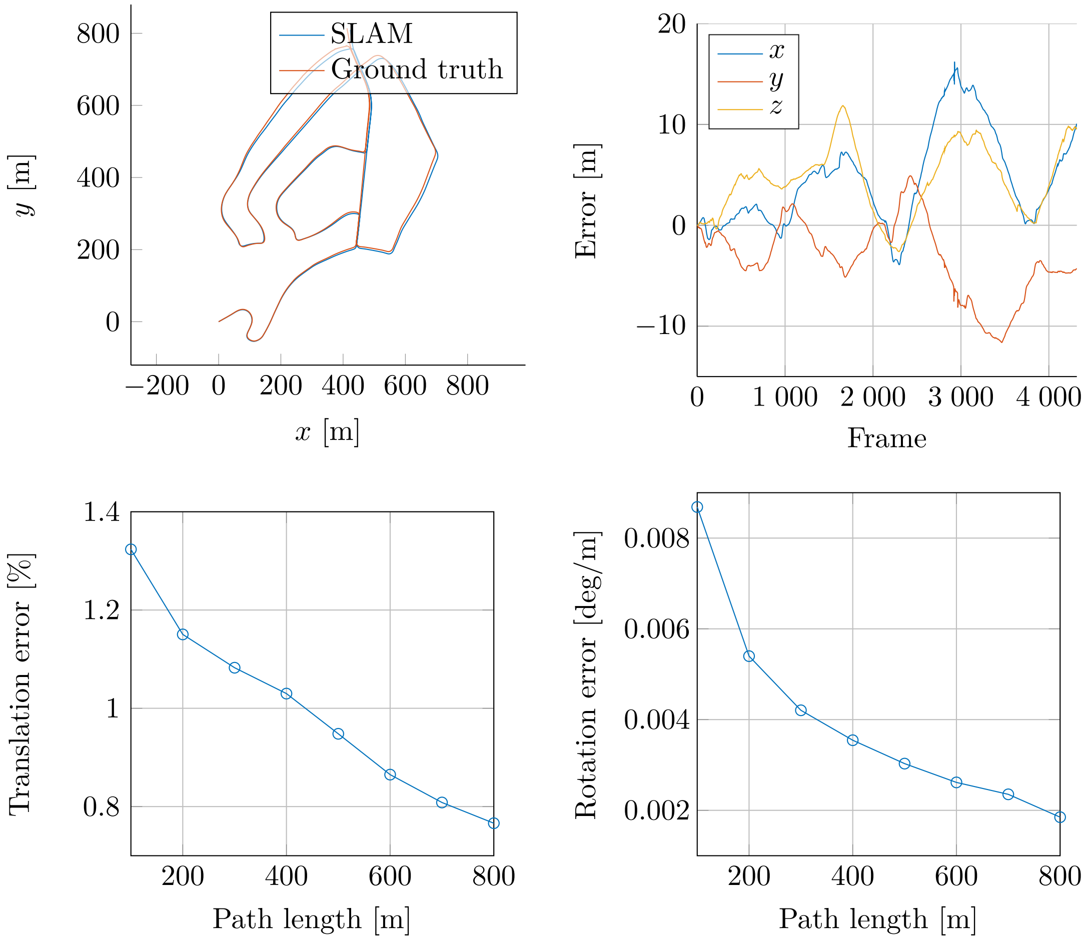


Figure 4.7: Plots of evaluation data for dataset 2 using the full SLAM solution. Average translation error: 1.00 %. Average rotation error: 0.00405 deg/m.

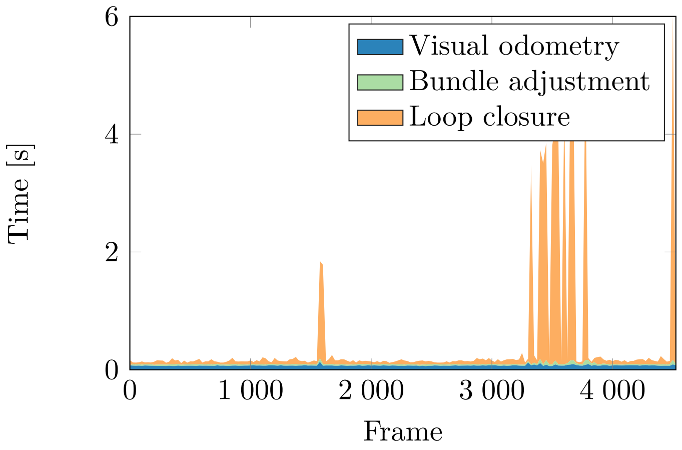


Figure 4.8: Execution time for every 25th step of the different parts of the SLAM algorithm for dataset 1.

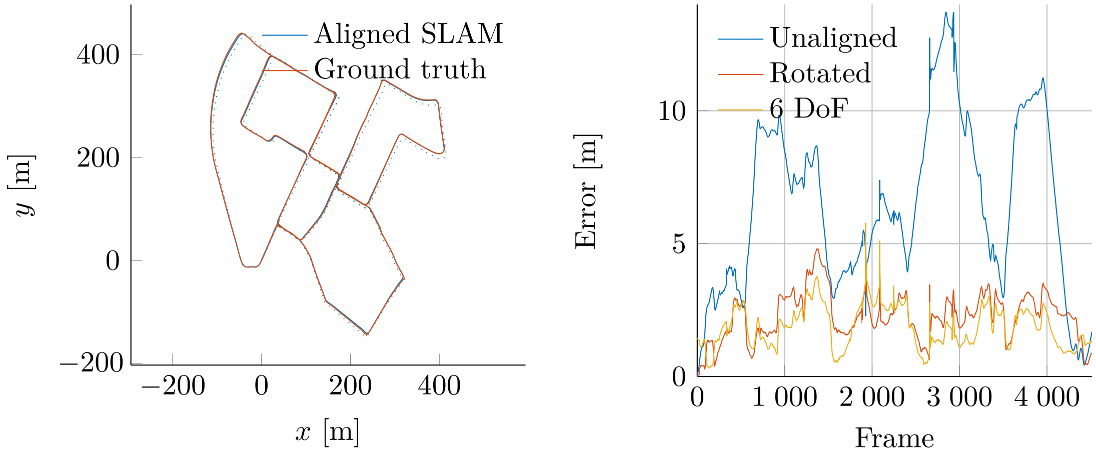


Figure 4.9: Plot of aligned trajectory and distance to ground truth for dataset 1. The dotted blue line in the left figure is the unaligned estimate. The RMS distance improved from 7.214 m to 2.435 m by rotating and 2.030 m with 6 DoF alignment

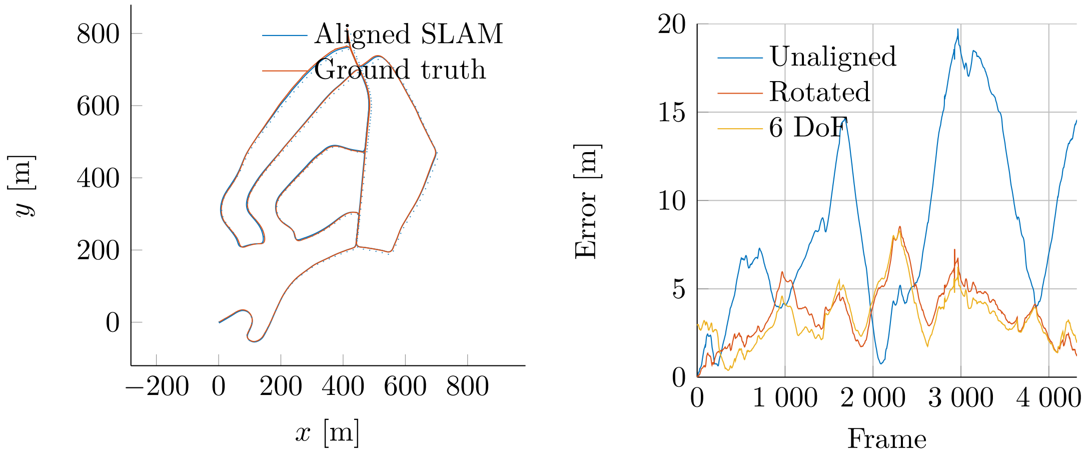


Figure 4.10: Plot of aligned trajectory and distance to ground truth for dataset 2. The dotted blue line in the left figure is the unaligned estimate. The RMS distance improved from 9.905 m to 4.044 m by rotating and 3.731 m with 6 DoF alignment

5

Discussion

The development of this area has accelerated in the recent years, due increased interest from the automotive industry and better digital cameras and processing units becoming available. Due to the recent development in this area, the availability of literature is limited. Big parts of our work builds on *Multiple View Geometry in Computer Vision* by Hartley and Zisserman [24], which is cited in almost every paper regarding the subject. The book covers the basics of camera modelling and terminology, which was our biggest challenge to overcome during this work, which is why we recommend this read. Others sources have mainly been conference proceedings, which in general are not going into details, and can be hard to grasp without prior knowledge. This chapter continues with a discussion of our implementation and ends with our ideas on how to continue this work in the future.

5.1 Implementation

The overall performance of the VO is good, and the mean of our test datasets would currently achieve a high rank among the V-SLAM algorithms on the odometry benchmark of the KITTI dataset [10]. Note however that this is not completely comparable, as there is no speed measurement available for the datasets used for the benchmark, which is needed for this implementation to resolve the scale ambiguity of monocular data.

In some situations the algorithm struggles. Three of these situations occur in dataset 3, the highway scenario. In this scenario, the close proximity of the camera has relatively few good features to track. Many of the feature points that are tracked end up on other, moving, traffic. Another problem is that points on repeating objects, like fences or road surface markings, can get matched to the wrong, similar looking, object, if the frequency of their recurrence matches poorly with the frame rate of the camera. The third problem occurs in a sharp turn, where many of the feature points end up on an overpass, which renders them approximately coplanar, as shown in figure 5.1. Coplanar points, as explained in section 2.2, will render the estimation of the fundamental matrix degenerate, and will give poor results from the VO. This problem could be partially resolved by using algorithms to directly estimate the essential matrix, like the 5-point algorithm by Nistér [31]. Another approach to solve the problem of when the close proximity of the camera has few good feature points could be to use lane feature based localisation, as is done by Ziegler *et al.* in 2014 [61]. The KLT tracker effectively uses the image patch as descriptor to match to a predicted location in the image, which, while tracking has produced longer feature tracks in environments with plenty of good features to track, also has the effect of continuing to track bad features in difficult environments when the prediction and image patch are similar.

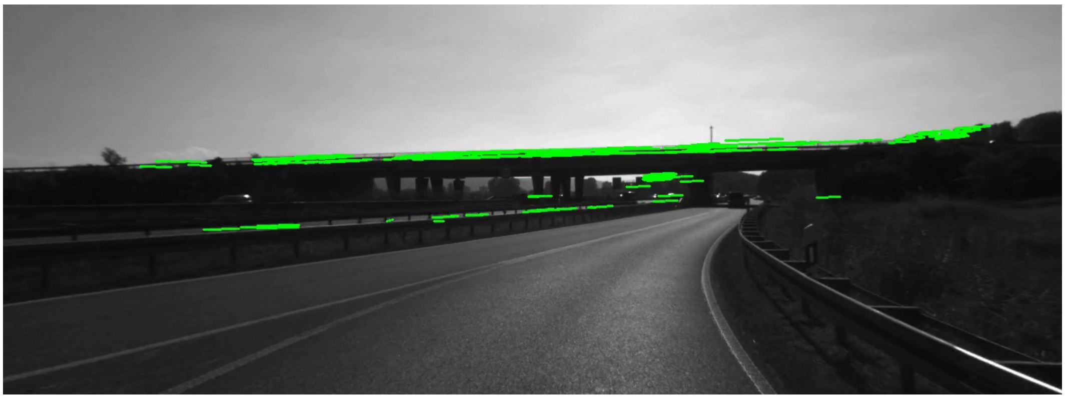


Figure 5.1: A frame from dataset 3, with the tracked feature points, which are approximately coplanar, shown in green.

As can be seen in figure 4.5, the execution time for the visual odometry is slightly faster than real-time, however, no particular effort has been directed to computational performance, and there are many aspects of the code that could be improved to achieve higher rate. The biggest improvements could be made in the RANSAC procedure and the motion estimation, by using other algorithms, like the 5-point algorithm by Nistér in 2004 [31]. Even further improvement could be made by using 3D-2D feature point correspondences, which only requires 3 correspondences, like PnP [62]. In discussions with Mikael Persson, the author of "Robust Stereo Visual Odometry from Monocular Techniques" [63], the current 2nd place holder in the KITTI benchmark, we have learned that using the other algorithms also would produce a better odometry than the 8-point algorithm, and therefore also would require less BA iterations. Another improvement that could reduce the number of required iterations of the BA is to, instead of using a robust error function such as Huber, remove outliers by identifying edges with high reprojection error and remove them.

The VO currently works better by setting unreasonably low variance on the distance edge between the poses to fix the scale. When using higher noise, the BA requires more iterations. This could be caused by a bad initialisation from the 8-point algorithm or point triangulation. If this is the case, this could also be improved by using another initial motion estimation algorithm. By using this low variance on the distance edge high demands are put on the accuracy of the speed measurement. In the raw data from the KITTI dataset [10], this is in general not a problem, since the speed measurement comes from a highly accurate IMU. However, in sharp corners, the travelled distance is often underestimated by the VO. This could be explained by that the speed given from the IMU is the forward speed of the vehicle, which, in corners, will be lower than the speed of the camera centre.

As can be seen when comparing figures 4.1 and 4.6, the average translation error on short path lengths actually gets worse when using the full SLAM algorithm. This is due to that the result from the BA is the best local solution, and by transforming this solution to pose-pose edges, with the same covariance on all edges, information about which pose-pose relations that are more uncertain, and thus should be corrected to get a more correct global solution, is lost. This could be improved by calculating the covariance based on the matched feature points.

Tests have shown that the choice of training data to use and build the vocabulary from has a high impact on the success rate of the FAB-MAP 2.0 [21] algorithm. When the algorithm was used with other training data, there were many false positives and very few correct loop closure detections. The false positives could be explained by perceptual aliasing, two images from different places that have some objects that produce very similar features, like lamp posts or other reoccurring objects. The effect of perceptual aliasing should be reduced by using a training data to determine how frequent each word in the vocabulary is, but, if the training data is insufficient, this will work poorly. The lack of correct loop closure detections could not be explained. The performance of the loop closure detection was greatly improved by using a better training data and using a slightly larger cluster size. By introducing the two firewall conditions, no false positive loop closures have yet been seen. However, no false positives from the FAB-MAP 2.0 algorithm by itself, without the firewall conditions, as is done in the experiments by Cummins and Newman in 2011 [21], has not been achieved.

The SLAM algorithm does not run in real-time. However, the loop closure detection is deemed to be possible to perform in real-time. In the implementation, for each new image a set of Star features and SURF descriptors are calculated, which currently is the bottle neck in the implementation. By instead using the tracked features and other, faster feature descriptors, the computation time could be improved. The time used for the comparison of BoWs is negligible compared to the time used for feature extraction. However, to optimise the graph in real-time would require some major improvements, as it now grows linearly with the number of frames, and takes several seconds just for a couple of thousand poses, as can be seen in figure 4.8. One possible area to investigate for improvement could be to instead of extracting new features for each key frame, use the tracked points and relate them with 3D-2D feature correspondences with PnP [62], instead of 2D-2D matching. Another area where the time used for the graph optimisation could be decreased, is to reduce the number of optimised poses, e.g. by approximate marginalisation, as proposed by Stachniss and Kretzschmar in 2011 [64], which will maintain the sparsity pattern observed in SLAM.

5.2 Future work

A interesting area to explore to further improve the use of a camera for localisation and mapping, could be to classify objects in the images. This would be especially helpful for situations when objects, that are a known source for disturbance of the VO, like other traffic, that can be removed. This information could also be used for tracking of those objects, and construction of a map that can be used for other purposes than localisation. A solution for this is proposed by Song and Chandraker in 2014 [65], where the obtained information is also used for scale estimation.

Another interesting area to investigate are other approaches to the VO problem. One approach is to use direct methods, like the semi-dense method proposed by Engel *et al.* in 2014 [23]. Another approach for road vehicle localisation is to localise based on the lanes, like the solution proposed by Ziegler *et al.* in 2014 [61], where this is used in combination with feature based VO, and the methods are found to complement each others weaknesses.

To use a camera for localisation in a autonomous vehicle would probably require a map with classified objects, that is static and built offline, with the possibility to localise in real-time within this map. To achieve this, a method similar to loop closure would have to be developed. Probably it would be more suitable to use 3D-2D correspondences, rather than 2D-2D, like in FAB-MAP, but with similar concepts for fast feature matching.

Bibliography

[1] E. Krug, "Decade of action for road safety 2011-2020", *Injury*, vol. 43, no. 1, pp. 6–7, 2012.

[2] S. Singh, "Critical reasons for crashes investigated in the National Motor Vehicle Crash Causation Survey", Tech. Rep. February, 2015, pp. 1–2.

[3] R. Sims *et al.*, "Transport", in *IPCC Fifth Assessment Report*, 2014, pp. 599–670.

[4] A. Spalanzani, J. Rios-Martinez, C. Laugier, and S. Lee, *Handbook of Intelligent Vehicles*, A. Eskandarian, Ed. Springer, 2012.

[5] (2015). Google self-driving car project, [Online]. Available: `http://www.google.com/selfdrivingcar/` (visited on 06/22/2015).

[6] J. Ziegler *et al.*, "Making Bertha Drive — An Autonomous Journey on a Historic Route", *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 2, pp. 8–20, 2014.

[7] P. Pfaff, R. Triebel, C. Stachniss, P. Lamon, W. Burgard, and R. Siegwart, "Towards mapping of cities", in *IEEE International Conference on Robotics and Automation, ICRA'07*, 2007, pp. 4807–4813.

[8] E. Olson, "Recognizing places using spectrally clustered local matches", *Robotics and Autonomous Systems*, vol. 57, pp. 1157–1172, 2009.

[9] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[10] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset", *The International Journal of Robotics Research*, vol. 32, pp. 1231–1237, 2013.

[11] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry", in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR'04.*, vol. 1, 2004.

[12] M. Schreiber, C. Knöppel, and U. Franke, "LaneLoc: Lane marking based localization using highly accurate maps", in *IEEE Intelligent Vehicles Symposium, IVS'13*, 2013, pp. 449–454.

[13] J. Engel, J. Sturm, and D. Cremers, "Semi-dense visual odometry for a monocular camera", in *IEEE International Conference on Computer Vision, ICCV'13*, 2013, pp. 1449–1456.

[14] A. J. Davison, “Real-time simultaneous localisation and mapping with a single camera”, in *IEEE International Conference on Computer Vision, ICCV’03*, vol. 2, 2003, pp. 1403–1410.

[15] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, “FastSLAM: A factored solution to the simultaneous localization and mapping problem”, in *AAAI Conference on Artificial Intelligence, AAAI’02*, vol. 68, 2002, pp. 593–598.

[16] M. Montemerlo, S. Thrun, D. Roller, and B. Wegbreit, “FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges”, in *International Joint Conference on Artificial Intelligence, IJCAI’03*, 2003, pp. 1151–1156.

[17] M. J. Milford, G. F. Wyeth, and D. Prasser, “RatSLAM: A Hippocampal Model for Simultaneous Localization and Mapping”, in *IEEE International Conference on Robotics and Automation, ICRA’04*, 2004, pp. 403–408.

[18] G. Klein and D. Murray, “Parallel tracking and mapping for small AR workspaces”, in *IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR’07*, 2007.

[19] H. Lim, J. Lim, and H. J. Kim, “Real-Time 6-DOF Monocular Visual SLAM in a Large-Scale Environment”, in *IEEE International Conference on Robotics and Automation, ICRA’14*, 2014, pp. 1532–1539.

[20] M. Cummins and P. Newman, “FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance”, *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 647–665, 2008.

[21] ——, “Appearance-only SLAM at large scale with FAB-MAP 2.0”, *The International Journal of Robotics Research*, vol. 30, no. 9, pp. 1100–1123, 2011.

[22] M. J. Milford and G. F. Wyeth, “SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights”, *IEEE International Conference on Robotics and Automation, ICRA’13*, pp. 1643–1649, 2012.

[23] J. Engel, T. Schöps, and D. Cremers, “LSD-SLAM: Large-Scale Direct Monocular SLAM”, in *European Conference on Computer Vision, ECCV’14*, 2014, pp. 1–16.

[24] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004.

[25] A. Einstein, “On a Heuristic Viewpoint Concerning the Production and Transformation of Light”, *Annalen der Physik*, vol. 17, no. 6, pp. 132–148, 1905.

[26] J. Heikkilä and O. Silvén, “A four-step camera calibration procedure with implicit image correction”, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR’97*, 1997.

[27] Z. Zhang, “A flexible new technique for camera calibration”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[28] D. Scaramuzza and F. Fraundorfer, “Visual Odometry : Part I: The First 30 Years and Fundamentals”, *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[29] G. Bradski, “The OpenCV library”, *Dr. Dobb’s Journal of Software Tools*, vol. 25, no. 11, pp. 120, 122–125, Nov. 2000.

[30] R. I. Hartley, “In defense of the eight-point algorithm”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 6, pp. 580–593, 1997.

[31] D. Nistér, “An efficient solution to the five-point relative pose problem”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 756–770, 2004.

[32] F. Fraundorfer and D. Scaramuzza, “Visual odometry: Part II: Matching, robustness, optimization, and applications”, *IEEE Robotics and Automation Magazine*, vol. 19, no. 2, pp. 78–90, 2012.

[33] C. Harris and M. Stephens, “A Combined Corner and Edge Detector”, in *Alvey Vision Conference*, 1988, pp. 147–151.

[34] E. Rosten and T. Drummond, “Machine learning for high-speed corner detection”, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3951 LNCS, 2006, pp. 430–443.

[35] J. S. J. Shi and C. Tomasi, “Good features to track”, *Computer Vision and Pattern Recognition, CVPR’94*, pp. 593–600, 1994.

[36] D. G. Lowe, “Distinctive image features from scale-invariant keypoints”, *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[37] H. Bay, T. Tuytelaars, and L. Van Gool, “SURF: Speeded up robust features”, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3951 LNCS, pp. 404–417, 2006.

[38] M. Agrawal, K. Konolige, and M. R. Blas, “CenSurE: Center Surround Extremas for Realtime Feature Detection and Matching”, in *Computer Vision – ECCV 2008 SE - 8*, ser. Lecture Notes in Computer Science PART 4, D. Forsyth, P. Torr, and A. Zisserman, Eds., vol. 5305, Springer Berlin Heidelberg, 2008, pp. 102–115.

[39] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, “BRIEF: Binary robust independent elementary features”, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6314 LNCS, pp. 778–792, 2010.

[40] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB: An efficient alternative to SIFT or SURF”, in *IEEE International Conference on Computer Vision, ICCV’11*, 2011, pp. 2564–2571.

[41] S. Leutenegger, M. Chli, and R. Y. Siegwart, “BRISK: Binary Robust invariant scalable keypoints”, in *IEEE International Conference on Computer Vision, ICCV’11*, 2011, pp. 2548–2555.

[42] M. A. Fischler and R. C. Bolles, “Random Sample Consensus: A Paradigm for Model Fitting with Applicatlons to Image Analysis and Automated Cartography”, *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[43] P. D. Sampson, “Fitting conic sections to “very scattered” data: An iterative refinement of the bookstein algorithm”, *Computer Graphics and Image Processing*, vol. 18, no. 1, pp. 97–108, 1982.

[44] D. Scaramuzza, F. Fraundorfer, and R. Siegwart, “Real-time monocular visual odometry for on-road vehicles with 1-point RANSAC”, *IEEE International Conference on Robotics and Automation, ICRA’09*, pp. 4293–4299, 2009.

[45] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots.* MIT Press, 2004.

[46] G. Sibley, “Relative Bundle Adjustment”, Tech. Rep., 2009, pp. 1–26.

[47] S. Klose, P. Heise, and A. Knoll, “Efficient compositional approaches for real-time robust direct visual odometry from RGB-D data”, *IEEE International Conference on Intelligent Robots and Systems, IROS’13*, pp. 1100–1106, 2013.

[48] M. Pizzoli, C. Forster, and D. Scaramuzza, “REMODE : Probabilistic , Monocular Dense Reconstruction in Real Time”, *IEEE International Conference on Robotics and Automation, ICRA’14*, 2014.

[49] J. Stühmer, S. Gumhold, and D. Cremers, “Real-time dense geometry from a handheld camera”, *Pattern Recognition*, no. x, pp. 11–20, 2010.

[50] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, “A tutorial on graph-based SLAM”, *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[51] M. Dissanayake, P. Newmann, S. Clark, H. F. Durrant-White, and M. Corsba, “A solution to the Simultaneous localisation and Map Building problem”, *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229–257, 2001.

[52] H. Durrant-Whyte and T. Bailey, “Simultaneous localization and mapping”, *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–116, 2006.

[53] B. Williams, M. Cummins, J. Neira, P. Newman, I. Reid, and J. Tardós, “A comparison of loop closing techniques in monocular SLAM”, *Robotics and Autonomous Systems*, vol. 57, no. 12, pp. 1188–1197, 2009.

[54] L. Clemente, A. Davison, I. Reid, J. Neira, and J. D. Tardós, “Mapping Large Loops with a Single Hand-Held Camera”, in *Robotics: Science and Systems Conference*, 2007, pp. 297–304.

[55] E. Eade and T. Drummond, “Unified Loop Closing and Recovery for Real Time Monocular SLAM”, in *British Machine Vision Conference, BVMC’08*, 2008.

[56] B. Williams, M. Cummins, J. Neira, P. Newman, I. Reid, and J. Tardós, “An image-to-map loop closing method for monocular SLAM”, in *IEEE/RSJ International Conference on. Intelligent Robots and Systems, IROS’08*, 2008, pp. 2053–2059.

[57] N. Sünderhauf, P. Neubert, and P. Protzel, “Are we there yet? challenging seqslam on a 3000 km journey across all four seasons”, in *IEEE International Conference on Robotics and Automation, ICRA'13*, 2013.

[58] A. Glover, W. Maddern, M. Warren, S. Reid, M. Milford, and G. Wyeth, “OpenFABMAP: An open source toolbox for appearance-based loop closure detection”, in *IEEE International Conference on Robotics and Automation, ICRA'12*, 2012, pp. 4730–4735.

[59] H. Strasdat, J. M. M. Montiel, and A. J. Davison, “Real-time monocular SLAM: Why filter?”, in *IEEE International Conference on Robotics and Automation, ICRA'10*, 2010, pp. 2657–2664.

[60] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, “G2o: A general framework for graph optimization”, in *IEEE International Conference on Robotics and Automation, ICRA'11*, 2011, pp. 3607–3613.

[61] J. Ziegler *et al.*, “Video based localization for Bertha”, in *IEEE Intelligent Vehicles Symposium, IV'14*, 2014, pp. 1231–1238.

[62] H. Badino, A. Yamamoto, and T. Kanade, “Visual odometry by multi-frame feature integration”, in *IEEE International Conference on Computer Vision, ICCV'13*, 2013, pp. 222–229.

[63] M. Persson, T. Piccini, R. Mester, and M. Felsberg, “Robust stereo visual odometry from monocular techniques”, in *IEEE Intelligent Vehicles Symposium*, 2015.

[64] C. Stachniss and H. Kretzschmar, “Pose graph compression for laser-based slam”, in *International Symposium of Robotics Research, ISRR'11*, 2011, pp. 1–16.

[65] S. Song and M. Chandraker, “Robust Scale Estimation in Real-Time Monocular SFM for Autonomous Driving”, in *IEEE International Conference on Intelligent Robots and Systems, IROS'14*, 2014, pp. 1566–1573.

A

Parameter List

Name	Value
KLTNPoints	300
KLTFASTThreshold	15
KLTRANSAC8PointThresh	0.01
KLTRANSAC8PointIterations	500
KLTNWin	44
KLTWinThreshold	0.5
KLTWinWidthMargin	100
KLTWinBottomMargin	60
KLTPMotionDistanceThreshold	3
KLTMotionRatioThreshold	0.9
PixelVar	1
DistanceVar	10^{-5}
BAWindow	8
BAIterations	20
TransVar	10^{-4}
RotVar	10^{-5}
FMPzGe	0.39
FMPzGNe	0
FMStarThreshold	60
FMStarIterations	10
FMStarNMax	250
FMStarNMax	350
FMMatchThreshold	0.99
LCDistThreshold	100
LCFASTThreshold	50
LCBRISKDistanceThreshold	45
LCRANSAC8PointThreshold	0.08
LCRANSAC8PointIterations	500
LCNMatchedTreshold	24
LCIterations	20

Table A.1: List of user tunable parameters and the values used in the result section.