Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Fredrik Bjerkås

Robust GNSS-Aided Visual Simultaneous Localization and Mapping for an Unmanned Ground Vehicle

Master's thesis in Cybernetics and Robotics

April 2019

NTNU
Norwegian University of Science and Technology

FFI Forsvarets forskningsinstitutt

Fredrik Bjerkås

Robust GNSS-Aided Visual Simultaneous Localization and Mapping for an Unmanned Ground Vehicle

Master's thesis in Cybernetics and Robotics
April 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Problem Description

Background

The Norwegian Defence Research Institute (FFI) is developing an Unmanned Ground Vehicle (UGV), called Olav, that should be capable of long-term autonomous driving in challenging terrain environments. For the UGV to be truly autonomous, it needs to have a map of the environment it is operating in, and it needs to localize itself in that map. Visual simultaneous localization and mapping (V-SLAM) makes use of on-board cameras to solve this problem, but suffers from other problems such as drift between the estimated and real trajectory, and re-initialization issues if tracking is lost.

Task

Investigate ways to utilize other available sensors on Olav to correct for accumulated drift in the V-SLAM trajectory estimates, and aid the re-initialization of the system in the case of tracking failure without discarding previously built sections of the map.

Abstract

The Norwegian Defense Research Establishment (FFI) is currently developing an Unmanned Ground Vehicle (UGV) named Olav which will operate as part of the next generation military base defence system as an effort to improve reliability and minimize risk to human personnel. Olav should be capable of performing Intelligence, Surveillance and Reconnaissance (ISR) missions autonomously, which implies the need for it to be able to estimate its position reliably and accurately relative to a global reference frame.

In this project, a real-time capable stereo Visual Simultaneous Localization and Mapping (V-SLAM) system which incorporates Global Navigation Satellite System (GNSS) measurements was developed. It combines locally accurate estimates from V-SLAM with the globally accurate GNSS sensor, enabling the host platform to report accurate trajectory estimates relative to a global reference frame. The developed system was tested on datasets recorded from Olav, but because of issues with the camera calibration, the main portion of the testing was performed on the publicly available KITTI dataset [25]. Testing showed that the Visual Odometry (VO) module outperforms the popular stereo VO system LIBVISO2 [28] on most tested datasets, while achieving a much higher frame-rate. Testing also showed that the system is able to recover from tracking failures without discarding information from previously explored areas, and is robust to false positive loop closure detections and GNSS measurements corrupted by multipath artifacts.

Sammendrag

Forsvarets forskningsinstitutt (FFI) utvikler for tiden et ubemannet bakkekjøretøy, kalt Olav, som skal fungere som en del av fremtidens baseforsvar. Olav skal være i stand til å utføre oppklaringsoppdrag autonomt, som krever at den må kunne estimere sin posisjon pålitelig og nøyaktig relativt til en global referanseramme.

I denne oppgaven presenteres et navigasjonssystem basert på visuell simultan lokalisering og kartlegging (V-SLAM) som kan benytte GNSS målinger. Dette kombinerer de lokalt nøyaktige estimatene fra V-SLAM med den globalt nøyaktige GNSS sensoren, som setter vertsplatformen i stand til å presist rapportere sin globale posisjon. Systemet ble testet på datasett produsert av Olav, men på grunn av problemer med kamerakalibreringen ble hoveddelen av testingen utført på det offentlig tilgjengelige KITTI datasettet [25]. Testresultatene viser at kameraodometrimodulen produserer bedre estimater enn LIBVISO2 algoritmen [28] på majoriteten av de testede datasettene, samtidig som den oppnår en høyere bilderate. Testresultatene viser også at systemet er i stand til å fortsette å operere i tilfeller der kameraodometrien feiler, og er robust mot falske løkkedeteksjoner og GNSS målinger som er påvirket av flerveisinterferens.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU), and was written in collaboration with FFI. It builds on the knowledge acquired from my semester project on visual navigation for Olav. For this project, FFI has provided me with datasets from previous missions with Olav to use for testing. I would like to thank my co-supervisor Trym Vegard Haavardsholm at FFI for his guidance during the project period.

Fredrik Bjerkås

Contents

List of Figures

List of Tables

Abbreviations

ATE	Absolute Trajectory Error
BA	Bundle Adjustment
BoW	Bag of Words
BRIEF	Binary Robust Elementary Features
DAG	Directed Acyclic Graph
DBN	Dynamic Bayesian Network
DBOW2	Dynamic Bag of Words 2
DOF	Degrees of Freedom
DSO	Direct Sparse Odometry
ECEF	Earth-Centered, Earth-Fixed
EKF	Extended Kalman Filter
ENU	East, North, Up
FAB-MAP 2.0	Fast Appearance-Based Mapping 2.0
FAST	Features from Accelerated Segment Test
FFI	the Norwegian Defense Research Establishment
GN	Gauss-Newton
GNSS	Global Navigation Satellite System
GPU	Graphics Processing Unit

GTSAM	Georgia Tech Smoothing and Mapping library
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
iSAM2	Incremental Smoothing and Mapping 2
ISR	Intelligence, Surveillance and Reconnaissance
KF	Kalman Filter
Lat/Long	Latitude/Longitude
LIBVISO2	Library for Visual Odometry 2
LIDAR	Light Detection and Ranging sensor
LM	Levenberg-Marquardt
LSD-SLAM	Large Scale Direct SLAM
MAP	Maximum a Posteriori
ML	Maximum Likelihood
MRF	Markov Random Field
NTNU	the Norwegian University of Science and Technology
OpenCV	Open Source Computer Vision Library
ORB	Oriented FAST and Rotated BRIEF
ORB-SLAM2	"An open-source SLAM system for monocular, stereo and RGB-D cameras"
PGM	Probibalistic Graphical Model
RANSAC	Random Sample Consensus
ROS	Robot Operating System

RTK	Real-Time Kinematic
SLAM	Simultaneous Localization and Mapping
SOFT	Stereo Odometry Based on Feature Selection and Tracking
SURF	Speeded Up Robust Features
SVD	Singular Value Decomposition
tf-idf	Term Frequency-Inverse Document Frequency
UGV	Unmanned Ground Vehicle
UKF	Unscented Kalman Filter
V-SLAM	Visual Simultaneous Localization and Mapping
VO	Visual Odometry
WGS-84	World Geodetic System 1984

Nomenclature

λ	Scalar variable, denoted with a lowercase letter.
$\mathbf{a}$	A column vector, denoted with a lowercase bold letter.
$\tilde{\mathbf{a}}$	A column vector expressed in homogeneous coordinates.
$\mathbf{A}$	A matrix, denoted with an uppercase bold letter.
$\mathbf{I}_m$	The $m \times m$ identity matrix.
$\mathbf{0}_{m\times n}$	A zero matrix or vector with m rows and n columns.
$\lVert\mathbf{x}\rVert_\Sigma$	The weighted L_2 norm $\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x}$
$\mathcal{F}_{C_i}$	The camera frame. $i \in \{l, r\}$ denotes left or right camera.
$\mathcal{F}_{B_t}$	The body coordinate frame at time t.
$\mathcal{F}_{\hat{B}_t}$	The body coordinate keyframe at time t.
$\mathcal{F}_W$	The fixed world coordinate frame.
$\mathbf{T}_{B_t}^{W}$	A homogeneous transformation matrix from frame $\mathcal{F}_{B_t}$ to frame $\mathcal{F}_W$
$\mathbf{R}_{B_t}^{W}$	A rotation from frame $\mathcal{F}_{B_t}$ to frame $\mathcal{F}_W$
$\mathbf{R}_x(\theta)$	A rotation of θ radians around the x-axis.

$\mathbf{t}^{W_t}$	A translation vector from the origin of frame $\mathcal{F}_{B_t}$ to frame $\mathcal{F}_W$.
$\mathbf{l}^W$	A point vector relative to frame $\mathcal{F}_W$.
$\mathbf{u} = (u, v)^T$	Image coordinate in pixels.
$\hat{\mathbf{u}} = (\hat{u}, \hat{v})^T$	Image coordinates in meters.

Chapter 1

Introduction

FFI is currently developing an UGV called Olav, depicted in figure 1.1, that should be capable of performing military ISR missions autonomously. Automating these kinds of missions could potentially bring several benefits such as reduced risk to personnel, cost benefits, and higher reliability as a machine can operate 24/7 without getting tired or bored. However, there are many challenges associated with developing this capability as the military domain has some specific challenges compared to commercial autonomous cars. In addition to handle traffic and urban areas, Olav must be able to navigate in rough terrain environments associated with military areas. Also, since the UGV is controlled by a remote base-station, it has to know and report its position reliably and accurately.

1.1 Motivation

For an UGV, to be truly autonomous, it needs to localize itself in the environment it is operating in. A common way of performing localization is by using a GNSS which has the advantage of being able to localize the UGV globally. However, relying on GNSS as a single sensor for localization provides low redundancy, as GNSS data might not always be available or be severely degraded. This could result from a technical failure,

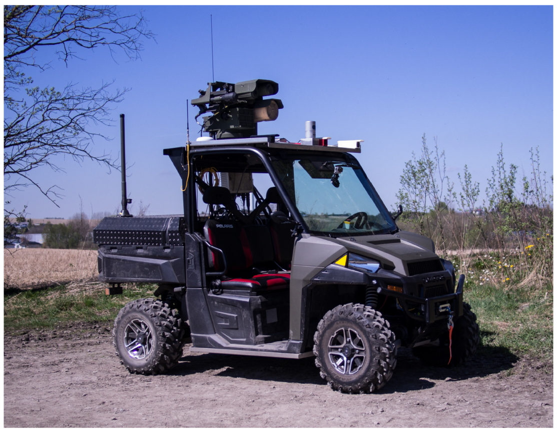


Figure 1.1: Olav, a Polaris Ranger XP 900 EPS is the autonomous platform.

a war-scenario where GNSS satellites are taken down, or in a dense forest environment where GNSS signals are blocked or degraded by multipath artifacts. All these scenarios are relevant for the UGV considered in this project. It is therefore desirable to be able to perform localization by processing information from on-board sensors only, while still utilizing the GNSS data when it is available.

The simplest method to localize a robot without GNSS is to process sensor information and compute incremental motion, and this process is known as odometry. Some common sensors used for odometry are wheel encoders, Inertial Measurement Units (IMUs), Light Detection and Ranging sensors (LIDARs), or visual sensors, e.g. cameras. Odometry allows to estimate the trajectory of the UGV, but the estimate will inevitably accumulate error and deviate more and more from the real trajectory. Odometry techniques are therefore suitable for short-term motion estimation, but for long-term operation in the same environment one would desire to have a map that allows drift-free localization. Mapping is the process of creating a map from on-board sensors given that localization is known. However, in order to localize the robot, we have introduced the necessity of having a map. This is a chicken-and-the-egg problem known as Simultaneous Localization and Mapping (SLAM), where we aim to solve localization and mapping problems simultaneously.

In order to map the environment, the sensors needs to obtain information from the external world. These types of sensors are known as exteroceptive sensors, and can provide distances to objects, intensity of ambient light, bearing to magnetic north, etc. Among the exteroceptive sensors that can be used for SLAM, cameras are one of the most promising. They are passive sensors in that they don't influence the environment by emitting energy, e.g. light, and observe the world as it is. They also provide dense information of the viewed scene, in comparison to LIDARs which only provides sparse point clouds. A dense sensor is useful for applications such as 3D reconstruction and object recognition, that can be useful for other aspects of an autonomous system, and also place recognition which is a key part of a SLAM system. SLAM performed with cameras as the main sensor is known as V-SLAM, but will hereby be referenced to simply as SLAM.

With a built map, the UGV can bound its odometry drift during GNSS outages by recognizing previously visited locations, and thereby "close the loop". If the map is built with available GNSS data, the accuracy of the map is high and the location of the robot at the time of loop closure would be known with a high probability. If the map is built without available GNSS data, the odometry error would still be bounded to the uncertainty of the robots location at the time when the map was built, which would be lower than the uncertainty of the robots location just before a loop closure event occurred.

Robustness of the navigation system is of high importance in a real-world application. A failure in one or more sensors should not result in a navigation failure, but should rather provide the best possible estimates based on the sensors currently available at all times. Using only a traditional Inertial Navigation System (INS), a GNSS outage would result in navigation errors which would grow unbounded over time. This project aims to incorporate cameras into the navigation system such that we can map the environment in order to provide bounds on the localization error.

1.2 Earlier Work

1.2.1 Visual Odometry

The problem of estimating a vehicle's ego-motion based on visual input was started in the early 1980s and was described by Moravec [49]. Most of the early research in this field was done for planetary rovers, and was motivated the need to provide all-terrain rovers with the capability to measure their 6-Degrees of Freedom (DOF) motion in the presence of wheel slippage and rough terrains. This method of doing odometry with cameras later became known as Visual Odometry (VO) after the work by Nistér [54], who coined the term after providing the first real-time capable VO system. A tutorial by Scaramuzza and Fraundofer [61], [20] provides an in depth review of the history of VO as well as the key components of modern VO systems which can essentially be seen as SLAM systems without the ability to detect previously visited locations.

VO algorithms can be divided into two categories, namely feature based, which extracts and matches important keypoints in the images to retrieve the relative pose between them, and direct methods, which uses pixel intensities directly to achieve the same result.

Feature based approaches is the category with the richest literature, because of the limited computation involved when dealing with sparse keypoints, allowing for real-time execution on a wide range of platforms. One well known real-time capable feature based approach is Libviso2 by Geiger et al. [28]. This approach matches keypoint features in a circular fashion between consecutive stereo images, and estimates motion between the consecutive frames by minimizing the reprojection error of the sparse feature matches. The circular matching approach has also been used in the work of Cvišsić et al. [8], [9], which currently is the best performing VO algorithm on the public KITTI dataset [25].

In recent years, there have been many algorithms focusing on the direct approach. One of such methods that have gained a lot of attention is Direct Sparse Odometry (DSO) [14] which has achieved remarkable results on public datasets. The downside of these kinds of algorithms are that they are very sensitive to geometric distortions resulting from imperfect calibration, and they also require photmetrically calibrated datasets.

1.2.2 Visual SLAM

According to Durrant-Whyte and Bailey in their two-part tutorial [13], [2], the research on SLAM dates back to 1986, when the idea of using estimation-theoretic methods for robot localization and mapping were first discussed in the IEEE Robotics and Automation Conference held in San Francisco. Yet, it was not until the 1995 International Symposium on Robotics Research that the structure of the SLAM problem, the convergence result and the coining of the acronym SLAM were presented.

The tutorial of Durrant-Whyte and Bailey reviews solutions to the SLAM problem up until 2006, which mainly focused on Bayesian filtering methods, such as the Extended Kalman Filter (EKF), particle filters, the Unscented Kalman Filter (UKF), and information filters. Lately, the focus of the SLAM research community has shifted towards optimization-based approaches, as these have proven to be more accurate and efficient than the original approaches based on non-linear filtering [62]. The reason for this is that optimization, or "smoothing", approaches retains and allows for re-linearization of past states. This is contrary to filtering techniques which commit to a linearization point when marginalizing, leading to a gradual build-up of linearization errors which leads to drift and possible inconsistencies. Cadena et al. [4] more recently presented a review of modern algorithms which mainly approaches SLAM as a maximum a-posteriori optimization problem.

In recent years, one of the most popular SLAM algorithms is "An open-source SLAM system for monocular, stereo and RGB-D cameras" (ORB-SLAM2) [51] which combines feature based VO with the Dynamic Bag of Words 2 (DBOW2) [23] place recognition library. The system is highly multi-threaded and optimizes poses and landmarks over a local and global graph, yielding accurate results.

1.2.3 Sensor Fusion

Combining measurements from many different sensors into a best possible estimate is known as information fusion, where the combined estimate has less uncertainty than if all the sensors were used individually. Traditionally this has been done using filtering techniques using different variants of the Kalman Filter (KF) (see for example [63]), but as in visual SLAM, new techniques based on optimization/smoothing have emerged. Many of them use the formalism of factor graphs, described in section 6.2, to reason about the interdependence among the involved variables.

Indelman et al. [33] describes how information from different sensors operating asynchronously at different rates can be incorporated into a factor graph formulation in a natural way. This method, based on a recently developed incremental smoother [34], automatically determines the number of states to recompute at each step, effectively acting as an adaptive fixed-lag smoother. This yields an efficient and general framework for information fusion. To reduce the number of variables required in the optimization when dealing with the high rate IMU sensor, Forster et al. describes how to summarize IMU measurements into a single factor between keyframes [19]. Kaess et al. [36] does filtering and smoothing in parallel in order to filter out high frequency measurements, while retaining important states which can be re-linearized to better accommodate for loop closures.

1.3 Assumptions

This section defines assumptions that will be adopted throughout the thesis.

Assumption 1. The navigation system must be able to operate in real-time.

Assumption 2. The navigation system must be able to operate over long periods of time.

Assumption 3. The navigation system must be able to operate over large areas.

Assumption 4. The navigation system must be able to operate in a wide range of environments, such as terrain and urban.

Assumption 5. The navigation system must be able to operate in areas with limited GNSS coverage.

Assumption 6. Olav is equipped with a Velodyne HDL-32E 3D LIDAR which is an active sensor providing sparse but accurate sensor suitable for navigation and mapping. However, to investigate the potential of using cameras in the navigation system, LIDAR based algorithms was not considered in this project.

Remark 1. Assumption 4 implies that the odometry must be able to describe motion with 6-DOF.

1.4 Contributions

- The main contribution of this thesis is a real-time capable V-SLAM system which incorporates GNSS measurements. This combines locally accurate VO from stereo cameras, with the globally accurate GNSS sensor. The system is able to work without a GNSS sensor for long periods of time, and perform re-localization if the VO estimation fails, due to the systems ability to close loops.
- A Graphics Processing Unit (GPU) accelerated stereo VO algorithm based on Library for Visual Odometry 2 (LIBVISO2) [28], the Stereo Odometry Based on Feature Selection and Tracking (SOFT) [8] algorithm and the work by Manthe et al. [45].
- A back-end optimization framework based on Incremental Smoothing and Mapping 2 (iSAM2) [34] which uses switchable constraints [66] to robustly handle false positive loop closures and GNSS measurements corrupted by multipath artifacts.
- A method for finding the relative pose between earth and body frame based on GNSS measurements only.
- A modular implementation which makes it easy to replace, improve and add modules. This way, other sensors can be incorporated at a later stage.
- A map and frame viewer which can be used to visually interface the system during testing.

1.5 Outline

After this introductory chapter, the outline of the thesis is as follows: Chapter 2 presents the sensor setup on Olav, and the datasets used in this project. Some relevant background theory on geometry is provided in chapter 3, which lays the foundation to discuss the stereo camera model and calibration in chapter 4, as well as the SLAM problem. The topic of SLAM is divided into two chapters. Chapter 5 reviews VO and place recognition, where after each topic, a module to incorporate into the final system is presented after a discussion. Chapter 6 provides a probabilistic formulation of SLAM, and how it can be formalized in a factor graph which can incorporate different sensors. An optimization framework is chosen and presented in section 6.6.2, based on a discussion of available frameworks in section 6.6.1. The system implementation is described in chapter 7, followed by experiments and results in chapter 8. Lastly, chapter 9 draws conclusions and suggests directions for further work.

Chapter 2

Sensor Setup

In this chapter, I will do a review of the different sensors available on Olav. This will lay the foundations for which algorithms to implement in the navigation system in later chapters.

2.1 The Vehicle

Olav, the autonomous platform that the navigation algorithm is going to run on, is a Polaris Ranger XP 900 EPS modified to enable autonomous driving [46], and is depicted in figure 1.1. The sensors installed on Olav, which are of importance to this project, are three forward facing Point Grey Grasshopper3 cameras (two gray-scale stereo cameras, and one centered color camera) depicted in figure 2.1, a Honeywell HG9900 IMU, and a Trimble SPS855 GNSS receiver which supports Real-Time Kinematic (RTK) and Omni-Star correction streams to enhance accuracy. The GNSS uses all currently available satellite signals including L1, L2 and the modernized L2C code. Currently the system estimates global position and orientation with FFI's in-house developed INS [30] which fuse the IMU and GNSS measurements in a Kalman filter. Additionally, Olav is equipped with two NVIDIA GeForce GTX 980 Ti GPUs which can be utilized to speed up computation of image streams.

Software-wise, the platform runs Robot Operating System (ROS) [57], which is a publisher-subscriber framework where nodes implement core functionality. The nodes subscribe on topics from other nodes, process the received information and then publish their results.

2.2 The Cameras

The Grasshopper3 GS3-U3-91S6M-C gray-scale stereo cameras has 9.1 mega-pixels and are equipped with global shutters which eliminates spatial and temporal aliasing problems associated with conventional rolling shutters. The cameras has fixed apertures, but the shutter speed is adjusted automatically to the lighting conditions, albeit with some delay when the lighting conditions change rapidly. In the provided dataset, the images has a resolution of 1688×1352 pixels, and a frame-rate of 6 fps.

Figure 2.1: Point Grey Grasshopper3 Camera

Chapter 3

Geometry Fundamentals

This chapter presents geometry fundamentals on homogeneous coordinates and rigid body motion, necessary for describing the perspective camera model in chapter 4, and Olav's motion through the environment. Additionally, this chapter discusses the relationship between the different coordinate frames involved when working with different sensors, and how to do to handle incremental updates on rotation- and homogeneous transformation matrices which is used when discussing optimization in section 6.6.

3.1 Homogeneous Coordinates

Homogeneous coordinates are a system of coordinates used in projective geometry and rigid body kinematics. They have the advantage that the coordinates of points, including points at infinity, can be represented by finite coordinates. They also simplify the algebra when working with projective transformations, extensively used in computer vision.

A point in the Euclidean plane may be represented by the pair of coordinates (x_1, x_2) in $\mathbb{R}^2$. Considering $\mathbb{R}^2$ as a vector space, the coordinate pair is a point identified as a vector $\mathbf{x} = (x_1, x_2)^T$. We can represent this same point in homogeneous coordinates, denoted by $\tilde{\mathbf{x}}$ (the tilde notation is used for homogeneous coordinates throughout this thesis), by adding a third dimension such that $\tilde{\mathbf{x}} = (x_1, x_2, 1)^T \in \mathbb{P}^2$ where $\mathbb{P}^2$ is called the projective plane. For any non-zero λ, $\tilde{\mathbf{x}} = \lambda\tilde{\mathbf{x}}$ holds, because an arbitrary homogeneous vector representative of a point on the form $\tilde{\mathbf{x}} = (x_1, x_2, x_3)^T$, represents the point $(x_1/x_3, x_2/x_3)$ in $\mathbb{R}^2$.

With this, points at infinity can now be represented as $\tilde{\mathbf{x}} = (x_1, x_2, 0)^T$ (which is not part of the Euclidean plane), defined as the limit of a point that moves in the direction specified by the ratio $x_1 : x_2$. Parallel lines in the Euclidean plane is said to intersect at a point at infinity corresponding to their common direction.

The notion of homogeneous coordinates can also be extended to 3D space, where the coordinates (x_1, x_2, x_3) in $\mathbb{R}^3$ is a point vector when $\mathbb{R}^3$ is considered to be a vector space. The homogeneous representation of the point vector can then be expressed as $\tilde{\mathbf{x}} = (x_1, x_2, x_3, 1)^T \in \mathbb{P}^3$.

3.2 Rigid Body Kinematics

3.2.1 Pose Representation

Since Olav is a terrain vehicle operating in uneven environments, we have to describe its motion with 6 DOF. If we attach a coordinate frame $\mathcal{F}_B$ to the moving vehicle body, the pose of the body frame relative to the fixed world frame $\mathcal{F}_W$ at time $t \in \{0, \dots, T\}$ can be described by the matrix

$$\mathbf{T}^W_{B_t} = \begin{bmatrix} \mathbf{R}^W_{B_t} & \mathbf{t}^{W_t} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t^{W_t}_x \\ r_{21} & r_{22} & r_{23} & t^{W_t}_y \\ r_{31} & r_{32} & r_{33} & t^{W_t}_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \in SE(3) \tag{3.1}$$

which is a homogeneous transformation of the projective space $\mathbb{P}^3$. The transformation is homogeneous because $\mathbf{T}^A_{B_t} = \lambda\mathbf{T}^W_{B_t}\ \forall\lambda \in \mathbb{R} \setminus \{0\}$. The Special Euclidean group $SE(3)$ is the set of all 3×3 transformation matrices defined as

$$SE(3) = \left\{ \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_{1\times3} & 1 \end{pmatrix} \middle| \mathbf{R} \in SO(3), \mathbf{t} \in \mathbb{R}^3 \right\} \tag{3.2}$$

and $SO(3)$ is the Special Orthogonal group, i.e. the set

$$SO(3) = \{\mathbf{R} \in \mathbb{R}^{3\times3} | \mathbf{R}^T\mathbf{R} = \mathbf{I}_3, det(\mathbf{R}) = 1\} \tag{3.3}$$

which is also known as the group of 3-dimensional rotation matrices. $SO(3)$ and $SE(3)$ are Lie groups with multiplication as group operations, and has respectively 3- and 6-DOF manifolds embedded in $\mathbb{R}^{3\times4}$ and $\mathbb{R}^{4\times4}$.

For a landmark represented by the point vector $\tilde{\mathbf{l}}^{B_t}$ expressed in homogeneous coordinates relative to the body frame at time t, the same point can be expressed in the world frame $\mathcal{F}_W$ by applying the following transformation:

$$\tilde{\mathbf{l}}^W = \mathbf{T}^W_{B_t}\tilde{\mathbf{l}}^{B_t}. \tag{3.4}$$

Note that $\tilde{\mathbf{l}}^W \neq \tilde{\mathbf{l}}^{B_t}$. Inversely; given $\mathbf{T}^W_{B_t}$ and $\tilde{\mathbf{l}}^W$, $\tilde{\mathbf{l}}^{B_t}$ can be found as:

$$\tilde{\mathbf{l}}^{B_t} = \mathbf{T}^{W\,-1}_{B_t}\tilde{\mathbf{l}}^W = \mathbf{T}^{B_t}_W\tilde{\mathbf{l}}^W = \begin{bmatrix} \mathbf{R}^{W\,T}_{B_t} & -\mathbf{R}^{W\,T}_{B_t}\mathbf{t}^{W_t} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} \tilde{\mathbf{l}}^W \tag{3.5}$$

Consecutive homogeneous transformations are convenient to work with, as the group operation of $SE(3)$ is matrix multiplication. For two consecutive vehicle poses [1] at time $t-1$ and t, the relative transformation between them can be found as:

$$\mathbf{T}^{B_{t-1}}_{B_t} = (\mathbf{T}^W_{B_{t-1}})^{-1}\mathbf{T}^W_{B_t} \tag{3.6}$$

The vehicle pose at time t can parametrized by the vector

$$\mathbf{x}_t = (x, y, z, \psi_{roll}, \psi_{pitch}, \psi_{yaw})^T, \tag{3.7}$$

where the the first three elements define the vehicle's translation $\mathbf{t}^W_{B_t} = (x, y, z)^T$

[1]The combination of position and orientation is referred to as the pose

relative to the world frame, and the last three parameters (given in radians) parametrizes a rotation composed of yaw, pitch and roll relative to the world frame:

$$\mathbf{R}_{B_t}^{W} = \mathbf{R}_z(\psi_{yaw})\mathbf{R}_y(\psi_{pitch})\mathbf{R}_x(\psi_{roll}) \tag{3.8}$$

where $\mathbf{R}_x$, $\mathbf{R}_y$, $\mathbf{R}_z$ denotes basic rotations around the x, y and z axes, respectively. There is a singularity problem associated with the yaw, pitch, roll parameterization when the first and third rotation axes align (when $p = \pi/2$ or $p = 3\pi/2$), but this is not relevant for an UGV. The notation $\mathbf{T}_{B_t}^{W} = \mathbf{T}(\mathbf{x}_t)$ is used to recover the corresponding transformation matrix from the pose vector.

3.2.2 Rotation Updates

Using 3×3 matrices to represent three-dimensional rotations has the advantage that rotations can be concatenated by matrix multiplication. A major drawback, however, is that with nine parameters and only three degrees of freedom, rotation matrices are over-parameterized, which is disadvantageous when doing numerical optimization. Also, using rotation matrices for incremental updates is not a good idea, since the result may produce a matrix which is not in $SO(3)$, and likewise for poses in $SE(3)$. We can solve this problem by using the associated Lie Algebra $\mathfrak{so}(3)$ of the Lie Group $SO(3)$.

A rotation can be parametrized by an axis-angle representation $\xi = \theta\omega \in \mathbb{R}^3$, where $\omega = (\omega_x, \omega_y, \omega_z)^T$ is a unit vector representing the axis of rotation, and θ is the angle magnitude. Using this representation, we can find 3-dimensional increments ξ onto proper rotations by using the exponential map defined by

$$SO(3) \ni exp(\hat{\xi}) = \mathbf{I}_3 + \frac{\sin\theta}{\theta}\hat{\xi} + \frac{1-\cos\theta}{\theta^2}\hat{\xi}^2 \tag{3.9}$$

where the hat operator maps elements from the 3-vector ξ to elements of the Lie algebra $\mathfrak{so}(3)$. In the case of $SO(3)$, the hat operator creates a skew symmetric matrix

$$\wedge : \mathbb{R}^3 \longrightarrow \mathfrak{so}(3) : \hat{\xi} \mapsto \begin{bmatrix} 0 & -\xi_z & \xi_y \\ \xi_z & 0 & -\xi_x \\ -\xi_y & \xi_x & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\omega_x & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \theta \in \mathfrak{so}(3). \tag{3.10}$$

Incremental rotation updates around an estimate $\mathbf{R}_0$ can now be computed as:

$$\mathbf{R}_0 \oplus \xi \triangleq \mathbf{R}_0 exp(\hat{\xi}). \tag{3.11}$$

3.2.3 Pose Updates

We can also think of the parameter θ as a finite time $\Delta\tau$ which we multiply with the angular velocity ω to obtain the incremental rotation. This is an easier analogy when dealing with incremental pose updates ξ which can be thought of as multiplying the angular velocity $\omega = (\omega_x, \omega_y, \omega_z)^T$ and translational velocity $v = (v_x, v_y, v_z)^T$ by a finite time $\Delta\tau$, i.e., $\xi \triangleq (\omega, v)^T \Delta\tau \in \mathbb{R}^6$. The hat operator defines a mapping from $\mathbb{R}^6$ to the Lie algebra $\mathfrak{se}(3)$ associated with the Lie group $SE(3)$:

$$\wedge : \mathbb{R}^6 \longrightarrow \mathfrak{se}(3) : \xi \mapsto \begin{bmatrix} \hat{\omega} & v \\ \mathbf{0}_{1\times3} & 0 \end{bmatrix} \Delta\tau \in \mathfrak{se}(3). \tag{3.12}$$

Incremental pose update around an estimate $\mathbf{T}_0$ can be found using the exponential map as in (3.11). The exponential map for $SE(3)$, as for $SO(3)$, has a close form solution, however when doing optimization on $SE(3)$ it is more computationally efficient to not use the exponential map, but rather to define a retraction $\mathcal{R}_\mathrm{T}$ which uses a simpler expression for the translational update. An incremental pose update around an estimate $\mathbf{T}_0$ can then be found as:

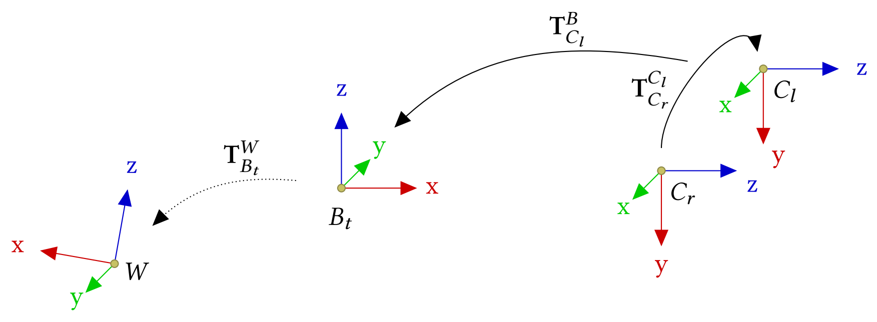


Figure 3.1: Transformations between the camera frames $\mathcal{F}_{C_r}$, $\mathcal{F}_{C_l}$ and the body frame $\mathcal{F}_{B_t}$ are constant over time, and represented with solid lines. The transformation between the body frame and the world frame $\mathcal{F}_W$ changes with time, and is represented with a dotted line.

$$\begin{aligned}\mathrm{T}_0 \oplus \xi \triangleq \mathcal{R}_{\mathrm{T}_0}(\xi) &\triangleq \begin{bmatrix} \mathbf{R}_0 & \mathbf{t}_0 \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} \begin{bmatrix} exp(\hat{\omega}\Delta\tau) & v\Delta\tau \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{R}_0 exp(\hat{\omega}\Delta\tau) & \mathbf{t}_0 + \mathbf{R}_0 v\Delta\tau \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix}.\end{aligned} \tag{3.13}$$

3.3 Working with Different Coordinate Systems

3.3.1 Visual Odometry

When performing frame-to-frame VO, we find the relative motion, $\mathrm{T}_{C_{l,t}}^{C_{l,t-1}}$, between (left) camera frames $\mathcal{F}_{C_{l,t-1}}$ and $\mathcal{F}_{C_{l,t}}$, $t \in \{0,\dots,T\}$. The procedure of finding this transformation is explained in subsection 5.3.3 and 5.3.4. The INS measurements that are published on the ROS network at time t are given in the INS frame, defined to coincide with the body frame $\mathcal{F}_{B_t}$. To easily combine these measurements, the camera frame is transformed into the body frame. For a pose estimate based on VO, the pose is given as

$$\mathbf{T}_{B_t}^{W} = \mathbf{T}_{B_0}^{W}\mathbf{T}_{C_l}^{B}\mathbf{T}_{C_{l,t}}^{C_{l,0}}\mathbf{T}_{B}^{C_l} \tag{3.14}$$

where

$$\mathbf{T}_{C_{l,t}}^{C_{l,0}} = \mathbf{T}_{C_{l,1}}^{C_{l,0}}\mathbf{T}_{C_{l,2}}^{C_{l,1}}\cdots\mathbf{T}_{C_{l,t-1}}^{C_{l,t-2}}\mathbf{T}_{C_{l,t}}^{C_{l,t-1}} \tag{3.15}$$

and the transformation $\mathbf{T}_{C_l}^{B}$ (as well as $\mathbf{T}_{C_r}^{B}$) and its inverse are constant over time, as illustrated in figure 3.1:

$$\mathbf{T}_{C_l}^{B} = \mathbf{T}_{C_{l,t}}^{B_t} = (\mathbf{T}_{B_t}^{C_{l,t}})^{-1} = (\mathbf{T}_{B}^{C_l})^{-1} \qquad \forall t \in \{0, \ldots, T\}. \tag{3.16}$$

This transformation, along with all other transformations between different coordinate frames defined on the vehicle are found by off-line calibration. When a mapping session begins, we have to find the transformation matrix $\mathbf{T}_{B_0}^{W}$ which is not straight forward as will be explained in the next section.

3.3.2 GNSS

The GNSS receiver on Olav publishes its measured position in geodetic latitude (lat.) φ, longitude (lon.) λ and height h relative to the World Geodetic System 1984 (WGS-84) reference ellipsoid. When doing odometry, however, it is more convenient to work with Cartesian coordinates, so in order to combine the two sensors, a local tangent plane on the reference ellipsoid is defined. The plane is illustrated in figure 3.2, and remains fixed throughout the navigation session. The reference frame $\mathcal{F}_W$ is defined to be at the origin at this tangent plane (where the tangent plane coincides with the reference ellipsoid), with the x, y and z axes pointing to true north, east and up (perpendicular to fixed the tangent plane) respectively. Note that this is not the same as an East, North, Up (ENU) coordinate frame as it is commonly referenced in the literature, since the origin of an ENU frame is the projection of the body frame onto the reference ellipsoid rather than the fixed tangent plane. Assuming that the x, y and z axes of $\mathcal{F}_{B_t}$ is aligned respectively front, left, up with the vehicle, the UGV is pointing due east and aligned with the tangent plane when $\mathbf{R}_{B_t}^{W} = \mathbf{I}_3$. The tangent provides a good local reference

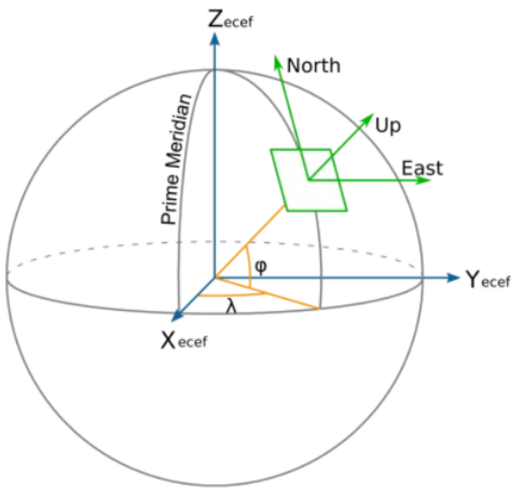


Figure 3.2

system, but for very large scale navigation, a new tangent plane would have to be defined when the distance away from the origin becomes too large.

The tangent plane can be initialized at any time and place on the reference ellipsoid, but in this case it is initialized at $t = 0$ with origin corresponding to φ_0, λ_0 and h_0, based on the assumption that the UGV has remained stationary for some time to get a good fix on the GNSS satellites before the navigation session begins.

The conversion between the geodetic lat., lon. coordinate system and the local tangent plane reference system is done via conversion to Earth-Centered, Earth-Fixed (ECEF) coordinates, and is described in detail in [67]. In practice, this conversion is made easy with the GeographicLib C++ library [37] and the `LocalCartesian` class which defines the local tangent coordinate system, and provides easy conversions between local and lat., lon. coordinates. The output of the visually augmented navigation system is given in lat., lon. coordinates so that the UGV can report this position back to a base station.

Defining the tangent reference frame at time $t = 0$ provides the translational component $\mathbf{t}^{W_0}$ of the pose $\mathbf{T}_{B_0}^{W}$, but the rotation $\mathbf{R}_{B_0}^{W}$ is still unknown. Assuming a scenario where this is not available from gyroscope or magnetic measurements, several GNSS measurements with sufficient relative motion in between are needed, and this procedure is explained in section 7.3.

Chapter 4

Camera Model and Calibration

This chapter explains the perspective camera model, stereo camera setup and camera calibration as well as how the camera calibration is performed in practice. Understanding the camera model is important since it describes how information from the world is interpreted by the cameras, which is central to understand when doing camera-based odometry.

4.1 Pinhole Camera and Perspective Projection

The Grasshopper3 cameras mounted on Olav approximately follows the perspective (pinhole) camera model, that is, objects that are projected onto the image appear smaller if they are far away from the camera than if they are close. The pinhole camera model describes the relationship between coordinates in the real world and their projected pixel coordinates on the image plane. In the model, the camera aperture is described as a point and no lenses are used to focus the light.

The image is formed by the intersection of the light rays from the objects through

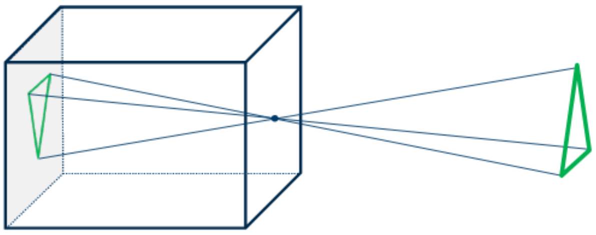


Figure 4.1: The pinhole camera model. Image courtesy of [29].

the center of projection (pinhole) with the image plane, depicted in figure 4.1. Figure 4.2 illustrates the same situation in a coordinate system, but here the image plane is moved in front of the center of projection in order to simplify the math, and this is the common convention. The coordinate frame $\mathcal{F}_C$, the camera frame, has the center of projection as its origin and its z-axis coinciding with the principal ray. The camera frame, which in our case is fixed to a robot which is moving, is often represented relative to a fixed world frame $\mathcal{F}_W$ which can be anywhere in the world.

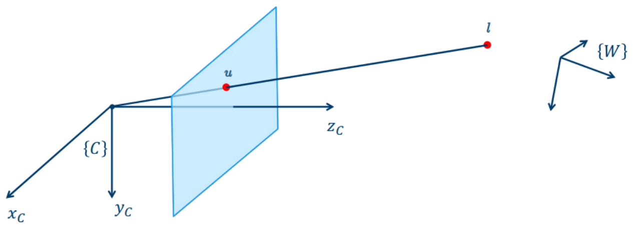


Figure 4.2: The normalized pinhole model. Image courtesy of [29].

We want to find the mapping from points in 3D space given relative to the world coordinate frame, to the corresponding pixel location of that same point on the image plane. Let $\tilde{\mathbf{l}}^W = (l_x^W, l_y^W, l_z^W, 1)^T$ be the coordinates of a landmark point given in the world coordinate frame, expressed in homogeneous coordinates. First, this point vector needs to be transformed into the camera reference frame $\mathbf{l}^C$, and this transformation is a function of the extrinsic camera parameters:

$$\mathbf{l}^C = \begin{bmatrix}\mathbf{R}_W^C & \mathbf{t}^C\end{bmatrix}\tilde{\mathbf{l}}^W \tag{4.1}$$

where $\mathbf{R}_W^C \in SO(3)$ and $\mathbf{t}^C \in \mathbb{R}^3$ is, respectively, the rotation matrix and translation vector of the world frame relative to the camera frame, which is known as the extrinsic parameters and will hereby be referred to as **R** and **t** to simplify notation. The matrix [**R t**] is known as the matrix of extrinsic parameters.

Now we need to find the point projection, $\hat{\mathbf{u}} = (\hat{u}, \hat{v})^T$, of $\mathbf{l}^C$ onto the image plane. The superscripts are temporarily omitted to simplify notation, but all the points are given in the camera frame. The image plane is assumed to be located at $z^C = f$, where f is called the focal length defined as the distance between the image plane and center of projection. From similar triangles, we see that:

$$\frac{\hat{v}}{l_y} = \frac{f}{l_z} \Rightarrow \hat{v} = \frac{f l_y}{l_z} \tag{4.2}$$

and by the same approach, we find:

$$\hat{u} = \frac{f l_x}{l_z}. \tag{4.3}$$

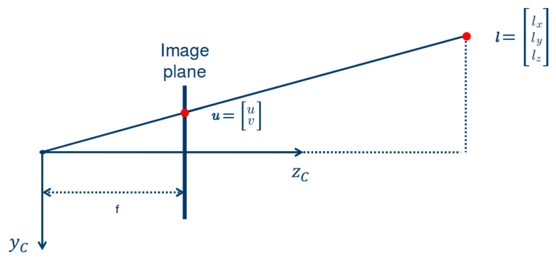


Image plane
$l = \begin{bmatrix} l_x \\ l_y \\ l_z \end{bmatrix}$
$u = \begin{bmatrix} u \\ v \end{bmatrix}$
z_C
f
y_C

Figure 4.3: Finding the point projection from similar triangles in the pinhole camera model. Image courtesy of [29].

Note that $\hat{\mathbf{u}}$ is given in metrical coordinates. Now, in order to find the pixel coordinates $\mathbf{u} = (u, v)^T$ on the image, we need to go through some additional steps. Because pixels are usually not perfectly square, the focal lengths in the x and y

direction, denoted f_u and f_v respectively, are different. f_u and f_v are given in pixels, and are products of the focal length f and the pixel densities given in pixels per millimeter, denoted s_u and s_v respectively. Additionally, since the principal point, i.e. the intersecting point of the principal ray on the image plane, is not necessarily the center of the image plane, the parameters c_u and c_v are introduced to model the possible displacement. The parameters (f_u, f_v, c_u, c_v) together form the intrinsic camera parameters. Assuming a high quality camera, we do not include pixel shear. The pixel coordinates can now be expressed as:

$$u = \frac{f_u x}{z} + c_u = s_u \hat{u} + c_u \tag{4.4}$$

and

$$v = \frac{f_v y}{z} + c_v = s_v \hat{v} + c_v \tag{4.5}$$

which can conveniently be expressed as a linear transformation:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} l_x \\ l_y \\ l_z \end{bmatrix} = \begin{bmatrix} f_u & 0 & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} l_x \\ l_y \\ l_z \end{bmatrix}. \tag{4.6}$$

where $\mathbf{K}$ is called the calibration matrix, or matrix of intrinsic parameters.

Putting it all together, we get:

$$\tilde{\mathbf{u}} = \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K}[\mathbf{R}\ \mathbf{t}]\tilde{\mathbf{l}}^W = \pi(\mathbf{l}^W; \mathbf{x}). \tag{4.7}$$

where $\pi(\mathbf{l}^W; \mathbf{x})$ is the projection function which depends on the vehicle state vector $\mathbf{x}$.

4.1.1 Distortion Model

The geometry of the perspective camera is simplified since we assume the pinhole to be infinitely small, but in reality the light needs to pass through a lens in order for the imager to gather enough light for rapid exposure. The need for a lens complicates the camera intrinsics, because it can cause radial distortion, which makes straight lines in the scene appears as curves in the image due to inaccurate or wide-angle lenses, and tangential distortion caused by misalignment between the lens and the image sensor. The two effects are depicted in figure 4.4. To account for these artifacts, we include a radial-tangential distortion model.

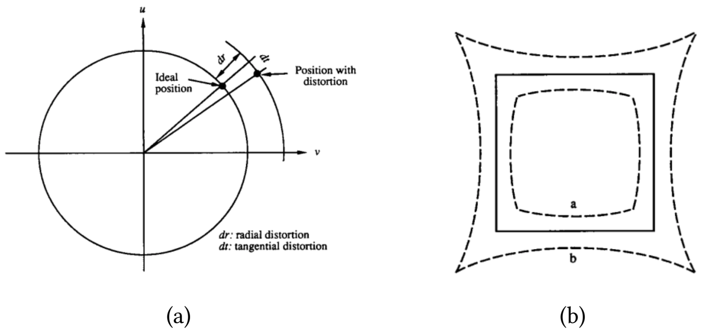


Figure 4.4: Fig. 4.4a depicts the effects of radial and tangential distortion. Fig. 4.4b depicts the effect of radial distortion. Solid lines: no distortion; dashed lines: with radial distortion (a: negative, b: positive). Image courtesy of [71].

Let $(\hat{u}_d, \hat{v}_d, 1)^T = \mathbf{K}^{-1}(u, v, 1)^T$ be the normalized image coordinates, denoted with the subscript d to make it clear that these image coordinates are affected by distortion. To correct for the radial distortion we use the Open Source Computer Vision Library (OpenCV) distortion model [3], which adds a correction term that follows from a Taylor expansion of the distortion factor dependent on the radial distance $r = \hat{u}_d^2 + \hat{v}_d^2$ from the principal point (assumed distortion center):

$$\Delta\hat{u}_{radial} = \hat{u}(k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{4.8}$$

$$\Delta\hat{v}_{radial} = \hat{v}(k_1 r^2 + k_2 r`4 + k_3 r^6) \tag{4.9}$$

For tangential distortion the correction term is given by

$$\Delta\hat{u}_{tangential} = 2p_1\hat{u}_d\hat{v}_d + p_2(r^2 + 2\hat{u}_d^2) \tag{4.10}$$

$$\Delta\hat{v}_{tangential} = p_1(r^2 + 2\hat{v}_d^2) + 2p_2\hat{u}_d\hat{v}_d. \tag{4.11}$$

The total expression for the corrected coordinates (x, y) are then

$$\hat{u} = \hat{u}_d + \Delta\hat{u}_{radial} + \Delta\hat{u}_{tangential} \tag{4.12}$$

$$\hat{v} = \hat{v}_d + \Delta\hat{v}_{radial} + \Delta\hat{v}_{tangential} \tag{4.13}$$

Finally, the undistorted image pixel coordinates are given by

$$u = f_u\hat{u} + c_u \tag{4.14}$$

$$v = f_v\hat{v} + c_v. \tag{4.15}$$

The intrinsic- and distortion parameters are independent of the scene viewed, and only has to be found once for each camera. The process of determining the camera parameters is known as camera calibration, and will be discussed in section ....

4.2 Stereo Vision

When projecting a point in 3D space onto the 2D image plane, the depth dimension is completely lost. This is intuitively similar to human vision, where it is hard to

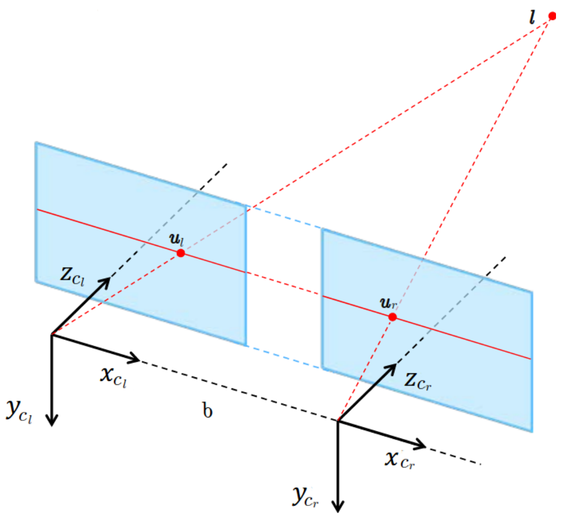


Figure 4.5: Rectified Stereo setup. Image courtesy of [29].

determine the distance to objects when one eye is kept closed. SLAM (or VO) can be performed using only a single camera, which is called monocular SLAM. This has the advantage of being the cheapest and smallest sensor setup for visual SLAM, but since the depth dimension is not observable, the scale of the map and estimated trajectory is unknown. To overcome this, monocular SLAM require multi-view or filtering techniques to produce an initial map, as it can not be triangulated from the first frame, and this adds quite a bit of complexity. Monocular SLAM also suffers from scale drift, and may fail when performing pure rotational motions. By using stereo cameras, i.e. two rigidly mounted and approximately aligned cameras looking at the same scene, the depth dimension can be recovered by triangulation in every frame, so problems related to scale drift and system bootstrapping are eliminated.

Two images looking at an overlapping scene is related through *epipolar geometry* so that all stereo correspondences, i.e. image points in the left and right image corresponding to the same 3D landmark, are constrained to lie on the corresponding *epipolar lines*. In a calibrated and rectified stereo camera setup, depicted in figure 4.5, the two image planes are perfectly aligned such that the epipolar lines are horizontal, and the focal lengths f and optical centers (c_u, c_v) are constrained to be the same for both cameras. For two image points $\mathbf{u}_l = (u_l, v_l)^T$ and $\mathbf{u}_r = (u_r, v_r)^T$ in respectively the left and right camera corresponding to the same 3D landmark $\mathbf{l}^W$, a detection of $\mathbf{u}_l$ in the left image would only require a search along the epipolar line, the u_r coordinate, in the right image to find the corresponding $\mathbf{u}_r$ in the right image. This severely constrains the search domain for stereo correspondences, which now only needs three coordinates to be described (u_l, v_l, u_r). Alternatively, we can use *disparity* to represent the third coordinate, given simply by $d = u_l - u_r$. The disparity value is inversely proportional to depth of the landmark, which can be found as

$$z = \frac{fb}{d} \tag{4.16}$$

where b is the baseline, i.e. the distance between the stereo cameras' principal points. Because of the nonlinear relationship in (4.16), small disparity differences make for large depth differences when $d \approx 0$, while large disparity differences make for small depth differences when d is large. The consequence is that stereo vision systems have high depth resolution only for object relatively near the camera.

For the stereo setup, we now have two camera projection functions:

$$\pi_i(\mathbf{l}^W; \mathbf{x}) = \tilde{\mathbf{u}}_i = \lambda_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_u \\ 0 & f & c_v \\ 0 & 0 & 1 \end{bmatrix} \left[\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \tilde{\mathbf{l}}^W - \begin{bmatrix} s_i \\ 0 \\ 0 \end{bmatrix} \right] \tag{4.17}$$

where $i \in \{l, r\}$, $s_l = 0$ and $s_r = b$. We can also define the re-projection function which re-projects stereo correspondences to the 3D landmark:

$$\pi^{-1}(\mathbf{u}_l, d; \mathbf{x}) = \tilde{\mathbf{l}}^W = \mathbf{T}^W_{C_l} \mathbf{Q} \begin{bmatrix} u_l \\ v_l \\ d \\ 1 \end{bmatrix} = \mathbf{T}^W_{C_l} \begin{bmatrix} 1 & 0 & 0 & -c_u \\ 0 & 1 & 0 & -c_v \\ 0 & 0 & 0 & f \\ 0 & 0 & 1/b & 0 \end{bmatrix} \begin{bmatrix} u_l \\ v_l \\ d \\ 1 \end{bmatrix} \tag{4.18}$$

4.3 Calibration and Rectification

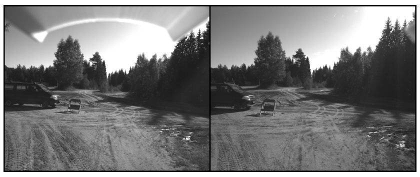
The goal of the calibration process is to find the intrinsic, extrinsic and distortion parameters of both cameras. This information can then be passed to a rectification step where the aim is to "correct" the individual images so that they appear as if they had been captured by two cameras with co-planar and row-aligned image planes.

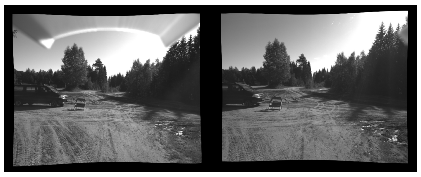
Finding the intrinsic and extrinsic parameters of each camera can be done by matching the pixel coordinates $\{\mathbf{u}_1, \mathbf{u}_2, \ldots\}$ to the set of corresponding scene points $\{\mathbf{l}_1^W, \mathbf{l}_2^W, \ldots\}$ with known coordinates. This way, the intrinsic and extrinsic parameters of each camera can be found by solving equation (4.17). In practice, this is done by detecting 3D points on a planar calibration object such as a chessboard, where the 3D points corresponds to the chessboard corners, and the corresponding pixels coordinates are usually sub-pixel refined. Because of the underlying distortion parameters, the problem is solved with optimization methods such as Levenberg-Marquardt (LM) or Gauss-Newton (GN) which optimizes over the intrinsic, extrinsic and distortion parameters simultaneously.

The camera parameters can then be passed to the stereo rectification step. This step is necessary, because it is practically impossible to perfectly align two stereo cameras such that they are perfectly co-planar and row-aligned. Therefore, the remaining correction is done virtually. Stereo rectification re-projects the image planes of the two cameras so that they reside in the exact same plane with exactly parallel optical axes, and with the same focal length f and optical center (c_u, c_v). Pixels from the original images are mapped to the rectified images through a rectification map. The result is perfectly horizontal epipolar lines, allowing a constrained search for stereo correspondences.

4.3.1 Kalibr and OpenCV

For the stereo calibration, I used the Kalibr library [21], [47], which is an open-source software developed by the Autonomous Systems Lab at ETH Zürich. Kalibr uses the ROS framework so that calibration sequences can simply be passed to the software as ROS-bags, making a very simple interface. In addition to chessboards, Kalibr provides the option of using an AprilGrid calibration object [55], which makes the calibration more robust to occlusions, i.e. if the calibration object is only partially visible to the camera. I found this software to be faster, easier to use and more robust than the OpenCV library, which provides a sample stereo calibration program. Using OpenCV, I had to carefully select individual stereo images to include in the calibration, because occluded calibration objects would cause the program to freeze and produce unacceptable results. Kalibr also provides a detailed report of the calibration result, including mean and variance for the reprojection error which is useful for specifying uncertainties in the VO algorithm.


(a) Original stereo images from the data set.


(b) Undistorted and rectified images.

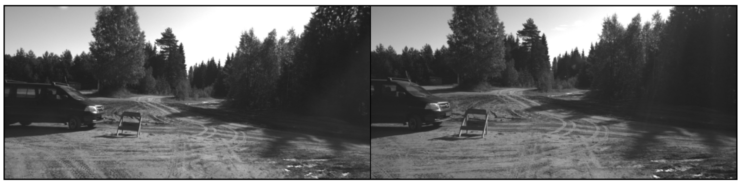

(c) Finally, the images are cropped to remove the sunscreen.

Figure 4.6: Rectification process.

The rectification process, however, had to be done in OpenCV, as Kalibr does not provide this functionality. The rectification map was produced with the OpenCV function `stereoRectify()`, based on the work by Loop and Zhang [43], which takes the intrinsic and extrinsic camera parameters of both cameras, and outputs new rotation and projection matrices for each camera. These matrices are then used as input to the function `initUndistortRectifyMap()` which initializes a pixel interpolation mapping for each camera, produces rectified images. The `stereoRectify()` function can optionally crop the images. This was necessary for the cameras on Olav, which has a sun-screen to protect against direct sunlight, but causes problems when performing VO. Figure 4.6 illustrates the effect of this mapping applied on a pair of stereo images from the provided data set.

Chapter 5

The SLAM Front-End

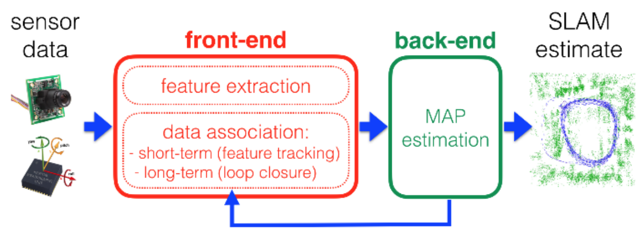


Figure 5.1: The front-end and back-end components of a modern SLAM system. Image courtesy of Cadena et al. [4].

As explained in chapter 1, SLAM is the problem of a robot incrementally building a consistent map of its environment while simultaneously determining its location within that map. The SLAM system must be able to process information in real-time as it moves through the environment, and this demand for parallelism is why most modern SLAM systems has some form of multi-threading, where the different parts of the system is often referred to as the *front-end* and the *back-end*.

As mentioned in section 1.2.2, most modern SLAM techniques are based on optimization/smoothing, and this is usually embedded in a Probibalistic Graphical Model (PGM). The front-end is responsible for initial construction of the graph, i.e., sensor abstraction, data association and initial pose estimates, while the back-end is responsible for performing probabilistic inference on the abstracted data produced by the front-end. The back-end can feed information back to the front-end, and provide information, for example, about potential loop closures. This anatomy is shown in figure 5.1.

The front-end and back-end creates a natural partitioning for discussing the different parts of the SLAM system. This chapter considers the front-end, and goes in-depth in two topics, namely VO and place recognition. For each topic, a module is chosen for the system implementation after a discussion.

5.1 Visual Odometry

VO is a part of the front end, and is responsible of finding the relative constraint between temporal frames. Generally, one can say that VO is SLAM with loop closing disabled, because all VO methods use information about the environment to find the relative pose constraints, and in that sense also does mapping. However, since the map is never re-used after finding the constraints (no loop closing), the map does not need to be maintained over long periods of time. VO methods can roughly be divided into two classes, namely indirect and direct methods.

5.1.1 Indirect Methods

Indirect-, or feature based VO methods pre-processes the raw images and extracts features/keypoints. These features can then be used to estimate the relative motion of the camera by finding point correspondences between frames. This can be done by matching features frame-to-frame, such as in LIBVISO2 [28] or SOFT [8], or by re-projection the features into a 3D feature map such that several poses can be optimized simultaneously using mutual observability constraints. This form of pose optimization is known as Bundle Adjustment (BA), and is used in seminal works such as ORB-SLAM2 [51]. Frame-to-frame methods does not track features over more than one frame at a time, and has the advantage that they are generally faster than BA methods, as there are less variables involved in the optimization, and there is no need to maintain a feature map. However, frame-to-frame methods has a disadvantage compared BA methods when it comes to accuracy.

Detecting and describing features is by itself a large field of research in computer vision. A huge variety of different detectors descriptors exist, and Pire et al. [56] provide a comparison of different detector and descriptor combinations in a VO application. A disadvantage of using features in general is that low textured and poorly illuminated environments creates problems when extracting and matching features, and these problems are partially solved using direct methods.

5.1.2 Direct Methods

Direct methods skips the image pre-processing step and uses the image pixel intensities directly in a probabilistic model. The direct approach then proceeds by minimizing a photometric error term, rather than geometric which is the case for indirect methods. Since direct methods eliminate the need for feature detectors and descriptors, they are expected to be more accurate and robust when there is little texture in the scene or blur on the image. They also provide a denser map reconstruction compared feature-based methods, which is only able to provide sparse reconstructions that is only useful for camera localization. A more dense map reconstruction could potentially be useful for other elements of the autonomous system.

Recent works by Engel et al. such as Large Scale Direct SLAM (LSD-SLAM) [15], [16] and DSO [14], [68] have proven very good results on publicly available data-sets compared to other state-of-the-art feature-based systems, as presented in the papers. Most surprising is the performance of DSO which is a VO method, i.e. no loop closure, that outperforms most available SLAM systems in regards to tracking accuracy. The down-side of using direct methods is that they are very sensitive to geometric noise that can originate from bad calibration or rolling shutter effects. Also, direct methods requires photometrically calibrated images to produce good results.

5.2 Selecting a Visual Odometry System

Initially, I investigated the potential of using existing algorithms which currently have an available open-source implementation. Here are the ones I tested:

- FOVIS, developed by Huang et al. [31], [32]. This is an algorithm originally intended for RGB-D cameras, which works by extracting Features from Accelerated Segment Test (FAST) features and then using the depth information only at those key points. Since the depth information is only used at keypoints, the depth information can be replaced by disparity information from the stereo cameras. After some experiments, I found that the algorithm could not process stereo frames faster than approximately 1Hz, which is too slow for the setup on Olav which publishes images at approximately 6 fps.
- ORB-SLAM2, by Mur-ArtalMur et al. [51]. Since ORB-SLAM2 is highly multi-threaded and does optimization over a co-visibility graph, essential graph, and local poses, integration of other sensor data directly into this system is a big challenge. I therefore tried to remove the loop closing functionality, and use the keyframes produced from local mapping in a pose graph optimized with iSAM2. This proved to be a challenge in practice, since ORB-SLAM2 uses g2o as optimization library. g2o and Georgia Tech Smoothing and Mapping library (GTSAM) did not work well together, as I experienced segmentation faults after introducing GTSAM into the code. This is possibly because of different versions of the Eigen library that the two libraries are built on. I then tried to rewrite the local bundle adjustment using GTSAM, but I did not get good results.
- Stereo-DSO by Wang et al. [68]. The TUM computer vision group has written an article about the DSO algorithm in a stereo version, but the open-source code is only available in a monocular version. Another repository from Horizon robotics [73] has used the paper to replicate the code, but I found this implementation to be highly unstable.
- LIBVISO2 by Geiger et al. [28], [38]. This library worked right out of the box, and has a very simple interface which made it easy to integrate into the

code. However, the algorithm does not achieve the best accuracy performance as compared for example to DSO or ORB-SLAM2, and was just barely able to achieve the real-time constraint on the KITTI dataset with 10.1 frames per second average. (The KITTI dataset has 10 FPS), leaving no overhead for loop closure detection etc. Also, LIBVISO2 uses its own matrix and vector data types, requiring data type conversions when working with other computer vision libraries such as OpenCv.

I ended up with implementing a visual odometry system inspired by LIBVISO2, the SOFT algorithm [8] which is currently the top contender on the KITTI dataset leaderboard, and the algorithm described in the paper by Manthe et al. [45]. The algorithm utilizes a lot of functionality from the OpenCV library, which has built in GPU support for many of its functions allowing for significantly higher computational performance and utilization of the on-board NVIDIA GPU installed on Olav. The algorithm is described in detail in the following section.

5.3 Visual Odometry Algorithm

An overview of stereo VO algorithm presented here is illustrated in figure 5.2. The algorithm is a feature based approach, which matches features in a circular fashion similarly to LIBVISO2 and the SOFT algorithm. Rather than using the custom corner and blob detectors used by those, features are matched between consecutive image pairs using a sparse Lucas Kanade optical flow algorithm [74], as is also done in [45]. The matched features are then used to estimate rotation using Nistérs 5-point algorithm [53], and translation by minimizing reprojection error between frames. The steps will be explained in more detail in the following subsections.

5.3.1 Feature Management

At each time step, a stereo image pair is processed. The first image pair initializes the algorithm, such that the first pose is set to the world origin. All images are assumed to be calibrated and rectified with coinciding focal lengths and principle points. For the

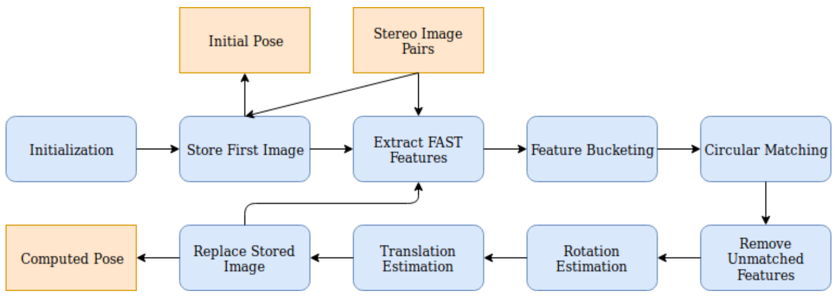


Figure 5.2: Pipelined overview of the VO algorithm. Boxes with rounded corners indicate algorithmic processes and boxes with sharp corners indicate input/output data. The arrows indicate the data flow.

first image pair, FAST features [58] [59] are extracted on the left image, serving as an initialization for the optical flow algorithm which requires good features to track in consecutive images. For the following frames, new FAST features are only extracted if the amount of tracked features falls below a given threshold, and new features are appended to the vector of tracked features.

To ensure that the extracted FAST features gets an approximately uniform distribution over the image domain, a bucketing procedure is used. The image is divided into buckets, or sectors, of size $b_x \times b_y$ where each bucket is filled with b_n number of features. As in SOFT, the bucket size is selected to be of size $b_x = b_y = 50$ pixels where $b_n = 4$ features per bucket seemed to yield a good trade off between computation time and pose estimation accuracy.

In addition to provide a good spread on the image features, the bucketing procedure also prioritizes which features to place in each bucket based on that features tracking history. Each tracked feature is assigned an age given by how many frames that feature has been detected. Features that are tracked for longer periods of time are considered to be more reliable, with lower probability of being an outlier. Therefore, if more than b_n falls into the same bucket, older features are prioritized over younger ones. However, during experimentation, features that was retained for a long period of time seemed to slowly drift, and therefore, features older than a threshold of 10 frames was replaced by new ones if available.

The first image pair along with its features are stored in memory. When the next image pair arrives, the initial features are tracked in a circle using a pyramidal implementation of the Lucas-Kanade Feature tracker algorithm developed by Bouget [74], implemented with GPU support in OpenCV by the function `calcOpticalFlowPyrLK()`. Starting in the first left image, each feature is consecutively tracked in the first right image, second right image, second left image, and lastly to the first left image again as depicted in figure 5.3. If the tracked feature coincides with the initial feature, the four feature correspondences are registered as a match and stored to use in the pose estimation step.

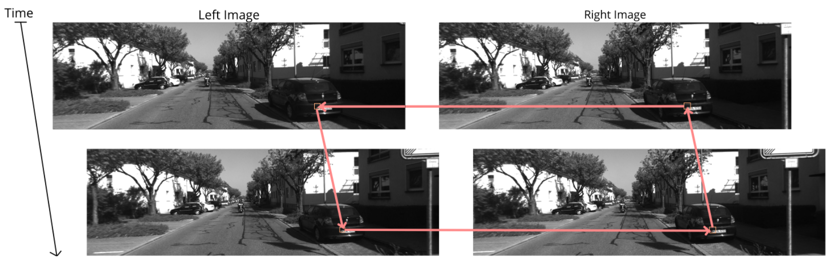


Figure 5.3: Circular matching of feature points: Starting from an initial feature in the left image at time $t-1$ (lower left), the feature is tracked consecutively in the right image at time $t-1$, right image at time t, left image at time t, and lastly at the left image at $t-1$ again. Only if the tracked feature location coincides with the starting location the match is accepted.

5.3.2 Pyramidal Lucas-Kanade Feature Tracker

The optical flow feature tracking algorithm tries to find feature correspondences between two images I and J. For a pixel $\mathbf{u} = [u_x, u_y]^T$ in the first image I, the algorithm tries to find the location $\mathbf{v} = \mathbf{u} + \mathbf{d} = [u_x + d_x, u_y + d_y]^T$ in the second image J such that $I(\mathbf{u})$ and $J(\mathbf{v})$ are similar. The vector $\mathbf{d}$ is known as an optical flow vector. The algorithm tries to minimize the residual function ϵ defined as:

$$\epsilon(\mathbf{d}) = \sum_{x=u_x-\omega_x}^{u_x+\omega_x} \sum_{y=u_y-\omega_y}^{u_y+\omega_y} (I(x,y) - J(x+d_x, y+d_y))^2 \tag{5.1}$$

where ω_x, ω_y defines an image neighborhood of size $(2\omega_x+1)\times(2\omega_y+1)$ on which the similarity is measured. A larger neighborhood is more robust as it allows to track features which moves with higher velocity between frames, but is also less accurate. In this implementation, $\omega_x = \omega_y = 10$ was used.

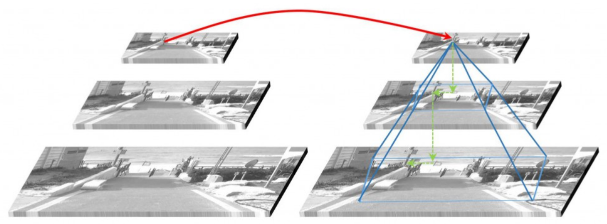


Figure 5.4: Optical flow pyramid: An initial estimate of the optical flow is computed for the image at lowest resolution, and continuously refined through the higher resolution levels down to the original image resulting in the final flow estimate **d**.

The pyramidal implementation builds, for each image, an image pyramid of $m+1$ levels where the original image serves as the bottom level, and higher levels are produced by recursively smoothing and down-sampling the input image. See figure 5.4. Features are initially tracked in the highest level of the pyramid, producing an initial flow vector which serves as an initial guess when searching for the feature in the higher resolution image one step lower in the pyramid. This process is repeated for each level of the pyramid until the bottom level/original image at the highest resolution is reached, yielding the resulting flow vector **d**. In this implementation, a pyramid with 3 levels is used.

5.3.3 Rotation Estimation

As in the SOFT algorithm, rotation is estimated using Nistérs five point algorithm [53]. For this, only the left images are used. For two images taken with the same calibrated camera, the epipolar constraint between two views can be expressed as:

$$\hat{\mathbf{u}}_t^T \mathbf{E} \hat{\mathbf{u}}_{t'} = 0, \tag{5.2}$$

where $\hat{\mathbf{u}}_t$ and $\hat{\mathbf{u}}_{t'}$ are normalized image projections of a world point at time t and t' respectively. The essential matrix is related to the relative pose as

$$\mathbf{E} = [\mathbf{t}]_\times \mathbf{R} \tag{5.3}$$

where $[\cdot]_\times$ denotes the skew symmetric operator. The essential matrix is in $\mathbb{R}^{3\times3}$, but since scale is unobservable, eight unknowns are left to be solved. However, Nistér uses the fact that

$$det(\mathbf{E}) = 0 \tag{5.4}$$

and

$$2\mathbf{E}\mathbf{E}^T\mathbf{E} - tr(\mathbf{E}\mathbf{E}^T)\mathbf{E} = 0 \tag{5.5}$$

to impose additional constraints such that only 5 correspondences are needed to estimate $\mathbf{E}$. The five point algorithm is used in conjunction with Random Sample Consensus (RANSAC) [18]. A number of random five point subsets are taken from the total set of points, and the Essential matrix is calculated for each subset. Finally, the essential matrix that has the larges set of inliers among all the points is selected as the final solution.

The relative pose from the essential matrix can be recovered up to the scale of $\mathbf{t}$, as shown by Longuet-Higgins [42]. Since we can only estimate $\mathbf{E}$ up to scale, we can always re-scale it so that the Singular Value Decomposition (SVD) has the form

$$\mathbf{E} = \mathbf{U}\mathbf{D}\mathbf{V}^T = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \mathbf{v}_3^T \end{bmatrix} \tag{5.6}$$

where $det(\mathbf{U}) = det(\mathbf{V}) = 1$. Then one can show that

$$\mathbf{R} \in \{\mathbf{U}\mathbf{W}\mathbf{V}^T, \mathbf{U}\mathbf{W}^T\mathbf{V}^T\} \tag{5.7}$$

$$\mathbf{t} = \pm\lambda\mathbf{u}_3,\ \lambda \in \mathbb{R} \setminus \{0\} \tag{5.8}$$

where

$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.9}$$

The different possible combinations of **R** and **t** thus leaves 4 possible solutions. In order to determine which combination is correct, some keypoints are triangulated to ensure that the Cheirality constraint is satisfied, i.e. all scene points are in front of both cameras. This is all done with the OpenCV function `recoverPose()` which takes in the essential matrix computed from the five point algorithm with the OpenCV function `findEssentialMat()`.

5.3.4 Translation Estimation

To recover the translation, each stereo correspondence from the previous frame are re-projected into 3D using the inverse projection function π^{-1} as defined in equation (4.18). The 3D point is then projected into the current left frame using the projection function π_l and π_r as defined in (4.17). Using LM optimization, the error term

$$\sum_{i=1}^{N} ||\mathbf{u}_{l,i} - \pi_l(\mathbf{l}_i^W; \mathbf{R}, \mathbf{t})||^2 + ||\mathbf{u}_{r,i} - \pi_r(\mathbf{l}_i^W; \mathbf{R}, \mathbf{t})||^2 \tag{5.10}$$

is iteratively minimized with respect to the relative translation **t**, as the rotation is already found from the 5-point algorithm. The translation is initialized to the result of the previous configuration to speed up convergence. To be robust against outliers, the estimation is embedded in a RANSAC scheme.

Estimating *both* rotation and translation by minimizing the reprojection error was also experimented with, yielding slightly better computation times, but worse odometry accuracy.

5.4 Place Recognition

A key feature that separates any SLAM system from a VO system is the ability to recognize previously mapped environments. This allows the system to re-localize after tracking failure, which might happen due to occlusion, abrupt movements or driving towards the sun. It also enables detecting loop closures in the map, which allows to correct for errors that have accumulated during exploration. Loop closures also informs the robot about the true topology of the environment, allowing it to find shortcuts between locations, as illustrated in figure 5.5.

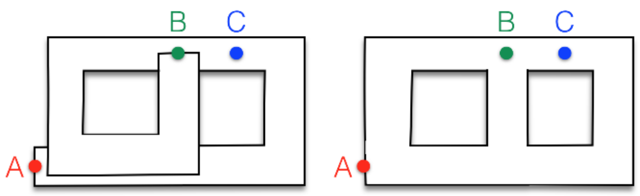


Figure 5.5: A robot performing VO only will interpret the world as an infinite corridor. Loop detection informs the robot about the true topology of the environment. Image courtesy of [4].

One potential downside of a SLAM system with loop detection compared to a VO system is that wrong loop closures can corrupt the map, making it useless to navigate in. Therefore, the properties of the place recognition module is very important in the SLAM system, as well as having a policy for rejecting wrong associations. As mentioned earlier, the loop detection module is usually considered part of the front-end, with the back-end informing the front-end about potential loop-closures and also providing geometrical verification which allows for discarding unlikely ones.

As the system has to perform in real-time, a place recognition module has to be very effective. A naive brute force approach which detects features in the image and tries to match them against all previously detected features quickly becomes impractical, especially in large-scale environments. State-of-the-art real-time capable place recognition methods such as Fast Appearance-Based Mapping 2.0 (FAB-MAP 2.0) by Cummins and Newman, [7], [6], and DBOW2 by Galvez-Lopez and Tardós [24], are appearance-based methods which uses Bag of Words (BoW) techniques to build a database of previously visited locations from *visual words*. Both methods store their databases as tree structures, allowing for fast matching and retrieval. The tree structures are called *visual vocabularies*, which requires an offline training stage to be constructed. Visual- words and vocabulary will be explained in more detail in section 5.4.2.

In the case of severe illumination variations which is common in long-term operation, the BoW descriptors are harder to match and in these conditions appearance-based place recognition often fail. One new method that explicitly account for such variations by matching sequences instead of image templates is seqSLAM by Milford and Gordon [48], however the current implementation is quite expensive on memory consumption and is therefore, at this point, not suitable for this project.

5.4.1 Selecting a Place Recognition System

For this project I chose to use the DBOW2 algorithm. This choice is partially based on the survey by Williams et al. [72], which compares several approaches for place recognition and concluded that techniques based on appearance, image to image matching, seemed to scale better in large environments than map-to-map or image-to-map methods. DBOW2 achieves similar recall performance as FAB-MAP 2.0, but with much lower computation times. DBOW2 can reportedly search and retrieve matches from a database of 10 000 images using only 39 ms on average [52]. The reason for the computational advantage over FAB-MAP 2.0 is that DBOW2 uses binary descriptors, such as Oriented FAST and Rotated BRIEF (ORB) [60] or Binary Robust Elementary Features (BRIEF) [5], to construct the vocabulary and database, while FAB-MAP 2.0 uses Speeded Up Robust Features (SURF) features which takes 400 ms to extract [52]. The algorithm has an open-source implementation [22] which is templated, so it can work with any type of descriptor at the users convenience. The implementation is built on OpenCV which allows easy integration into the developed VO system from section 5.3. DBOW2 serves as the place recognition module in one of the state-of-the-art SLAM algorithms, ORB-SLAM2, which was reviewed in my project work last semester. I found it to be very reliable and did not encounter a single false positive during testing. Similar to ORB-SLAM2, I use the DBOW2 library with ORB features, as this descriptor is more invariant to changes in scale and rotation compared to BRIEF [52]. Additionally, a pre-trained visual vocabulary based on ORB features is readily available from the ORB-SLAM2 Git-Hub repository [50], which is convenient because training a good vocabulary requires thousands of images captured in a wide range of conditions.

5.4.2 Place Recognition with Dynamic Bag of Words 2

The visual vocabulary in DBOW2 is created offline in an offline training step. ORB features are extracted from thousands of images captured in a wide range of conditions, and the set of all these features is called the descriptor space. The descriptor space is first discretized into k_w binary clusters by performing k-means clustering with k-means++ seeding [1]. These clusters form the first level of nodes in the tree. Subsequent levels are created by repeating this operation with the descriptors associated with each node, up to L_w times. This results in a tree with W leaves called the vocabulary words. The ORB vocabulary used consists of $L_w = 6$, with 10 nodes at each level, resulting in one million words. Each word is weighted according to the Term Frequency-Inverse Document Frequency (tf-idf), i.e. each word is given a weight according to its relevance

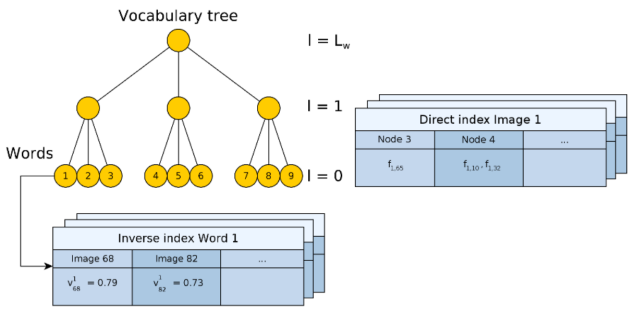


Figure 5.6: Example of the vocabulary tree and direct and inverse indexes that compose the image database. The vocabulary words are the leaf nodes of the tree. The inverse index stores the weight of the words in the images in which they appear. The direct index stores the features of the images and their associated nodes at a certain level of the vocabulary tree.

in the training set, decreasing the weight of words which are very frequent and thus less discriminative.

In order to detect if an image corresponds to a revisited place, the content of an image is summarized by converting it to a BoW vector. First, ORB features are detected and extracted from the image I_t taken at time t. To convert the extracted feature vectors to a BoW vector $\mathbf{v}_t \in \mathbb{R}^W$, the binary feature descriptors traverse a vocabulary tree from the root to the leaves, selecting at each level the intermediate nodes that minimize the Hamming distance.

To store the history of visited location, DBOW2 creates a database which stores three data structures, with the first being the visual vocabulary which was already discussed. The second is an inverse index, which for every word w_i in the vocabulary stores a list of images I_t in which it appears, together with the corresponding weight of the word in the image v_t^i (see figure 5.6). This is useful when querying the database, since it allows to perform comparisons only against those images that have some word in common with the query image.

The third structure is a direct index, which stores for each image I_t the nodes which are ancestors to the words present in it, as well as the list of local features f_{tj}

associated to each node. This allows to speed up geometrical verification between images by computing correspondences only between those features that belong to the same words.

To detect loop closures, a four step approach is used. First, the database is queried with the vector $\mathbf{v}_t$ which results in a list of potential matches $\{(\mathbf{v}_t, \mathbf{v}_{t_1}), (\mathbf{v}_t, \mathbf{v}_{t_2}), \ldots\}$ associated with their scores $s(\mathbf{v}_t, \mathbf{v}_{t_j}) \in [0, 1]$ which is computed by

$$s(\mathbf{v}_t, \mathbf{v}_{t_j}) = 1 - \frac{1}{2} \left| \frac{\mathbf{v}_t}{|\mathbf{v}_t} - \frac{\mathbf{v}_{t_j}}{|\mathbf{v}_{t_j}|} \right| . \tag{5.11}$$

This score is normalized with the score one would expect to get for an image showing the same place. This is approximated with the previous image processed:

$$\eta(\mathbf{v}, \mathbf{v}_{t_j}) = \frac{s(\mathbf{v}_t, \mathbf{v}_{t_j})}{s(\mathbf{v}_t, \mathbf{v}_{t-1})} \tag{5.12}$$

Matches whose $\eta(\mathbf{v}, \mathbf{v}_{t_j})$ score does not achieve a minimum threshold are then discarded.

The second step is to group matches that are close in time. If I_t and $I_{t'}$ represent a real loop closure, I_t is very likely to be similar to $I_{t' \pm \Delta t}, I_{t' \pm 2\Delta t}, \ldots$. Denoting the interval composed of timestamps $\{t_{n_i}, \ldots, t_{m_i}\}$ by T_i, several matches $\{(\mathbf{v}_t, \mathbf{v}_{t_{n_i}}), \ldots, (\mathbf{v}_t, \mathbf{v}_{t_{m_i}})\}$ are converted into a single match $(\mathbf{v}_t, V_{T_i})$ if the gaps between consecutive timestamps are small. This group match is then ranked according to a score H:

$$H(\mathbf{v}_t, V_{T_i}) = \sum_{j=n_i}^{m_i} \eta(\mathbf{v}_t, \mathbf{v}_{t_j}). \tag{5.13}$$

The group match with the highest score is selected as matching group and continues to the third step, which is a temporal consistency check.

After obtaining the matching island $V_{T'}$, it is checked for temporal consistency with previous queries. The match $(\mathbf{v}_t, V_{T'})$ must be consistent with k previous matches $\{(\mathbf{v}_{t-1}, V_{T_1}), \ldots, (\mathbf{v}_{t-k}, V_{T_k})\}$. If so, the vector $\mathbf{v}_{t'} \in V_{T'}$, that achieves the highest η score with $\mathbf{v}_t$ is kept and processed through the fourth and final stage which checks for geometrical consistency. The geometrical consistency checks tries to find with RANSAC the fundamental matrix between images I_t and the matching candidate $I_{t'}$ supported by at least 12 correspondences. The fundamental matrix $\mathbf{F} \in \mathbb{R}^{3\times3}$ relates two point matches $\mathbf{u}_t$ and $\mathbf{u}_{t'}$ in image I_t and $I_{t'}$ respectively, by the equation

$$\mathbf{u}_t^T \mathbf{F} \mathbf{u}_{t'} = 0. \tag{5.14}$$

To find **F**, the local features of the query image and those of the matched one must be compared. Using the direct index in the database, the features of each image is quickly extracted, and the matching is speeded up because only features associated with the same nodes in the vocabulary tree needs to be compared. If the computation of the fundamental matrix was successful, the match is finally accepted.

5.4.3 Finding the Loop Closure Relative Pose Constraint

In order to find the relative pose

$$\mathbf{T}_{C_{t'}}^{C_t} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix} \tag{5.15}$$

between the two matching frames, the matching stereo pair is loaded from the database of previously recorded frames. Then the relative pose is recovered by applying the algorithm detailed in section 5.3.

Chapter 6

The SLAM Back-End

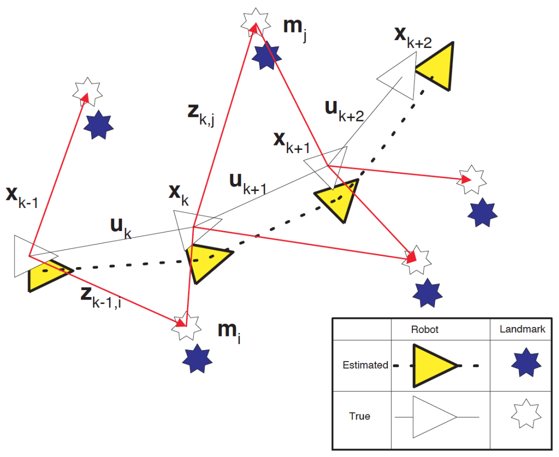


Figure 6.1: The essential SLAM problem. The pose estimate affects the estimates of landmark positions and vice versa, implying high correlation between the variables. Image courtesy of Durrant-Whyte and Bailey [13].

As mentioned in the beginning of chapter 5, the back-end of a SLAM system is responsible for performing probabilistic inference on the abstracted data produced by the front-end. Probabilistic inference is a term associated with the use of PGMs, and in SLAM there is one specific inference problem that we wish to solve, namely the joint Maximum a Posteriori (MAP) probabilities of the variables in the graph. This will be explained in the next section, followed by a review of factor graphs and how to incorporate different sensors into this framework. Lastly, in section 6.6, a review of different optimization frameworks is provided followed by a description of an optimization framework chosen for the system implementation.

6.1 Probabilistic Formulation of SLAM

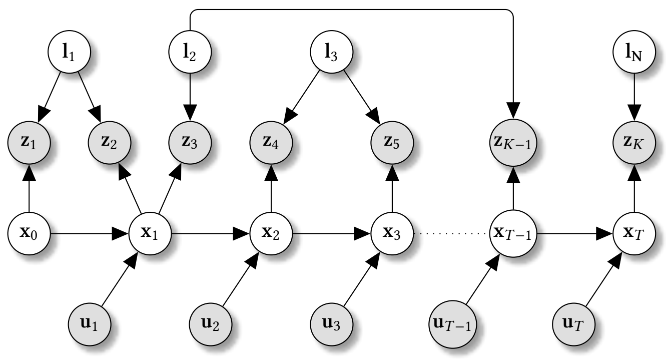


Figure 6.2: A Dynamic Bayesian Network representing a SLAM scenario. A loop closure is detected between the states $\mathbf{x}_1$ and $\mathbf{x}_{T-1}$ through their mutual observation of landmark $\mathbf{l}_2$. White nodes represent latent variables, and grey nodes represent observed variables.

Consider a mobile robot moving through an environment taking relative observations of a number of unknown landmarks using sensors located on the robot as shown in Figure 6.1. Due to the inherent noise in the measurements, the SLAM problem is described by means of probabilistic tools. At time step t, the state of the robot is parametrized by the state vector $\mathbf{x}_t$. We assume that the robot is moving through the environment according to the equation

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t) + \mathbf{w}_t \tag{6.1}$$

where $\mathbf{u}_t$ is the measured odometry vector (sometimes the control input) between the states $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$, and the noise $\mathbf{w}_t$ is assumed to be distributed according to a zero-mean Gaussian with covariance matrix $\mathbf{\Lambda}_t$, i.e., $\mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{\Lambda}_t)$. While the robot is moving, it makes landmark measurements described by the equation

$$\mathbf{z}_k = h_k(\mathbf{x}_{i_k}, \mathbf{l}_{j_k}) + \mathbf{v}_k \tag{6.2}$$

where $\mathbf{z}_k$ is the k^{th} landmark measurement vector, $\mathbf{l}_j$ is a vector describing the location of the j^{th} landmark whose true location is assumed time invariant, and the measurement noise $\mathbf{v}_k$ is also assumed to be Gaussian distributed, i.e. $\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \Sigma_k)$. The indices i_k and j_k denotes respectively the vehicle state and landmark related to the k^{th} measurement.

Figure 6.2 shows a Dynamic Bayesian Network (DBN) representation of a given SLAM scenario. In the model, there are some underlying assumptions that are usually made for this problem to be tractable; the world is static, the noise is independent, and the Markov property holds. The DBN is a Directed Acyclic Graph (DAG) that encodes the conditional independence structure of a set of variables, where each variable only directly depends on its predecessors in the graph. Since SLAM is usually performed on-line, the joint probability for all the states up to a time step t is given by

$$p(X_t, M_t, Z_t, U_t) \propto p(\mathbf{x}_0) \prod_{i=1}^{t} p(\mathbf{x}_i \mid \mathbf{x}_{i-1}, \mathbf{u}_i) \prod_{k=1}^{K_t} p(\mathbf{z}_k \mid \mathbf{x}_{i_k}, \mathbf{l}_{j_k}) \tag{6.3}$$

where X_t, U_t, M_t and Z_t are sets given by

- $X_t = \{\mathbf{x}_i\}_{i=0}^{t}$: The history of vehicle locations up to time step t.
- $U_t = \{\mathbf{u}_i;\}_{i=1}^{t}$: The history of control inputs up to time step t.
- $M_t = \{\mathbf{l}_j\}_{j=1}^{N_t}$: The set of landmarks observed up until time step t.
- $Z_t = \{\mathbf{z}_k\}_{k=1}^{K_t}$: The set of all landmark measurements up to time t.

In (6.3), $p(\mathbf{x}_0)$ is a prior on the initial state, $p(\mathbf{x}_i \mid \mathbf{x}_{i-1}, \mathbf{u}_i) \sim \mathcal{N}(f_i(\mathbf{x}_{i-1}, \mathbf{u}_i), \Lambda_i)$ is the motion model parametrized by a control input or odometry measurement, which is assumed known, and $p(\mathbf{z}_k \mid \mathbf{x}_{i_k}, \mathbf{l}_{j_k}) \sim \mathcal{N}(h_k(\mathbf{x}_{i_k}, \mathbf{l}_{j_k}), \Sigma_k)$ is the landmark measurement model. We assume a uniform prior on all landmarks, and states after $t = 0$. Note that this model assumes that the data association problem, i.e., finding the correct indices i_k and j_k relating to $\mathbf{z}_k$, has been solved.

We are interested in the most likely configuration of the vehicle states and landmarks, therefore we want to find the MAP estimate which we get by maximizing the joint probability of states and landmarks given all available measurements and priors. The joint posterior we want to maximize is given by

$$\begin{aligned} p(X_t, M_t \mid Z_t, U_t) &= \underset{X_t, M_t}{\arg\max} \frac{p(Z_t, U_t \mid X_t, M_t) p(X_t, M_t)}{p(Z_t, U_t)} \\ &\propto p(Z_t, U_t \mid X_t, M_t) p(X_t, M_t) = p(X_t, M_t, Z_t, U_t) \end{aligned} \tag{6.4}$$

where the first equality follows from Bayes law. The MAP estimate can now be found by

$$\begin{aligned} X_t^{*,MAP}, M_t^{*,MAP} &= \underset{X_t, M_t}{\arg\max}\, p(X_t, M_t \mid Z_t, U_t) \\ &= \underset{X_t, M_t}{\arg\max}\, p(X_t, M_t, Z_t, U_t) \\ &= \underset{X_t, M_t}{\arg\min} - \log(p(X_t, M_t, Z_t, U_t)) \\ &= \underset{X_t, M_t}{\arg\min} - \log(p(\mathbf{x}_0)) - \sum_{i=1}^{t} \log(p(\mathbf{x}_i \mid \mathbf{x}_{i-1} \mathbf{u}_i)) \\ &\qquad - \sum_{k=1}^{K_t} \log(p(\mathbf{z}_k \mid \mathbf{x}_{i_k}, \mathbf{l}_{j_k})) \end{aligned} \tag{6.5}$$

The prior $p(\mathbf{x}_0)$ often assumed known (initialized to the origin), and in that case, the MAP estimate becomes the Maximum Likelihood (ML) estimate.

$$\begin{aligned} X_t^{*,ML}, M_t^{*,ML} &= \arg\min_{X_t, M_t} - \sum_{i=1}^{t} \log(p(\mathbf{x}_i \mid \mathbf{x}_{i-1}\mathbf{u}_i)) - \sum_{k=1}^{K_t} \log(p(\mathbf{z}_k \mid \mathbf{x}_{i_k}, \mathbf{l}_{j_k})) \\ &= \arg\min_{X_t, M_t} \sum_{i=1}^{t} \|f(\mathbf{x}_{i-1}, \mathbf{u}_i) - \mathbf{x}_i\|^2_{\Lambda_i} + \sum_{k=1}^{K_t} \|h_k(\mathbf{x}_{i_k}, \mathbf{l}_{j_k}) - \mathbf{z}_k\|^2_{\Sigma_k} \end{aligned} \tag{6.6}$$

where

$$\|\mathbf{e}\|^2_\Sigma \triangleq \mathbf{e}^T \Sigma^{-1} \mathbf{e} \tag{6.7}$$

is the squared Mahalanobis distance. It is possible to substitute (6.7) with a robust loss function (e.g., Huber or Turkey loss), which implies a distribution with heavier tails than the Gaussian which can increase resilience to outliers.

6.2 Factor Graph Representations for SLAM

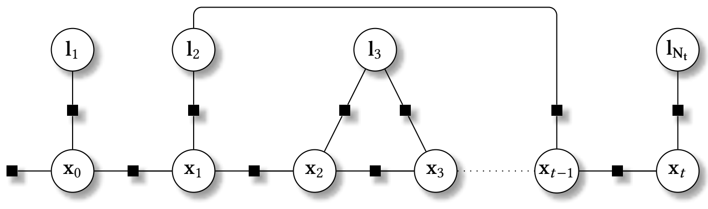


Figure 6.3: A Factor graph representing the same SLAM scenario as in figure 6.2. A loop closure is detected between the states $\mathbf{x}_1$ and $\mathbf{x}_{T-1}$ through their mutual observation of landmark $\mathbf{l}_2$. Black squares represent factor nodes.

The SLAM problem can be expressed using different kinds of PGMs which makes reasoning about the interdependence among the involved variables more convenient, as shown in the last section where equation (6.3) relates directly to the DBN in figure 6.2. The DBN has long been the preferred PGM for SLAM, but in recent years this preference has shifted to the factor graph [4], which Dellaert and Kaess reviews in [12].

In the paper, they also explain how the factor graph relates to other popular models like the DBN and the Markov Random Field (MRF).

A factor graph is a bipartite graph $\mathcal{G} = (\mathcal{F}, \Theta, \mathcal{E})$ with two types of nodes: factors $\phi_i \in \mathcal{F}$ and variables $\theta_j \in \Theta$. Edges $e_{ij} \in \mathcal{E}$ are always between factor nodes and variables nodes. A factor graph $\mathcal{G}$ defines the factorization of a function $\phi(\Theta)$ as

$$\phi(\Theta) = \prod_{i=0}^{|\mathcal{F}|} \phi_i(\Theta_i) \tag{6.8}$$

where $|\mathcal{F}|$ is the cardinality of the set $\mathcal{F}$, Θ_i is the set of variables θ_j adjacent to the factor ϕ_i, and independence relationships are encoded by the edges e_{ij}. Each factor ϕ_i is a function of the variables in Θ_i. The joint probability of all variables is proportional to ϕ, i.e., $p(\Theta) \propto \phi(\Theta)$, so factors act as unnormalized probabilities. Our goal is to find the variable assignment $\Theta^{*,MAP}$, corresponding to the MAP estimate, that maximizes

$$\Theta^{*,MAP} = \arg\max_{\Theta} \phi(\Theta) \tag{6.9}$$

When assuming Gaussian measurement models, we assume that each factor is on the form

$$\phi_i(\Theta_i) \propto exp(-\frac{1}{2}||h_i(\Theta_i) - \mathbf{z}_i||^2_{\Sigma_i}) \tag{6.10}$$

which corresponds to an error between the measurement function h_i evaluated at the estimated Θ_i, and the measurement corrupted by a zero mean Gaussian noise $\mathbf{z}_i = h_i(\Theta_i) + \mathbf{w}_i$ where $\mathbf{w}_i \sim \mathcal{N}(\mathbf{0}, \Sigma_i)$. In this case, the factored objective function to maximize (6.8) corresponds to the nonlinear least-squares criterion

$$\Theta^{*,MAP} = \arg\min_{\Theta}(-\log \phi(\Theta)) = \arg\min_{\Theta} \sum_{i=0}^{|\mathcal{F}|} ||h_i(\Theta_i) - \mathbf{z}_i||^2_{\Sigma_i} \tag{6.11}$$

where $h_i(\Theta)$ is a measurement function and $\mathbf{z}_i$ is a measurement. Factors are sometimes referred to as potentials, and minimizing (6.11) can be seen as an energy minimization problem over the factor graph.

Going back to the SLAM scenario in figure 6.3 and 6.2, the equivalence relations between equation (6.3) and (6.8) can be established by setting $\Theta = \{X_t, M_t\}$ and taking

$$\begin{aligned} \phi_{(\cdot)}(\mathbf{x}_0) &\propto p(\mathbf{x}_0) \\ \phi_{(\cdot)}(\mathbf{x}_{i-1}, \mathbf{x}_i) &\propto p(\mathbf{x}_i \mid \mathbf{x}_{i-1}, \mathbf{u}_i) \\ \phi_{(\cdot)}(\mathbf{x}_{i_k}, \mathbf{l}_{j_k}) &\propto p(\mathbf{z}_k \mid \mathbf{x}_{i_k}, \mathbf{l}_{j_k}) \end{aligned} \tag{6.12}$$

where the indices of the factors are left unspecified, since they can be ordered arbitrarily.

6.2.1 Pose Graphs

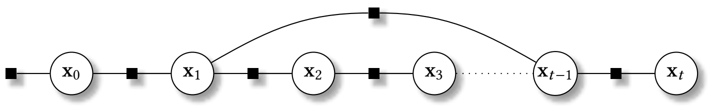


Figure 6.4: Pose factor graph representing the same scenario as in figure 6.4

For very large scale SLAM problems, optimizing over all observed landmarks can become computationally intractable over time. If we are not explicitly interested in building a map, but rather in knowing the history of vehicle poses to a high degree of certainty, we can omit the the landmarks as unknowns in the factor graph which we explicitly optimize over. This results in a pose factor graph, depicted in figure 6.4, where the loop closure constraint is represented by a factor between poses $\mathbf{x}_1$ and $\mathbf{x}_{t-1}$. In the model, the unknown variable set Θ_t from the last section becomes the history of vehicle poses only, i.e., $\Theta_t = X_t$, while explicit landmarks or other information about the environment are not part of the pose graph itself. However, such information can be attached to the nodes in the graph, hence allowing the pose graph to express landmark/feature maps at a later stage. In the case of landmark observations with a stereo camera, attaching the coordinates of all observed keypoints observed from a particular pose allows us to project the landmarks into the world frame and build a feature map. Mutual observations of landmarks between frames results in odometry information which are represented as factor nodes between variable nodes in the graph.

In a pose factor graph, two types of factor constraints between pose nodes are identified; odometry constraints between consecutive poses, and loop-closure constraints due to revisiting a previously explored region of the environment. For the odometry constraints, the model is the same as in section 6.1, namely

$$\mathbf{x}_t \sim \mathcal{N}(f_t(\mathbf{x}_{t-1}, \mathbf{u}_t), \mathbf{\Lambda}_t). \tag{6.13}$$

A detected loop is between two non-successive poses $\mathbf{x}_i$, $\mathbf{x}_j$ enforces the constraint

$$\mathbf{x}_j \sim \mathcal{N}(f_{ij}(\mathbf{x}_i, \mathbf{u}_{ij}), \mathbf{\Lambda}_{ij}), \tag{6.14}$$

where $\mathbf{u}_{ij}$ is a pseudo-odometry measurement between $\mathbf{x}_i$ and $\mathbf{x}_j$. In order for the robot to recognize previously visited places, it has to perform place recognition which is described in section 5.4. Section 5.4 also describes how we can find the pseudo odometry constraint $\mathbf{u}_{ij}$ between the images.

As mentioned, in pose graph SLAM we are only interested in the most likely configuration of the poses, given the set of odometry and loop-closure measurements up to time t, U_t. If there are no priors given, this can be found by maximizing the likelihood

$$\begin{aligned}\Theta^{*,ML} &= X_t^{*,ML} = \arg\max_{X_t} p(X_t \mid U_t) = \arg\min_{X_t} -\log p(X_t \mid U_t) \\ &= \arg\min_{X_t} \sum_{i=1}^{t} \|f_i(\mathbf{x}_{i-1}, \mathbf{u}_i) - \mathbf{x}_i\|^2_{\mathbf{\Lambda}_i} + \sum_{i,j} \|f_{ij}(\mathbf{x}_i, \mathbf{u}_{ij}) - \mathbf{x}_j\|^2_{\mathbf{\Lambda}_{ij}}\end{aligned} \tag{6.15}$$

6.3 Factor Graph Formulation in Inertial Navigation Systems

While a factor represents the general concept of an error function that should be minimized, it is common in the navigational literature to design a measurement model $h(\cdot)$ that predicts a sensor measurement given a state estimate. The factor then captures the error between the predicted measurement and the actual measurement.

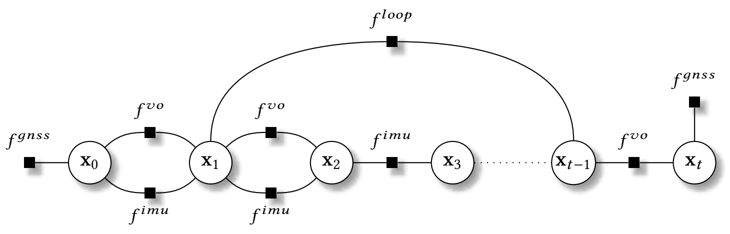


Figure 6.5: A factor graph with VO, GNSS, IMU and place recognition. Two poses can be constrained an IMU factor, a VO factor or both. In areas with poor GNSS coverage, only some poses will be constrained by a GNSS unary factor.

6.3.1 Stereo Vision Measurements

Assuming a known baseline, one can incorporate the observed landmark $\mathbf{l}$ as a variable into the optimization and add a factor that represents the projection of the observed landmark onto the image plane of the two cameras. Such a factor is defined as

$$f^{stereo}(\mathbf{x}_t, \mathbf{l}) = \|\mathbf{z}_{t,l} - \pi_l(\mathbf{l}; \mathbf{x}_t)\|^2_{\Sigma_l} + \|\mathbf{z}_{t_r} - \pi_r(\mathbf{l}; \mathbf{x}_t)\|^2_{\Sigma_r} \tag{6.16}$$

where π_l and π_r is defined in (4.17).

Alternatively, it is possible to first estimate the relative transformation $\mathbf{T}^{C_{l,i}}_{C_{l,j}}$ between two stereo frames at time i and j using all of the landmark observations in those frames. Such a transformation can be used to predict the robot pose at t_j based on $\mathbf{x}_i$. with the corresponding visual odometry binary factor $f^{VO}(\mathbf{x}_i, \mathbf{x}_j)$.

Assuming a known baseline, it is possible to estimate the relative transformation between two stereo frames. This can be formulated as a binary factor connected to navigation nodes at the time instances of these frames. Denoting this relative transformation by T_Δ and the global poses of the two stereo cameras by T_{k_1} and T_{k_2}, calculated based on the current values of the navigation nodes x_{k_1} and x_{k_2}, the binary factor becomes

$$f^{stereo}(x_{k_1}, x_{k_2}) \triangleq \|T_\Delta - (T_{k_1} - T_{k_2})\|^2_{\Sigma_{stereo}}. \tag{6.17}$$

6.3.2 GNSS Factor

The GNSS measurement equation is given by

$$\mathbf{z}^{gnss}_t = h^{gnss}(\mathbf{x}_t) + \mathbf{n}_{gnss} \tag{6.18}$$

where $\mathbf{n}_{gnss}$ is the measurement noise assumed Gaussian distributed, and h^{gnss} is the measurement function, relating between the measurement $\mathbf{z}^{gnss}_t$ to the robot's position. Since only one state variable is involved in the measurement function, the above equation defines a unary factor:

$$f^{gnss}(\mathbf{x}_t) \triangleq \|\mathbf{z}^{gnss}_t - h^{gnss}(\mathbf{x}_t)\|^2_{\Sigma_{gnss}}. \tag{6.19}$$

An alternative incorporation of GNSS measurement factors is to use raw pseudo-range measurements from the GNSS receiver. However, this was not tested in this project.

Factor graphs with GNSS measurements from other sensors, operating at different rates, are shown in figure 6.5

6.4 Switchable Constraints for Robust Loop Closure

The ability to close the loop is both the biggest strength and weakness of SLAM. A false positive loop detection will result in a wrong topology in the underlying factor graph, which will again likely cause a catastrophic failure in the SLAM system's navigation capability.

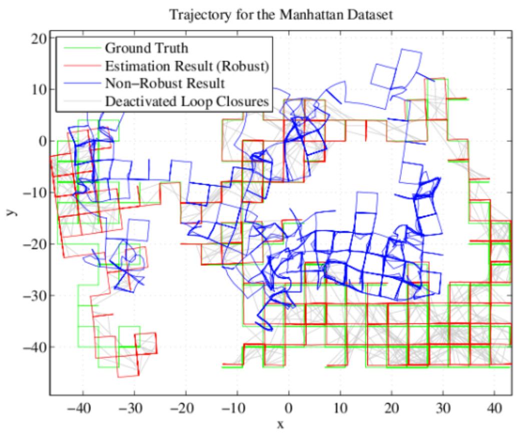


Figure 6.6: Pose graph of the Manhattan dataset corrupted by false positive loop detections which causes wrong estimation results. Using switchable loop constraints enables the optimizer to recover. Image courtesy of [66].

To prevent this, we can introduce a switch variable s_{ij} which induces a probability of whether a loop closure constraint between pose $\mathbf{x}_i$ and $\mathbf{x}_j$ is correct or not. We can then include this variable in the optimization, effectively enabling optimization over the topology of the graph itself and potentially allowing the SLAM system to recover from a false positive loop detection. The switch variable in combination with the loop constraint is called a switchable loop closure constraint, see figure 6.7, and was developed by Süderhauf and Protzel [66]. The implementation of the switchable constraint is available in an open-source library called Vertigo [66], which reportedly can deal with up to 1000 false positive loop closure constraints on various datasets.

Introducing the switch variable into the graph, the MAP estimation problem from (6.15) becomes:

$$X^*, S^* = \underset{X,S}{\arg\min} \underbrace{\sum_i \|f(\mathbf{x}_{i-1}, \mathbf{u}_i) - \mathbf{x}_i\|^2_{\Lambda_i}}_{\text{Odometry constraints}} + \underbrace{\sum_{ij} \|\Psi(s_{ij}) \cdot (f(\mathbf{x}_i, \mathbf{u}_{ij}) - \mathbf{x}_j)\|^2_{\Lambda_{ij}}}_{\text{Switchable Loop Closure Constraints}} + \underbrace{\sum_{ij} \|\gamma_{ij} - s_{ij}\|^2_{\Xi_{ij}}}_{\text{Switch Prior Constraints}} . \tag{6.20}$$

The activation function $\Psi : \mathbb{R} \longrightarrow [0, 1]$ maps s_{ij} to a probability, and can be chosen by the user. In [66] the best choice was found to be a linear function $\Psi^{\text{lin}}(s_{ij}) = s_{ij}$ constrained to $0 \leq s_{ij} \leq 1$. The idea behind the switch variables is that the influence of a loop closure constraint between the poses $\mathbf{x}_i$ and $\mathbf{x}_j$ can be removed by driving the associated switch variable s_{ij} to a value so that $\Psi(s_{ij}) \approx 0$.

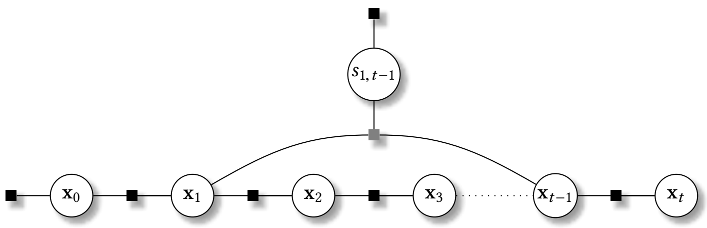


Figure 6.7: The pose factor graph from figure 6.4 with a switch variable $s_{1,t-1}$ which governs the loop closure factor (gray). Depending on the value assigned to the switch variable $s_{i,j}$, the loop closure factor is switched on or off, i.e. it is activated or deactivated as part of the optimization process. The switch variable is governed by a prior factor that penalizes the deactivation of loop closures.

To penalize the deactivation of loop closures, the switch variable are constrained by a prior $s_{ij} \sim \mathcal{N}(\gamma_{ij}, \Xi_{ij})$. Since it is reasonable to initially accept all loop closures, the prior mean γ_{ij} is always set to 1. A good general choice for the variance Ξ_{ij} was found empirically in the Vertigo paper to be 1.

6.5 Switchable Constraints for GNSS Measurements

Similar to the switchable constraint applied on a loop constraint factor, switchable constraints can also be applied to GNSS measurements. This is useful when GNSS measurements are affected by multipath artifacts, as these measurements can be completely wrong. The authors of the switchable constraint have applied this technique to GNSS based positioning by applying switch constraints on pseudo-range measurements [64] in a fixed lag smoothing framework. An open-source implementation of this approach is provided by Watson and Gross [69], [70].

As including all pseudo-range measurements and additional switch nodes into the factor graph quickly becomes expensive, a custom switchable constraint was developed that works on the GNSS measurement as a whole (not the pseudo-range measurements). A GNSS factor governed by a switch will result in the following minimization problem:

$$f^{gnss}(\mathbf{x}_{t-1}) \triangleq \|\Psi(s_{t-1}) \cdot (\mathbf{z}^{gnss}_{t-1} - h^{gnss}(\mathbf{x}_{t-1}))\|^2_{\Sigma_{gnss}} + \|\gamma_{t-1} - s_{t-1}\|^2_{\Xi_{t-1}}. \tag{6.21}$$

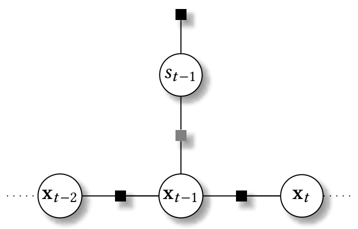


Figure 6.8: A section of a factor graph with a switch s_{t-1} which governs the GNSS measurement factor (gray). Depending on the value assigned to the switch variable $s_{i,j}$, the measurement factor is switched on or off, i.e. it is activated or deactivated as part of the optimization process. The switch variable is governed by a prior factor that penalizes the cost when disregarding a measurement.

6.6 Optimization

6.6.1 Choosing an Optimization Framework

For the navigation system to be able to operate in real time, the non-linear least squares problem in (6.11) has to be solved efficiently. In practice, one typically considers a linearized version of equation (6.11), which is obtained by using the measurement Jacobian found by linearizing the measurement function h_i around a given estimate Θ_i^0. For the SLAM problem the measurement Jacobian is sparse, and often has a specific structure. This insight is exploited by modern nonlinear optimization libraries for SLAM, of which maybe the two most popular is the g2o library by Kümmerle et al. [39], and the GTSAM library which contains implementations of the $\sqrt{SAM}$ [11], iSAM [35], and iSAM2 [34] algorithms. Both libraries are open-source, and can be found in [40], [17].

The g2o framework, as well as $\sqrt{SAM}$ are algorithms designed with the *full* SLAM problem in mind. That is, they operate in *batch* mode, solving the whole problem at once, *after* sensor data etc. has been collected. In contrast, incremental approaches like iSAM and iSAM2 are designed to accommodate the fact that a robot navigating in real time makes measurements at every time step, and manages to add new states to the estimation incrementally rather than rebuilding the data structures and recompute everything from scratch at every time step. This is especially useful for large-scale SLAM, where the number of variables can grow extremely large. In the case of loop closures, the iSAM2 algorithm utilizes a technique called fluid re-linearization, which can detect which variables are affected by the loop closure, and only re-linearize the affected ones.

Although g2o is the most efficient batch algorithm, as the authors show in the accompanying article [39], the incremental functionality of the GTSAM library seems very useful for a large-scale application like the one considered in this project. Additionally, I have found the GTSAM library to be better documented, with the tutorial by Dellaert [10] and extensively commented code. As factor graphs has become an unofficial standard in the SLAM literature, an additional advantage is that GTSAM uses a factor graph framework explicitly, in contrast to g2o which uses hyper-graphs.

Therefore, I have chosen to build the navigation system around the iSAM2 algorithm.

6.6.2 Optimization with iSAM2

For nonlinear measurement functions h_i, nonlinear optimization methods such as Gauss Newton or the Levenberg-Marquardt algorithm solve a succession of linear approximations to (6.11) in order to approach the minimum. We can linearize all measurement prediction functions h_i using the first order Taylor expansion

$$h_i(\Theta_i) = h_i(\Theta_i^0 \oplus \xi_i) \approx h_i(\Theta_i^0) + \mathbf{H}_i \xi_i \tag{6.22}$$

where ξ_i is the state update vector, and $\mathbf{H}_i$ is given by

$$\mathbf{H}_i \triangleq \left. \frac{\partial h_i(\Theta_i)}{\partial \Theta_i} \right|_{\Theta_i^0} \tag{6.23}$$

i.e., the measurement Jacobian evaluated at Θ_i^0, which is the estimate at the current iteration in the optimization.

By substituting (6.22) into (6.11), we obtain a linearized least squares problem in the state update vector ξ,

$$\begin{aligned} \xi^* &= \arg\min_{\xi} \sum_{i=0}^{|\mathcal{F}|} \|h_i(\Theta_i^0) + \mathbf{H}_i \xi_i - \mathbf{z}_i\|_\Sigma^2 \\ &= \arg\min_{\xi} \sum_{i=0}^{|\mathcal{F}|} \|\mathbf{H}_i \xi_i - \{\mathbf{z}_i - h_i(\Theta_i^0)\}\|_{\Sigma_i}^2 \end{aligned} \tag{6.24}$$

where $\mathbf{z}_i - h_i(\Theta_i^0)$ is the prediction error at the linearization point. Since we can write the Mahalanobis distance defined in (6.7) as

$$\|\mathbf{e}\|_\Sigma^2 \triangleq \mathbf{e}^T \Sigma \mathbf{e} = (\Sigma^{-\frac{1}{2}} \mathbf{e})^T (\Sigma^{-\frac{1}{2}} \mathbf{e}) = \|\Sigma^{-\frac{1}{2}} \mathbf{e}\|^2 \tag{6.25}$$

we can convert the weighted least squares problem in (6.24) into the standard least squares problem by defining

$$\mathbf{A}_i = \Sigma_i^{-\frac{1}{2}} \mathbf{H}_i \tag{6.26}$$

$$\mathbf{b}_i = \Sigma_i^{-\frac{1}{2}} (\mathbf{z}_i - h_i(\Theta_i^0)) \tag{6.27}$$

This is a form for whitening that eliminates the units of the measurements. We also obtain the standard least squares problem

$$\xi^* = \arg\min_{\xi} \sum_{i=0}^{|\mathcal{F}|} ||\mathbf{A}_i \xi - \mathbf{b}_i||^2 = \arg\min_{\xi} ||\mathbf{A}\xi - \mathbf{b}||^2 \tag{6.28}$$

where the state update vector $\xi \in \mathbb{R}^n$ contains all pose and landmark variables, the matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a large and sparse measurement Jacobian of all measurements, and $\mathbf{b} \in \mathbb{R}^m$ is the right-hand-side vector. $\mathbf{A}$ and $\mathbf{b}$ are obtained by collecting all $\mathbf{A}_i$ and $\mathbf{b}_i$ into one large matrix and right-hand-side vector, respectively. Solving this standard least squares problem, we can obtain an updated estimate $\Theta = \Theta^0 \oplus \xi$, which is then used as linearization point in the next iteration of the nonlinear optimization. The operator $\oplus$ is often a simple addition (as e.g. for landmark vectors), but for over-parameterized representations such as poses, it is defined as in equation (3.13).

By setting the derivative of $||A\Delta - b||^2$ to zero, we get the so called normal equations $A^T A \Delta = A^t b$. This equation system can be solved by Cholesky or QR decomposition of the information matrix $A^T A$ as is done in batch updates in $\sqrt{\text{SAM}}$, or incrementally in iSAM in combination with periodic variable reordering of the information matrix.

iSAM2, which is an improved version of the iSAM algorithm, avoids decomposing the information matrix by doing variable elimination on the factor graph and converting it to a Bayes net. The Bayes net is then further transformed to a Bayes tree, where each node encodes a density on fully connected clique variables, conditioned on its ancestors. This conditional structure allows the Bayes tree to be incrementally updated when a new measurement arrives. Only the cliques connected to the new factors and along the path to the root of the tree are removed an re-eliminated. Most of the tree is typically unaffected, allowing for efficient updates. The algorithm uses fluid re-linearization and partial state updates, which further reduces computational cost by detecting which variables that are affected when new measurements arrives, thus avoiding unnecessary re-linearizations and insignificant state updates. This can be thought of as an adaptive-lag smoother. Even though updates with iSAM2 are typically very fast, some updates, such as loop closures between the current state and a state far back in time will involve many cliques, and therefor take much longer. In addition, without some sort of sparsification, the factor graph will still grow unbounded in memory.

Chapter 7

Implementation

The goal of this project was to make a system capable of robust long term navigation by using on board cameras, to utilize other available sensors on Olav to correct for accumulated drift in the relative odometry estimates, and aid the re-initialization of the system in the case of tracking failure. Of the existing sensors on Olav, the GNSS sensors was evaluated to be the most complementary to visual navigation, since VO produces accurate relative pose estimates while the GNSS sensor provides global position estimates.

For the visual navigation, a GPU accelerated frame-to-frame VO system based on [28], [8] and [45] was developed, detailed in section 5.3. A frame-to-frame method was selected over a system based on BA, since the system must be able to operate long-term both temporally and spatially and should therefore try to reduce the amount of variables needed in the optimization.

To reduce the trajectory estimate's drift in a scenario where GNSS is unavailable, the visual navigation was augmented with a BoW-based loop-detector module, DBOW2 [23]. This module was chosen because of its ability to search and retrieve matches in a database of thousands of images in a matter of milliseconds. The loop detector is responsible for the M in the SLAM acronym, in the form that it builds a database that allows it to recognize previously visited locations.

To combine the results from the relative VO, loop-closing module and the GNSS measurements, a back-end optimization algorithm embedded in a factor graph framework, iSAM2, was chosen. The algorithm has an open source implementation in the GTSAM library, and I used version 4. Since the navigation system has to work in real-time, iSAM2 was considered to be particularly well suited because of its incremental approach to solving the SLAM problem by only re-linearizing variables affected by new measurements.

7.1 System Overview

A block diagram of the full system can be viewed in figure 7.1. The extracted FAST features is utilized in both the computation of relative constraints between frames and the loop detection where ORB descriptors are computed on the detected keypoints. If a loop is detected, the matched stereo frame is retrieved from a database of previous frames. The loop detection (including image retrieval) and the VO works in separate threads to save computation time. In the iteration subsequent to a detected loop, the retrieved stereo images are sent to a new VO object working in a separate thread where a relative pose constraint is computed between the previous and matched frame. In order to avoid conflicts when calling the GPU from different threads, different `cv::cuda::Stream` objects are used in each thread.

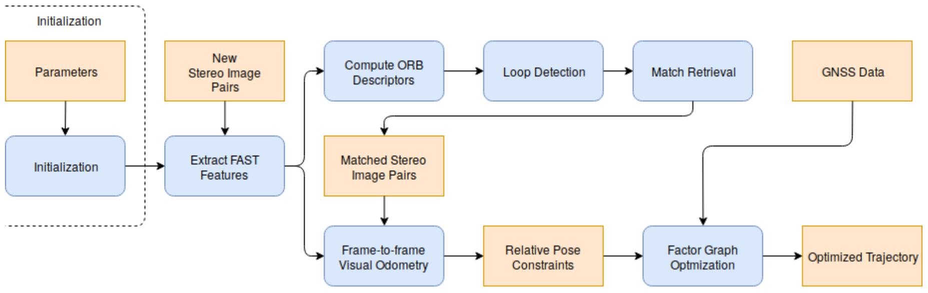
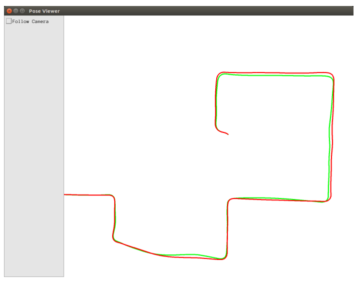
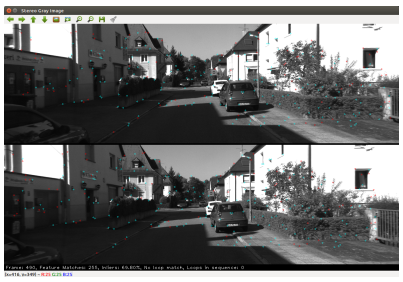


Figure 7.1: An overview of the complete SLAM system. Boxes with rounded corners indicate algorithmic processes and boxes with sharp corners indicate input/output data. The arrows indicate the data flow



(a) 3D Map Viewer.



(b) Frame Viewer.

Figure 7.2: Screen shot of the visualization tools developed in order to visually interface the system.

To visually interface the system, a map viewer was implemented using Pangolin [44] (see figure 7.2a), which is a rapid development library for managing OpenGl displays. The viewer displays the 3D-trajectory of the vehicle, and also provides the option of comparing to a ground truth trajectory, if provided. A stereo frame viewer, depicted in figure 7.2b, was also developed in OpenCV for convenience, which displays the current frames being processed along with the feature tracks between frames and other statistics.

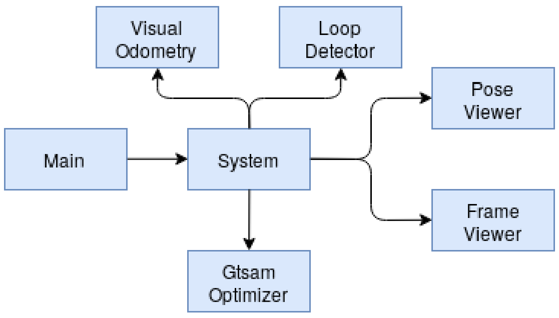


Figure 7.3: A block diagram representation of the code structure.

Figure 7.3 illustrates an overview of the code structure. The `System` class serves as an interface of the entire system, and is responsible for most of the logic and managing all the system threads. The calling main function is therefore only responsible for loading the initialization parameters to initialize the `System` class, which in turn initializes the rest of the system, and in the main loop load the stereo frames, timestamps and GNSS data to pass to the `System` class. By only providing an initialization function and a function to call in the main loop, this would hopefully reduce the amount of work needed later to implement a ROS interface needed for integration on Olav.

7.2 Sequence Diagrams

To better explain the sequential execution of the code, the three sequence diagrams below in figure 7.4, 7.5 and 7.6 illustrates calls between the classes in three different scenarios of a call to the `System` class from the main calling function.

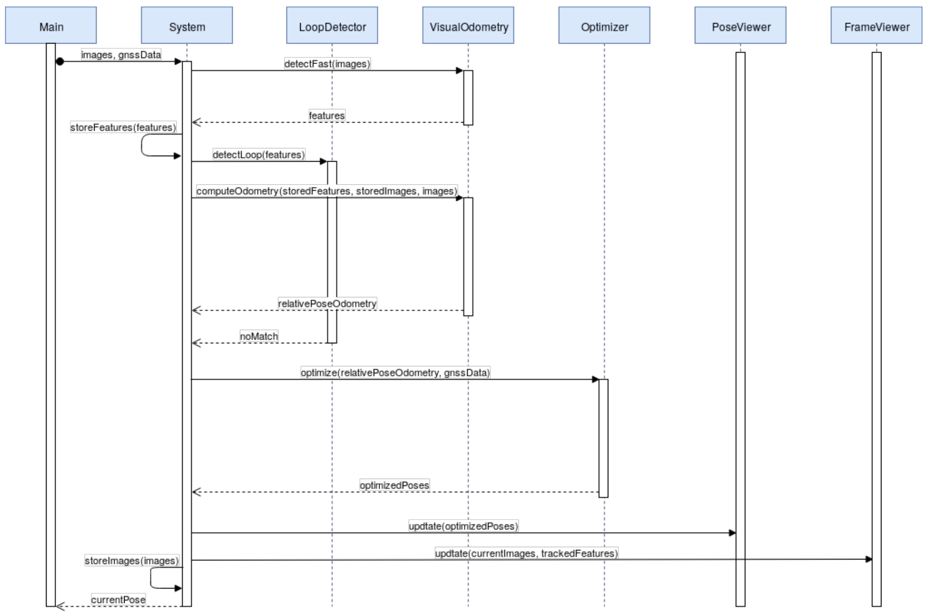


Figure 7.4: Sequence diagram of normal operation.

Figure 7.4 depicts the first scenario which is the most common. Here, no loops are detected and there is thus no need to retrieve any stored images, or to compute a loop-closure constraint. The system class stores extracted features and images for future iterations. The viewer classes are omitted from the other diagrams to avoid clutter.

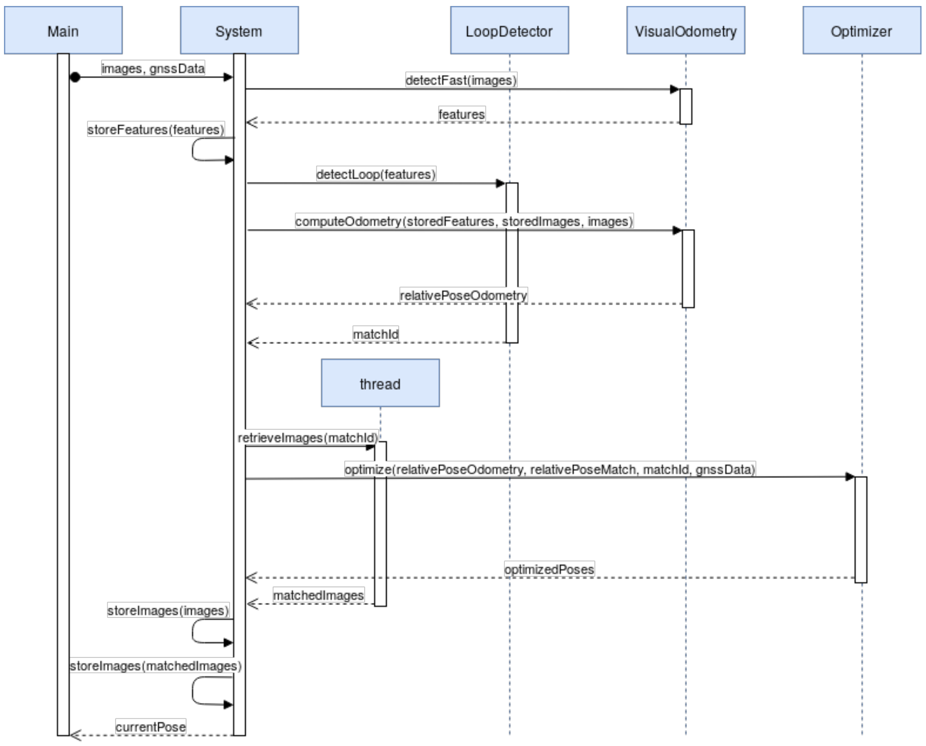


Figure 7.5: Sequence diagram of a scenario where a loop detection occurs.

The second scenario, in figure 7.5, a loop is detected. This launches another thread which retrieves the matched stereo frames based on the match id and stores it in memory for the next iteration depicted in figure 7.6. This happens concurrently with the graph optimization.

In the third scenario, in figure 7.6, a loop has been detected in the previous iteration, and the matched stereo frame is already loaded into memory. The `System` object then

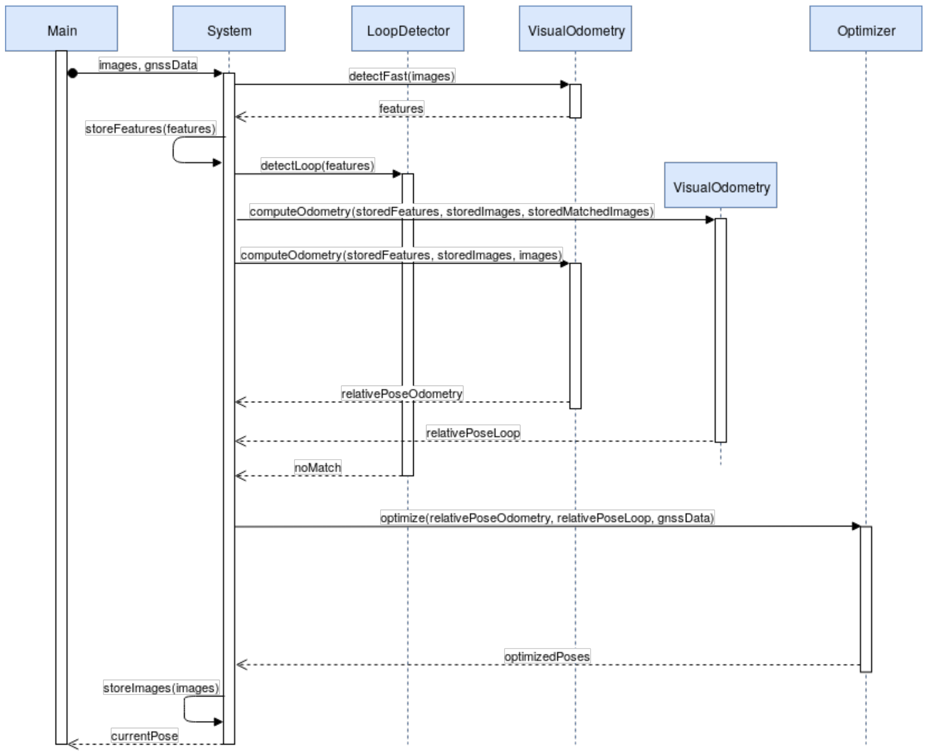


Figure 7.6: Sequence diagram of a scenario where a loop constraint is computed.

initializes a new `VisualOdometry` object in a new thread. This way, the loop detection, loop closure constraint and the odometry constraint are all computed concurrently in separate threads. The resulting loop closure constraint is then passed to the `Optimizer` object thread, which maintains the factor graph, together with the VO constraint and GNSS data.

7.3 GNSS Data Management and Global Frame Initialization

As explained in section 3.3.2, defining the tangent reference frame at time $t = 0$ provides the translational component $\mathbf{t}^{W_0}$ of the pose $\mathbf{T}^{W}_{B_0}$, but the rotation $\mathbf{R}^{W}_{B_0}$ is still unknown. Assuming a scenario where this is not available from gyroscope or magnetic measurements, several GNSS measurements with sufficient relative motion in between are needed.

Therefore, when the vehicle initially starts to move, a prior is applied on the first pose to have zero translation and identity rotation, and the pose estimation output is solely based on VO. Thus, the UGV will initially report its position relative to the initial frame. As the vehicle is moving, it obtains GNSS measurements which is put into a buffer which is continuously evaluated to check if the data has sufficient relative motion to initialize the pose. I set a minimum travel distance of 30 meters in the east and north direction, and a minimum of 3 measurements as a threshold criteria. When this threshold is met, the identity rotation prior on the first pose is removed, and the GNSS measurements are inserted from the buffer and into the graph. After optimization, all the poses are rotated into the global reference frame. This is illustrated in figure 7.7.

To visualize the output in the global frame, the system saves its trajectory in Latitude/Longitude (Lat/Long) coordinates to a .kml file when shutting down, which can be loaded directly into Google Maps.

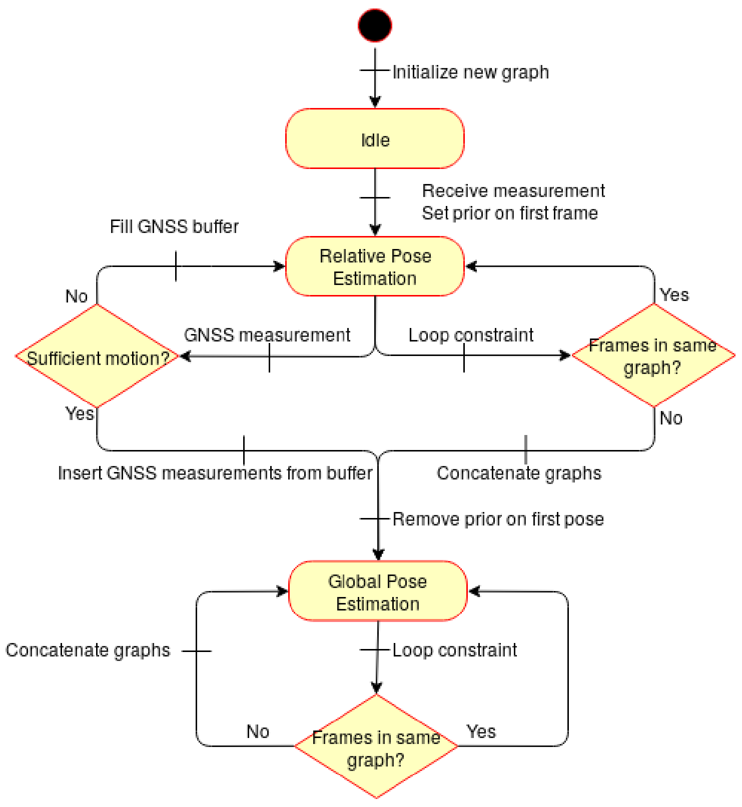


Figure 7.7: A state machine overview of the Optimization class. When the class is initialized, either on startup or when a tracking failure is signaled, a new graph is created and the initialization procedure begins.

7.4 Switchable Constraints

The switchable loop closure constraint was imported from the Vertigo open source library [65]. However, it had to be re-written to be compatible with GTSAM version 4.0, as the Vertigo implementation is based on an older version of GTSAM. The switchable GNSS constraint was implemented based on the `GPSFactor()` class in GTSAM and the linear switch variable from the Vertigo library. In the implementation, all loop closure constraints and GNSS measurements are governed by switch variables, but can easily be turned on and off in a configuration file.

7.5 Keyframe Insertion

Not all frames are inserted into the factor graph as this would require a lot of computational effort. Rather, only frames marked as a keyframes are included in the optimization. I chose a keyframe insertion strategy where a frame is selected to be a keyframe when the cumulative optical flow from the last keyframe reaches a certain threshold. This leads to fewer keyframes on straight lanes and more keyframes when going through corners, and no keyframe insertions if the robot is stationary. The reasoning behind this insertion strategy is that the odometry is expected to be less accurate when making turns, as the motion of each tracked feature is larger between consecutive frames, and matching features between frames in this case becomes more uncertain. The noise model between each key-frame is fixed, and so the estimated trajectory becomes more "flexible" in the turns, and less so on straight lanes when optimizing a loop closure.

Non-keyframes are stored with a reference to its respective keyframe, with a corresponding relative pose. Let $\mathcal{F}_{\hat{B}_t}$ denote the body coordinate frame at time t, where the $\hat{\cdot}$ notation indicates a keyframe. $\mathbf{T}^W_{\hat{B}_t}$ denotes the pose of that keyframe relative to the global origin. As mentioned, only the poses $\{\mathbf{T}^W_{\hat{B}_i}, \mathbf{T}^W_{\hat{B}_j}, \dots \mathbf{T}^W_{\hat{B}_t}\}$, i.e. poses corresponding to keyframes, are included in the optimization. The non-keyframe $\mathcal{F}_{B_{t+1}}$ is stored with a fixed transformation $\mathbf{T}^{\hat{B}_t}_{B_{t+1}}$ relative to its corresponding keyframe $\mathcal{F}_{\hat{B}_t}$, and to retrieve the pose $\mathbf{T}^W_{B_{t+1}}$ after an optimization step, $\mathbf{T}^W_{\hat{B}_t}\mathbf{T}^{\hat{B}_t}_{B_{t+1}}$ needs to be computed.

Every frame is inserted into the BoW database, and so matches can potentially occur between non-keyframes. The relative transformation between the matched frames is then propagated back to yield a constraint between their respective keyframes. If a loop is detected, there is no new keyframe insertion, so that the number of keyframes only grows with explored space. As an example, let's say a match occurred between the non-keyframes $\mathcal{F}_{B_{t+1}}$ and $\mathcal{F}_{B_{i+2}}$. The frame-to-frame odometry computes the transformation $\mathbf{T}^{B_{t+1}}_{B_{i+2}}$. This constraint is then propagated back to yield the constraint between their respective keyframes $\mathcal{F}_{\hat{B}_t}$ and $\mathcal{F}_{\hat{B}_i}$:

$$\mathbf{T}^{\hat{B}_t}_{\hat{B}_i} = \mathbf{T}^{\hat{B}_t}_{B_{t+1}} \mathbf{T}^{B_{t+1}}_{B_{i+2}} (\mathbf{T}^{\hat{B}_i}_{B_{i+2}})^{-1}. \tag{7.1}$$

7.6 Saving and Loading

In order to retain information from the factor graph and BoW database when turning the vehicle off and on, the class destructors calls their respective save functions which writes all the factors, estimated values of the nodes and the BoW database to file. If these files are provided when initializing the system, the SLAM session can continue where it was left off when booting the system again.

7.7 Recovery after tracking failure

If the VO processing fails, the system initializes a new factor graph and sets the first pose inserted into the factor graph to the origin. This is because when tracking fails, we no longer have information about its pose, and in the case of no GNSS measurements, we do not have information about its position. In GTSAM, all nodes in the factor graph are required to be connected, and my solution to this problem is to initialize a new graph and restart the global initialization procedure as explained in 7.3. This is illustrated in figure 7.7.

After the trajectory is initialized in the global reference frame, the graphs could be concatenated by inserting a between factor between the graphs based on their initial positions estimated from the GNSS data. However, there is little knowledge on the noise model between these initial positions in the two graphs, so the system continues to operate with two separate graphs. When a loop is detected, the system searches for the matched frames across all graphs. If the frames exist in separate graphs, the graphs are connected by a loop constraint and concatenated into one. If there are no GNSS measurements available, the graphs are still concatenated based on a loop closure, but the trajectory estimate will be reported relative to the origin of the earliest of the two graphs in chronological order instead of relative to the global frame.

After recovery, there is no constraint inserted at the point where track was lost since this would imply that information from the loop closure would have to be propagated all the way back to where tracking failed. Especially in the case of no GNSS measurements, this would result in a very uncertain constraint. To connect the trajectory at the point of tracking failure, the robot have to make another pass over that section of the trajectory to constrain the trajectory at that point.

The prior on the first pose after tracking failure can easily be changed to e.g. the last pose before track was lost, to report a more accurate position in the global frame. However as a sequence of frames could potentially fail to be processed, the prior was chosen to the origin to indicate that the system is in fact recovering from a tracking failure, and we can say with a quantifiable uncertainty what the position is relative to the initial pose in the new graph.

Chapter 8

Experiments and Results

In the following chapter, a brief explanation of the method evaluation is provided followed by the generated results. The results are presented by scenario, and for each scenario, a brief discussion is provided. All results was generated on a laptop computer with an Intel Core i7 8-core processor running at 2.9 GHz with 32 GB of memory. The laptop is also equipped with a NVIDIA Quadro M2200 GPU. Note that all results presented in this chapter is from running experiments once with only one application running on the computer, and this causes variations in results based on a single iteration such as maximum execution time.

8.1 Method of Evaluation

To evaluate the quality of the estimated trajectory, I used the trajectory evaluation toolbox by Zhang and Scaramuzza [75] [76]. This toolbox has support for calculating Absolute Trajectory Error (ATE), which is an error metric commonly used in the SLAM community as it produces a single metric which makes it easy to compare performance of different algorithms for a given trajectory. The toolbox assumes that the temporal correspondence between the estimate and the ground truth has been established.

Calculating the ATE starts with aligning the estimated trajectory with the ground

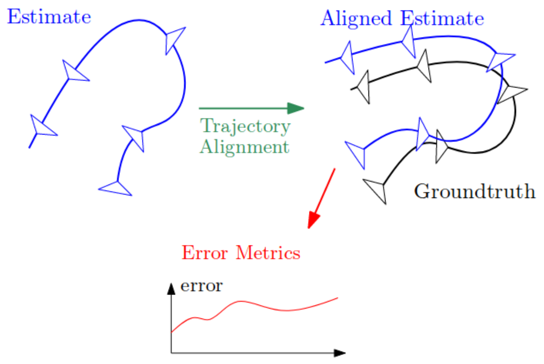


Figure 8.1: The process trajectory evaluation. First, the estimated trajectory (blue) needs to be aligned with the ground truth (black). Then, the trajectory estimation error can be calculated from the aligned estimate and the ground truth using certain error metrics.

truth, which is a non-trivial task since they are often given in different coordinate systems. The trajectories are essentially aligned by minimizing the term

$$\{\mathbf{R}^*, \mathbf{t}^*\} = \arg\min_{\mathbf{R},\mathbf{t}} \sum_{i=0}^{N-1} \|\mathbf{p}_i - \mathbf{R}\hat{\mathbf{p}}_i - \mathbf{t}\|^2 \tag{8.1}$$

where $\{\hat{\mathbf{p}}_i\}_{i=0}^{N-1}$ and $\{\mathbf{p}\}_{i=0}^{N-1}$ are the estimated and ground truth positions respectively, and $\{\mathbf{R}^*, \mathbf{t}^*\}$ is the optimal estimated rotation and translation between the reference frames. After calculating the optimal relative rotation and translation, the aligned trajectory estimate is then given by:

$$\hat{\mathbf{p}}'_i = \mathbf{R}^*\hat{\mathbf{p}}_i + \mathbf{t}^*, \qquad \hat{\mathbf{R}}'_i = \mathbf{R}^*\hat{\mathbf{R}}_i. \tag{8.2}$$

The trajectory alignment is described in more detail in Algorithm 1 in [76]. After alignment, the rotation and translation error at timestep i is given by

$$\Delta \mathbf{R}_i = \mathbf{R}_i(\hat{\mathbf{R}}_i')^T, \tag{8.3}$$

$$\Delta \mathbf{p}_i = \mathbf{p}_i - \Delta \mathbf{R}_i \hat{\mathbf{p}}_i' \tag{8.4}$$

respectively. The absolute rotation and translation error can then be found by calculating:

$$\mathrm{ATE}_{\mathrm{rot}} = \left(\frac{1}{N} \sum_{i=0}^{N-1} ||\angle(\Delta \mathbf{R})||^2 \right)^{\frac{1}{2}} \tag{8.5}$$

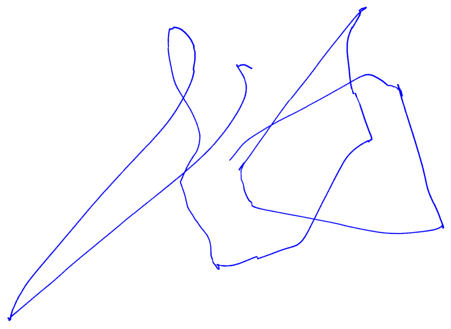
$$\mathrm{ATE}_{\mathrm{pos}} = \left(\frac{1}{N} \sum_{i=0}^{N-1} ||\Delta \mathbf{p}_i||^2 \right)^{\frac{1}{2}} \tag{8.6}$$

where $\angle(\cdot)$ means converting the rotation matrix to angle axis representation and using the rotation angle as the error.

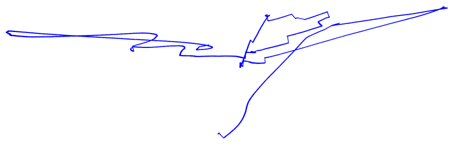
8.2 FFI Dataset

The visual odometry algorithm was tested on a recorded dataset from Olav. The sequence is recorded on a dirt road, and starts and ends in in approximately the same position and orientation. The resulting trajectory is depicted in figure 8.2. The result is of very poor quality, and the algorithm particularly struggles with estimating rotation, as can be seen in the figure where the estimated trajectory is jagged and non-smooth. The estimated initial and final position is estimated to be 156 meters apart, with a 56 meter offset in height which results from the poor rotation estimates which also affects pitch and roll. When introducing loop closure, the optimization crashes as the iSAM2 optimization module throws an indeterminant linear system exception.

The reason for the poor performance is because of incorrect stereo camera calibration. Several attempts was performed with different calibration sequences with both chessboard- and Apriltag calibration patterns. Both the calibration package from



(a) Top view.



(b) Side view.

Figure 8.2: Overview of the trajectory generated by the VO algorithm on the FFI dataset.

OpenCV as well as the Kalibr toolbox was used, but neither one could find a correct rectification mapping so that the epipolar constraint was satisfied. The best obtained calibration result was from the Kalibr toolbox, which reported an expected reprojection error of [0.000089, -0.000202] +- [0.087093, 0.070710] pixels, however when aligning the rectified stereo images next to each other and drawing horizontal (epipolar) lines on top of them, as shown in figure 8.3, one can see that the epipolar error is several pixels, especially when comparing the left side of the images.

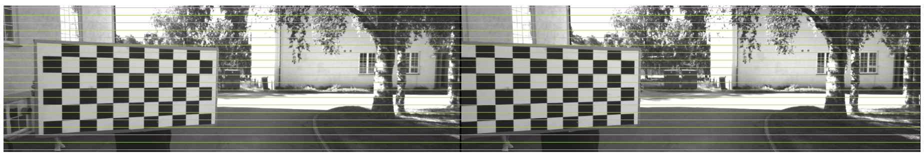


Figure 8.3: Horizontal lines on rectified images from the FFI dataset.

Looking at the locations of removed outliers in figure 8.4, one can see that the Kalibr toolbox has discarded several of the keypoints (chessboard corners) in the left side of the images, which could have contributed to the poor fit of the calibration model in this region. Similar results was obtained from the other calibration sequences.

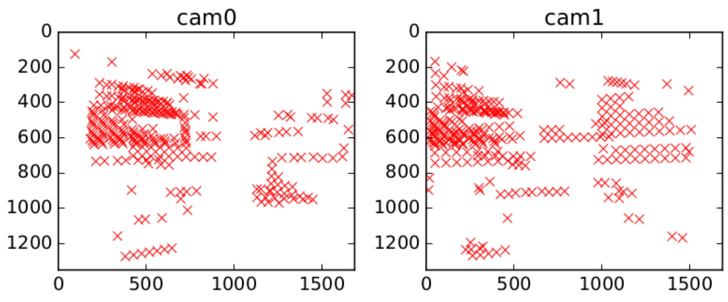


Figure 8.4: Locations of removed outlier corners from the calibration procedure with Kalibr.

As the visual odometry is the only component that provides relative motion between GNSS measurements in the developed system, further testing with GNSS was not performed with the dataset from Olav, as the necessary foundations to provide reasonable navigation results were not present.

8.3 KITTI Dataset

After many attempts of trying to get the datasets from Olav to work, the publicly available KITTI dataset published by Geiger et al. [25] was considered to be a good substitution because its similar sensor setup to Olav. All further test results presented here is from this dataset. It was recorded with two Point Grey Flea2 (FL2-14S3M-C) cameras equipped with global shutters, which has a bit lower resolution (1.4 Megapixels) and a higher frame rate (10 fps) than the Grasshopper 3 cameras on Olav. The dataset was also recorded with a high-quality INS (OXTS RT3003) which is RTK capable.

The sensor platform is a standard VW Passat, depicted in figure 8.5, and all the recordings are from public roads in Karlsruhe, Germany. Urban areas and traffic are conditions that Olav's navigation system should be able to handle, and the similarity in the sensor-setup, as well as the fact that the data comes pre-calibrated, rectified and synchronized, makes this dataset suited as a test set.

The KITTI dataset has 11 sequences (sequence 00-10) which is provided with a ground truth trajectory, 6 of which contains one or more loop closure events. Sequence 00 is one of the longer sequences which contains several loop closure events, which makes this sequence more flexible for testing loop detection and re-localization. Therefore, I will especially focus on this sequence when evaluating the results.

8.4 Visual Odometry

To evaluate the performance of the visual odometry module, I compared the developed system with the well known frame-to-frame VO method, LIBVISO2. Tables 8.1 and 8.2 compares the ATE for the two systems.

Looking at the tables, where the best results are marked in bold, we can see that the developed system performs better on most sequences. LIBVISO2 has a slight edge over the developed system on the two shortest sequences, however, these sequences

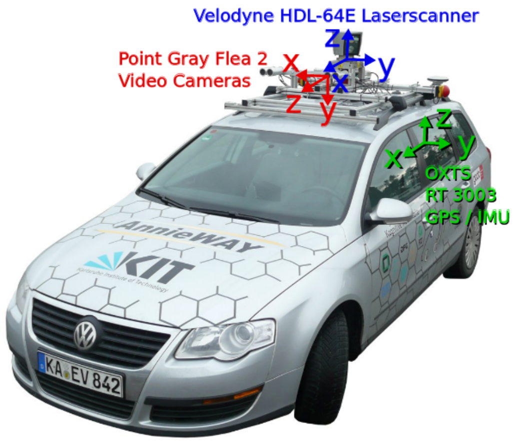

Velodyne HDL-64E Laserscanner
Point Gray Flea 2 Video Cameras
OXTS RT 3003 GPS / IMU
x
y
z

Figure 8.5: A VW Passat is the sensor platform used in the KITTI dataset. The location of the cameras and INS system is illustrated. Image courtesy of [25].

are very short and since the developed system performs very similarly, this could be coincidental. The developed system generally performs better on the longer sequences, except for sequence 01 and 06. On sequence 06 where the vehicle drives in an oblong loop, the developed system under-estimates the second turn, and so it drifts further and further away from the ground truth after this. The trajectory estimate aligned with the ground truth can be seen in figure 8.6.

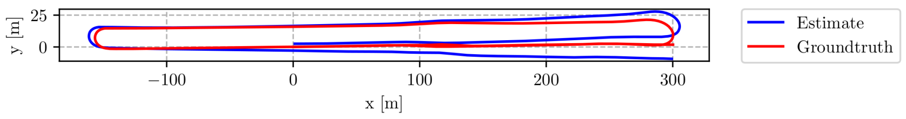


Figure 8.6: The trajectory estimate from the KITTI 06 sequence aligned with the ground truth. The algorithm under-estimates the second of the two turns in the sequence, which makes the estimate drift further and further away from the ground truth after this turn.

On sequence 01, both algorithms accumulates a large error, but the developed

		Developed System		LIBVISO2	
Sequence	Length [m]	ATE_{pos} [m]	σ [m]	ATE_{pos} [m]	σ [m]
00	3724	**8.60**	**4.06**	31.01	15.76
01	2454	55.83	20.64	**38.34**	**7.76**
02	5066	**19.38**	**6.39**	38.73	15.52
03	560	2.36	0.93	**1.88**	**0.75**
04	392	1.61	0.73	**0.92**	**0.41**
05	2204	**4.98**	**1.79**	12.50	6.79
06	1232	6.17	2.76	**4.12**	**1.91**
07	694	**4.06**	**2.02**	5.84	3.38
08	3222	**6.61**	**4.12**	20.92	8.46
09	1704	**13.19**	**6.59**	17.37	10.57
10	918	**2.78**	**1.32**	3.93	1.79

Table 8.1: Absolute translation error statistics from LIBVISO2 and the visual odometry part of the developed system.

system fail "harder". Figure 8.7 shows a screen-shot from this sequence, which is from a highway environment. The feature detector struggles with finding features in the sky and in the gray asphalt which covers most of the image. When other cars pass by, a lot of features gets detected on those objects, and as a result the algorithm estimates its motion relative to passing objects rather than the road. This leads to several incidents where the developed VO system under-estimates the translation, which can be seen in figure 8.8. Normally the algorithm considers features detected on moving objects as outliers, but in this case there are so few detected features that features on moving objects are no longer rejected by the RANSAC algorithm.

Looking closer at the 00 sequence, a comparison of the aligned trajectories from the developed system and LIBVISO2 is provided in figure 8.9, as well comparisons of the translation and rotation error per frame in figure 8.10 and 8.11 respectively. Note that the estimates are aligned with the ground truth to minimize the overall error, and so the errors are non-zero in the first frames. The spikes in yaw error at approximately

		Developed System		LIBVISO2	
Sequence	Length [m]	ATE_{rot} [deg]	σ [deg]	ATE_{rot} [deg]	σ [deg]
00	3724	**2.62**	**1.38**	7.41	3.09
01	2454	**7.79**	5.82	8.70	**4.03**
02	5066	**4.37**	**2.26**	8.64	3.36
03	560	1.49	0.80	**1.20**	**0.58**
04	392	**1.23**	0.18	1.50	**0.09**
05	2204	**2.46**	**0.99**	6.32	2.36
06	1232	**2.58**	**0.60**	3.63	1.56
07	694	**2.02**	**0.70**	4.78	1.45
08	3222	**3.23**	**1.55**	7.47	2.45
09	1704	**2.25**	**0.77**	7.78	2.61
10	918	**2.25**	**0.92**	3.78	1.60

Table 8.2: Absolute rotation error statistics from LIBVISO2 and the visual odometry part of the developed system.

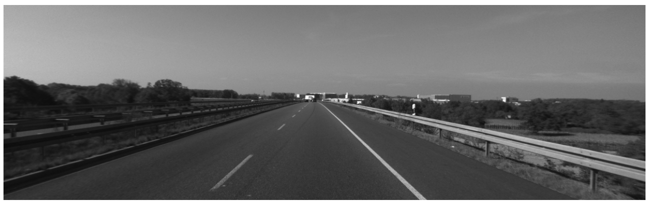


Figure 8.7: Screenshot from the KITTI 01 sequence.

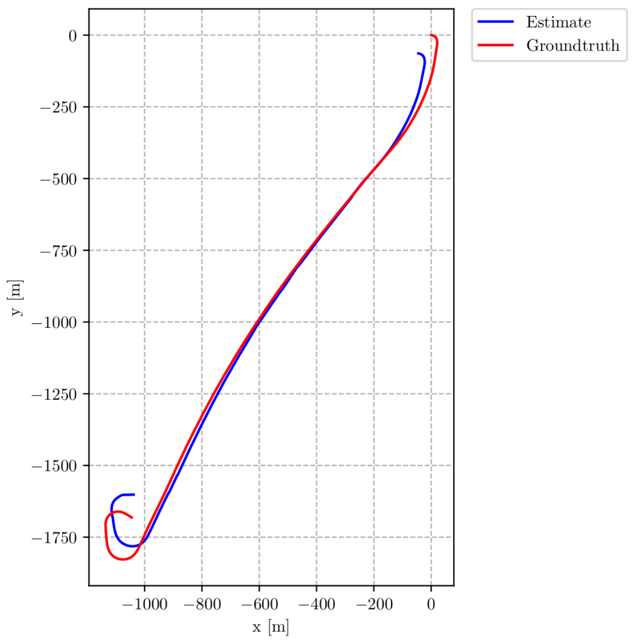


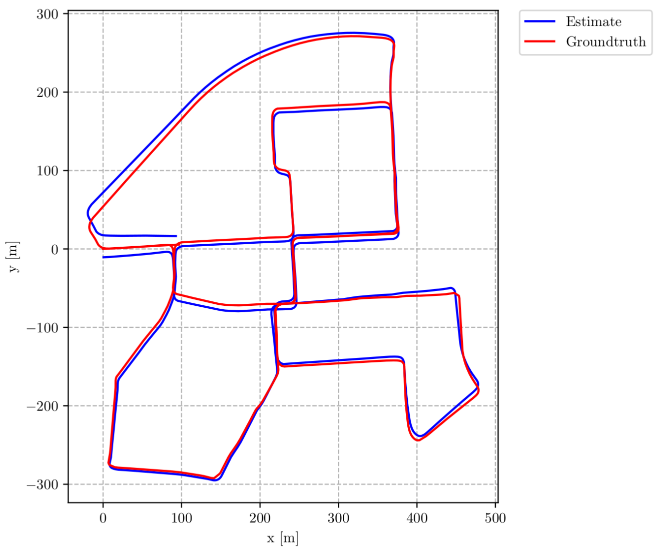
Figure 8.8: Trajectory estimate from the KITTI 01 sequence aligned with the ground truth. As there are so few features detected, the algorithm ends up estimating its motion relative to passing cars. This leads to a trajectory estimate that is too short compared to the ground truth.

1450 and 2300 meters happens for both algorithms, and as there is nothing particular happening in those corresponding frames, this seems to be an error in the ground truth which is estimated from a separate INS system. There appears to be a bias in the rotation yaw estimates with the LIBVISO2 algorithm, which was present in several of the other sequences as well. Both algorithms seems to struggle with the last part of the sequence which contains a long gentle turn to the left, which goes from coordinate $(x, y) = (470, 275)$ to $(-10, 40)$ in figure 8.8. In this part of the sequence, there is a hedgerow on the right side close to the road, which causes the optical flow on the right side of the image to be greater than the left side. This may contribute to the drift in the estimates.

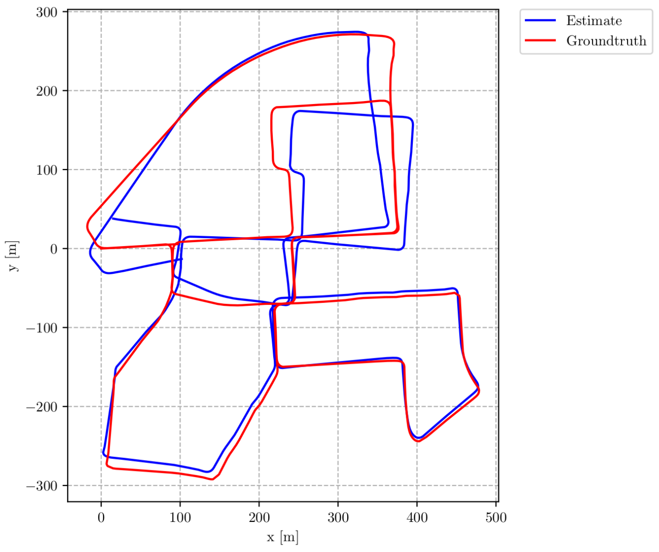
Looking at the execution time of each system in figure 8.12a, we see that both systems maintain a relatively constant execution time over the sequence, as both algorithms are frame-to-frame. From table 8.3 we see that on average, the developed system can process frames at 47.6 fps, while LIBVISO2 is only able to maintain 10.9 fps, which is just above the frame rate of the KITTI dataset. Of course, LIBVISO2 is not GPU-accellerated so the comparison is not fair, but this was the main reason for developing a new visual odometry algorithm as this could utilize the GPU which Olav is equipped with. When adding loop detection and the iSAM2 back-end, the execution time rises and this would leave a system based on LIBVISO2 unable to satisfy the real-time constraint.

	Developed System	LIBVISO2
Mean [s]	**0.0206**	0.0916
Max [s]	**0.0346**	0.1589
Min [s]	**0.0125**	0.0699
σ [s]	**0.0031**	0.0106

Table 8.3: Execution time statistics from LIBVISO2 and the developed VO system on the KITTI 00 sequence.

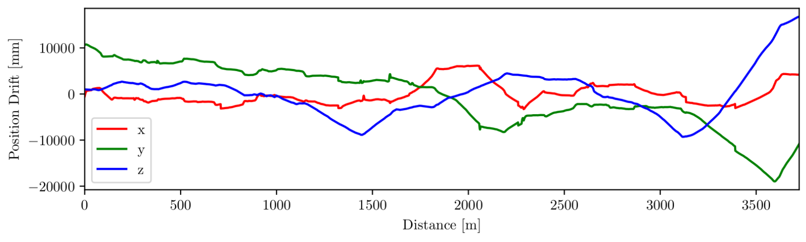


(a) Developed VO system.

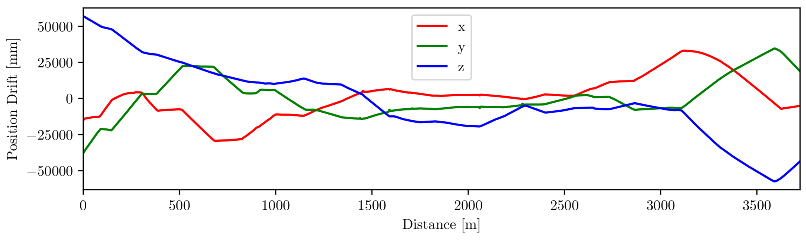


(b) LIBVISO2.

Figure 8.9: Comparison of trajectory estimates aligned with the ground truth on the KITTI 00 sequence.

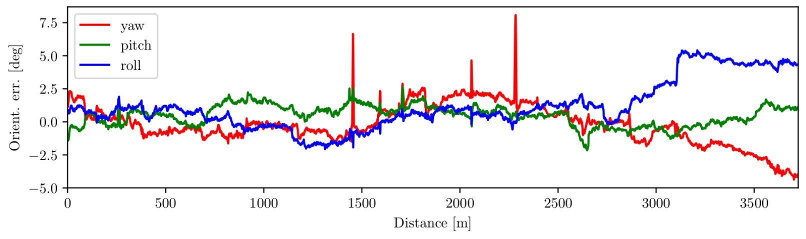


(a) Developed VO system.

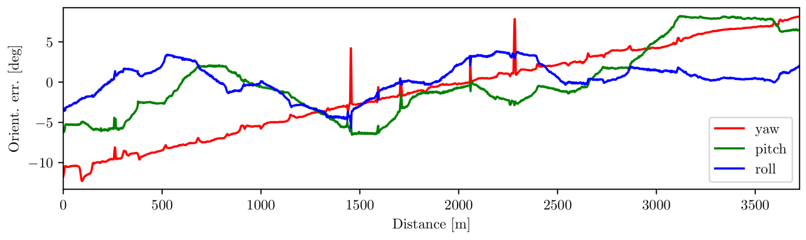


(b) LIBVISO2.

Figure 8.10: Comparison of the translation error per frame on the KITTI 00 Sequence.

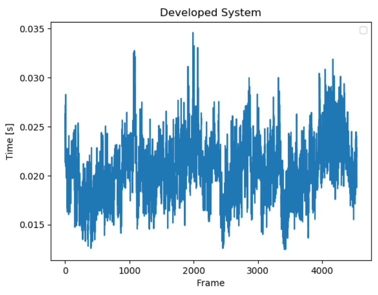


(a) Developed VO system.

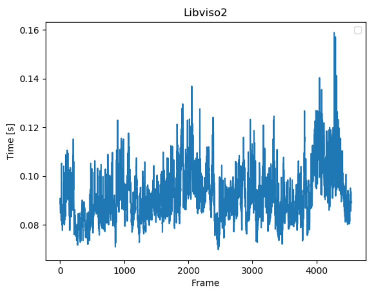


(b) LIBVISO2

Figure 8.11: Comparison of the rotation error per frame on the KITTI 00 Sequence.



(a) Developed VO system.



(b) LIBVISO2.

Figure 8.12: Comparison of the execution time of the developed VO system and LIBVISO2 on the KITTI 00 sequence.

8.5 Visual-SLAM

Turning on the loop closure module, the developed VO system becomes a SLAM system. The SLAM system was tested on the 6 sequences in the KITTI dataset that contains loop events, and the result from 5 of them is listed in table 8.4. Sequence 09 also contains a loop event between the first and last frame, but since the loop detection algorithm requires 3 consistent matches in a row to accept a loop closure, the loop event was not added as a loop constraint in this case. From table 8.4, we see that the ATE in translation and rotation is reduced in all sequences compared to the results obtained from the visual odometry.

The trajectory estimate from the SLAM algorithm on sequence 00 is aligned with the ground truth in figure 8.13, together with the translation and rotation errors in figure 8.14a and 8.14b, respectively. Looking at the rotation error, we can see that its reduced overall, but the error in the yaw is more jagged. This is partially because the keyframe poses are now optimized to satisfy the loop constraints while the non-keyframes are stored with fixed transformations relative to their respective keyframes.

Looking at the execution time of the DBOW2 module in figure 8.15, which includes database query and geometric verification, we see that in figure 8.15a, the execution time exceeds 0.6 seconds at approximately frame 4100. This is because the BoW

		Translation		Rotation	
Sequence	Length [m]	ATE_{pos} [m]	σ_{pos} [m]	ATE_{rot} [deg]	σ_{rot} [deg]
00	3724	3.14	1.13	1.73	0.68
02	5066	11.46	4.98	0.69	0.92
05	2204	2.48	1.00	1.20	0.51
06	1232	3.74	1.76	2.52	1.68
07	694	1.40	0.47	1.50	0.85

Table 8.4: ATE statistics from the SLAM part of the developed system.

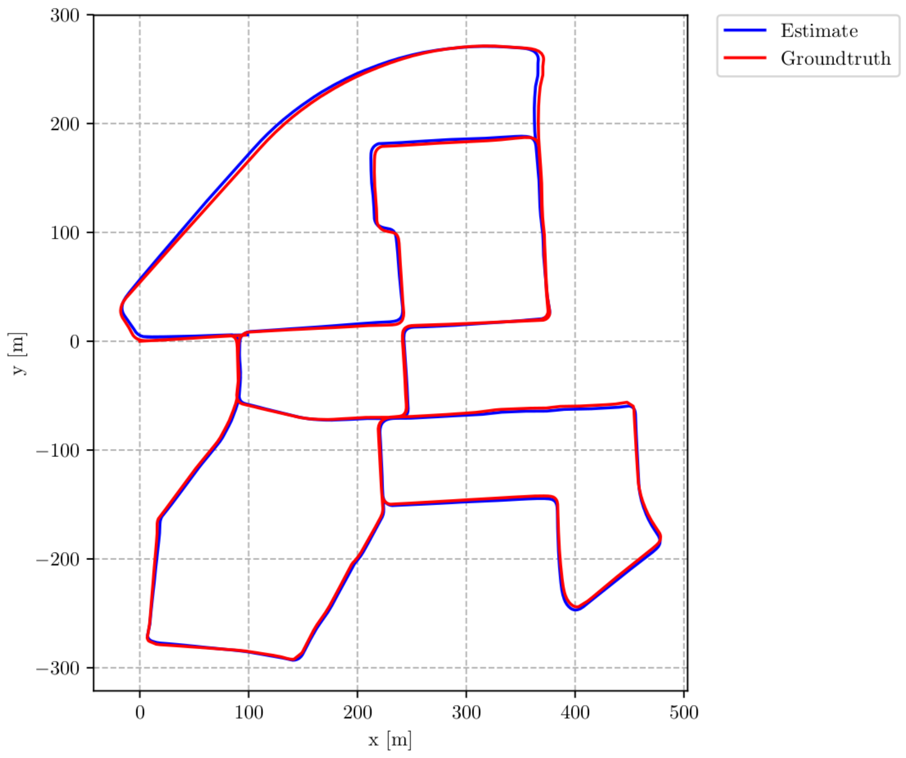
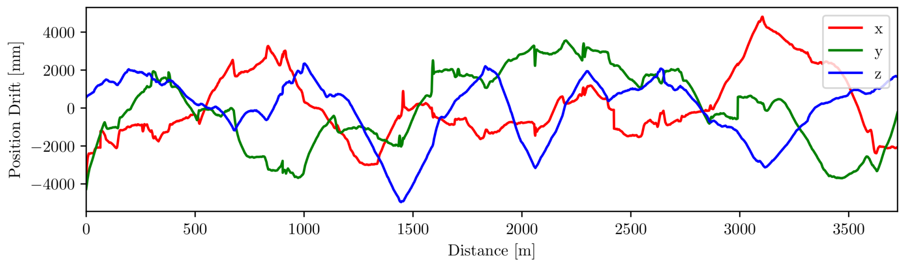
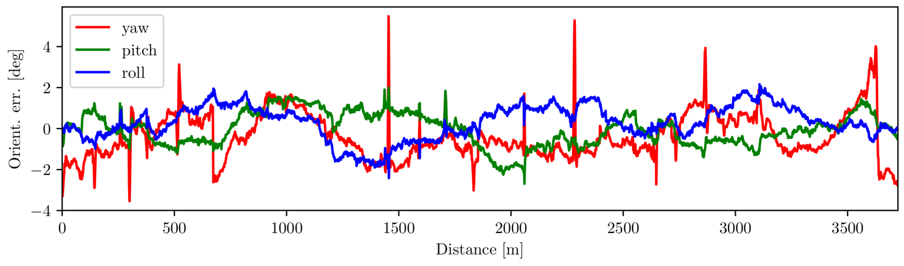


Figure 8.13: Estimated trajectory aligned with the ground truth for the V-SLAM system on the KITTI 00 sequence.



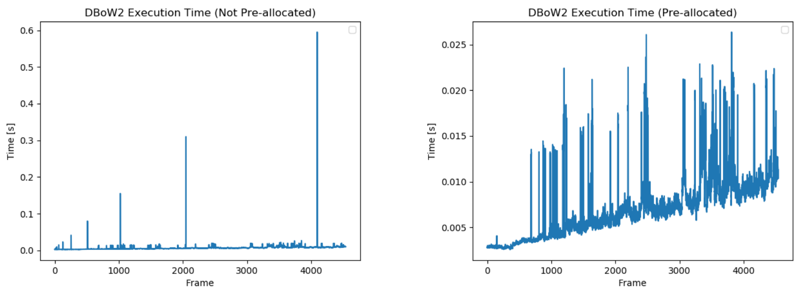
(a) Translation error.



(b) Rotation error.

Figure 8.14: Trajectory estimate errors per frame for the developed V-SLAM system on the KITTI 00 sequence.

database structure grows as more frames are added, and if it is not pre-alllocated, this causes huge penalties in the execution time. When pre-allocating this structure, see figure 8.15b, the execution time of this module peaks at 26 milliseconds.



(a) Not pre-allocating the BoW database leads to a worst case execution time of 0.6 seconds.

(b) Pre-allocating the BoW database reduces the worst case execution time to 26 ms.

Figure 8.15: Execution time of the DBOW2 loop detection module per frame.

The frame retrieval execution time in figure 8.17 remains relatively constant, with a mean of 9 ms and maximum of 12 milliseconds. This plot also indicates where the system accepted a loop closure. A big contributor to the maximum execution time is the optimization step which peaks at 38ms. The optimization execution per frame can be seen in figure 8.16.

The total execution time in figure 8.18 has an average of 28 ms, but peaks at 115 ms which is above the real time constraint of 100 ms for the KITTI dataset. It is still below the real time constraint of 167 ms for the FFI dataset, but looking at the graph, the execution time is expected to rise as more keyframes are added. The execution time spikes when loop closure constraints are processed, and there are a lot of variables that needs to be re-linearized. Note that the execution time includes the switchable constraint discussed in section 6.4.

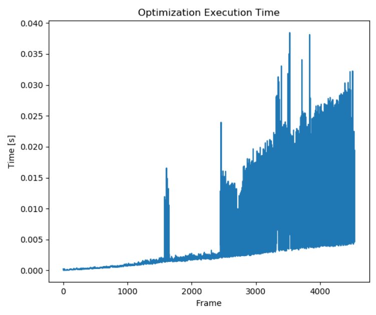


Figure 8.16: Optimization execution time with loop closure. Mean: 3ms. Max: 38ms.

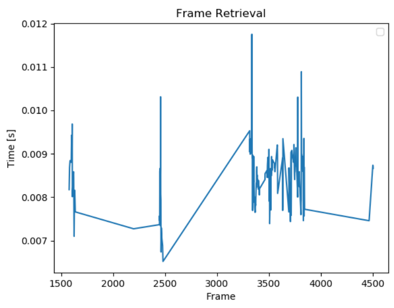


Figure 8.17: Execution time of the frame retrieval. Mean: 9ms. Max: 12ms.

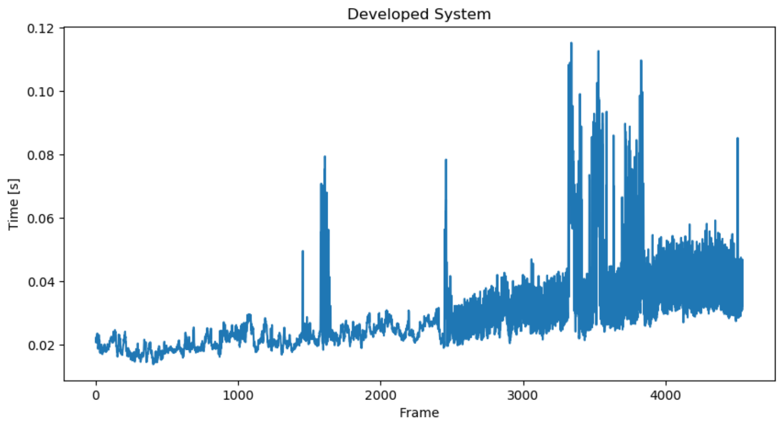


Figure 8.18: Overall execution time of the developed SLAM system, without GNSS measurements. Mean is 28ms, max is 115ms.

8.6 GNSS

The GNSS measurements provided with the 00 sequence is mostly corrected with RTK signals (see [25] for detailed information), which means that they are very precise. The largest reported error from the INS system in the horizontal plane is 38 cm. To test the integration of GNSS measurements into the factor graph, I applied GNSS measurements on every n^{th} keyframe, and the results from this is listed in table 8.5. We can see that the system can provide accurate global estimates with only 5 GNSS measurements on the 3724 meter long sequence when only providing measurements every 500^{th} keyframe. There is no visible penalty in the execution time by including the GNSS measurements compared to the results from V-SLAM-only in the previous section. All results listed in table 8.5 is from using the switchable GNSS constraint from section 6.5. One potential problem is that the estimate makes a large jump when it is initialized in the global frame, and this could cause problems when controlling the vehicle based on the estimate.

	Translation		Rotation		Execution Time	
Period [KF's]	ATE_{pos} [m]	σ_{pos} [m]	ATE_{rot} [deg]	σ_{rot} [deg]	Mean [ms]	Max [ms]
1	1.32	0.56	1.46	0.83	28	114
10	1.33	0.56	1.47	0.83	27	95
100	1.36	0.93	1.39	0.73	28	118
500	2.52	1.08	1.77	0.81	28	121

Table 8.5: Absolute translation and rotation error statistics on the KITTI 00 sequence with GNSS measurements every x^{th} keyframe, where x is given in the first (Period) column.

With global information, the estimated trajectory can now be plotted in a map. Figure 8.19 shows the estimated trajectory with GNSS measurement every 500^{th} keyframe laid over a satellite photo in the Google Maps application.

To test the system's ability to deal with noisy GNSS measurements, I performed three experiments. In all three, I applied (switchable) GNSS measurements on every keyframe, and in each experiment the measurements was perturbed with an additive

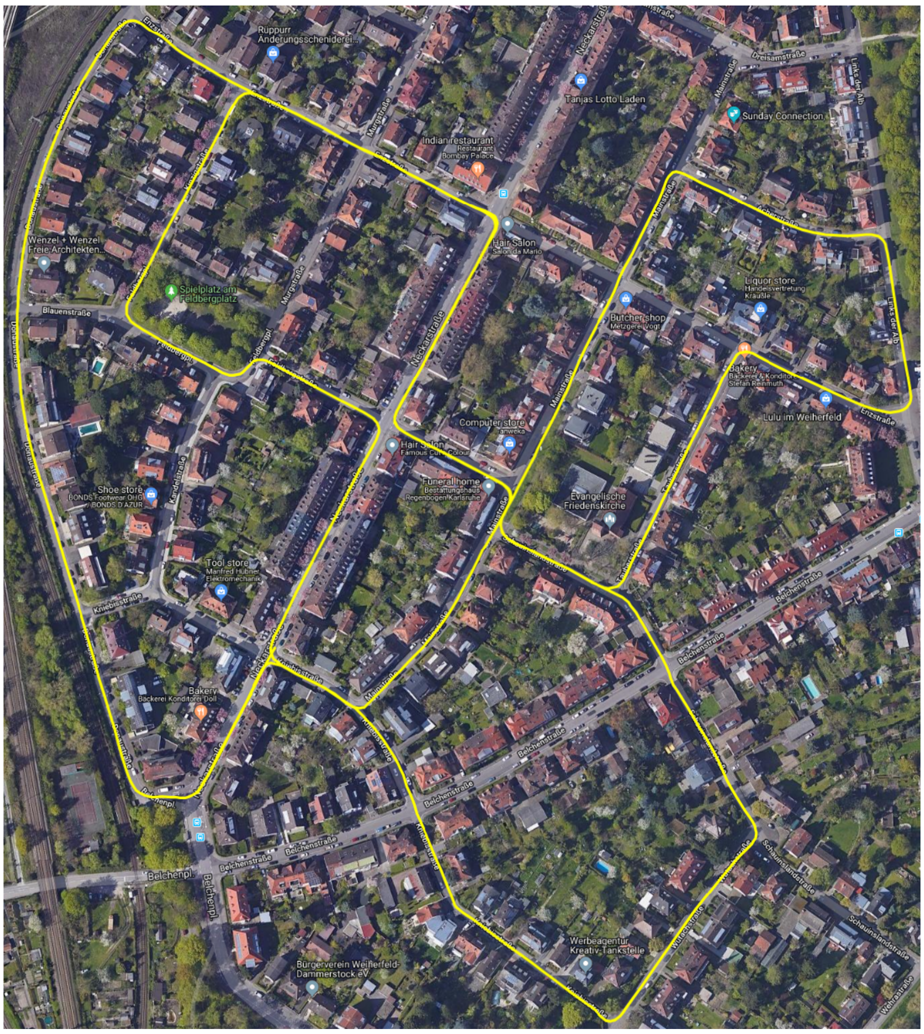


Figure 8.19: Trajectory estimate from SLAM with GNSS measurement every 500th keyframe plotted over a satellite photo.

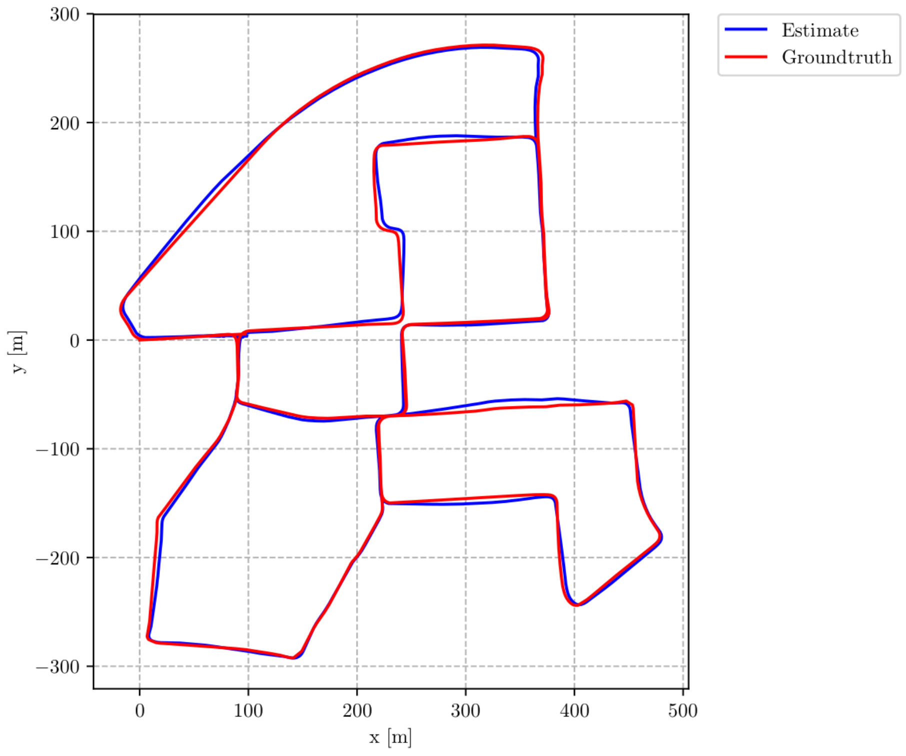


Figure 8.20: Trajectory estimate aligned with the ground truth from experiment with GNSS measurements, perturbed by a 30 meter diagonal Gaussian noise, every keyframe.

Gaussian noise with a standard deviation of respectively 10, 20, and 30 meters in every direction. In each experiment, the noise model on the GNSS measurement was adapted to the additive noise. The results are listed in table 8.6, and the trajectory estimated from the noisiest measurements is aligned with the ground truth in figure 8.20. As expected, the accuracy gets worse as the noise on the measurements increases, but the estimate is more accurate when provided GNSS measurements perturbed with the 10 meter Gaussian than with SLAM only. Also, the execution time increases compared to the experiment with unperturbed measurements in table 8.5.

	Translation		Rotation		Execution Time	
Noise σ [m]	ATE_{pos} [m]	σ_{pos} [m]	ATE_{rot} [deg]	σ_{rot} [deg]	Mean [ms]	Max [ms]
10	2.37	0.84	3.31	1.50	31	150
20	4.34	2.09	6.59	3.15	31	151
30	4.71	2.37	6.82	3.40	31	147

Table 8.6: Absolute translation and rotation error statistics on sequence 00 with noisy GNSS.

8.7 Switchable Loop Constraints

Because of how the loop constraint is computed, a false positive loop detection normally does not result in a false loop constraint. For a false positive match, the frame-to-frame odometry is normally not able to compute a loop closure constraint based on a false positive loop closure detection. This is because the feature tracker is only able to find very few matches which results in failure in the 5-pt algorithm, which again leads to the loop constraint being discarded. During testing, there were some false positive matches, but none of these resulted in a loop constraint.

To test the switchable loop constraint, I bypassed the relative pose estimation step between the matched frames and added the false constraint directly between frame 300 and the origin. The experiment was performed without GNSS measurements. The resulting trajectory estimate achieved an $\text{ATE}_{\text{pos}} = 2.76$ meters in translation and $\text{ATE}_{\text{rot}} = 1.75$ degrees in rotation, which is similar to the result without the false constraint from table 8.4. Without the switchable loop constraint, the iSAM2 optimizer crashed in the form of throwing and indeterminant linear system exception. As I added the constraint, the estimated trajectory started oscillating approximately 1 meter side to side, because the switch value corresponding to the false loop constraint was switched on and off by the optimizer, as can be seen in figure 8.21. I found that the value of the switch prior variance $\Xi_{300,0}$ explained in section 6.4, which was set to 1.0 in the listed result, had an impact on this oscillatory behaviour. I found that with a smaller value of $\Xi_{300,0} = 0.5$, the oscillation was much more significant and lasted throughout the sequence, and with a larger value $\Xi_{300,0} = 2.0$, more of the true positive loop constraints was rejected.

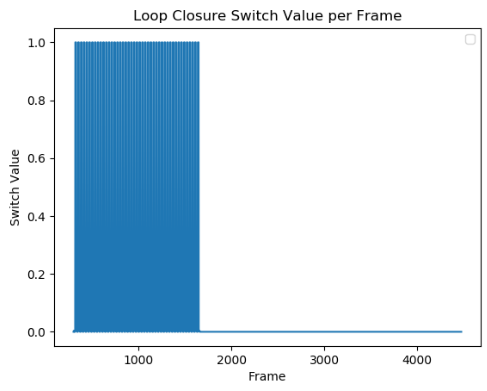


Figure 8.21: Value of the switch corresponding to the false loop constraint applied between frame 300 and the origin. The switch starts to exist at frame 300, and its value oscillates between 0 and 1 until a loop closure occurs.

Adding false loop constraints becomes more and more dangerous as the robot explores without receiving GNSS information or a correct loop closure. This is because the uncertainty grows larger and larger when performing pure visual odometry, and so the false positive loop closure becomes more and more likely to the optimizer.

The system was also able to handle false positive loop constraints added every

300th frame (15 total), and in this case the estimate achieved an $\text{ATE}_{\text{pos}} = 5.57$ meters and $\text{ATE}_{\text{rot}} = 2.91$ degrees. However, the estimate was very oscillatory as the switch values for many of the false constraints did not converge.

The experiment from section 8.5 was also performed without the switchable constraint. It yielded approximately the same performance with $\text{ATE}_{\text{pos}} = 3.14$ meters and $\text{ATE}_{\text{rot}} = 1.66$ degrees, but the optimization time was reduced, as can be seen in figure 8.22. Compared to figure 8.16, there are less oscillations in the execution time after the second sequence of loop closures. This indicates that some of the loop constraints added during normal operation are inaccurate, leading to oscillations in the switch values and thus more re-linearization of variables during optimization. This is also indicated by figure 8.23, where some of the true positive loop detections result in constraints that are fully or partially optimized out.

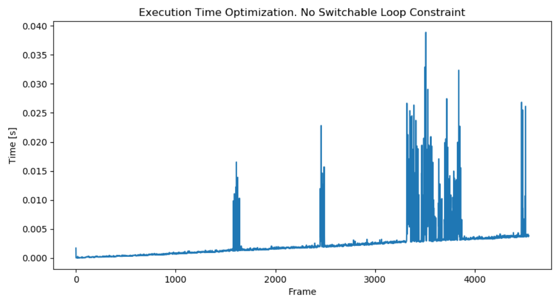


Figure 8.22: Execution time of optimization without the switchable loop constraint. Mean is 2ms, max is 39ms.

8.8 Switchable GNSS Constraint

To test the switchable GNSS constraints explained in section 6.5, I performed two experiments. In the first experiment, I applied GNSS measurements to every 10th keyframe, where every 10th measurement was corrupted with a fixed additive noise of 30 meters in a random horizontal direction to simulate multipath artifacts. The noise

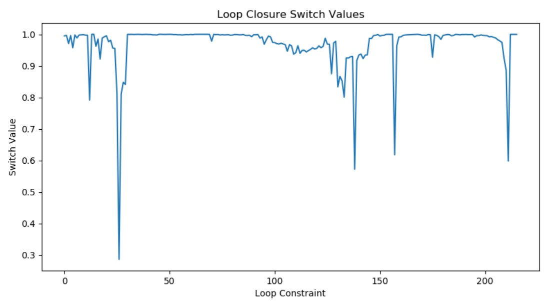
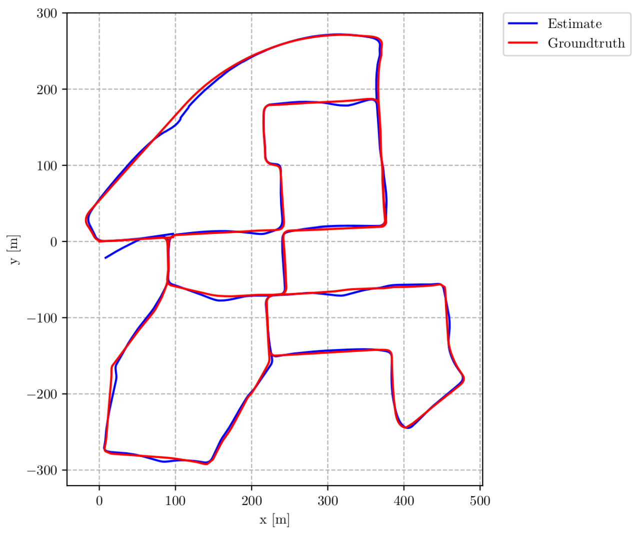


Figure 8.23: Switch values of all switchable loop constraints added when performing V-SLAM without GNSS. Some constraints are fully or partially optimized out even though there are no false positive loop detections.

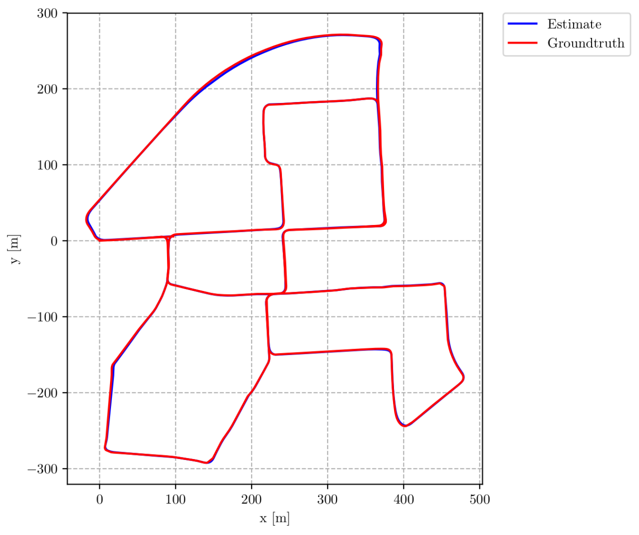
model on the GNSS measurement was a diagonal Gaussian with standard deviation of $\sigma = 5$ meters in every direction, so the added noise was not conforming to the noise model. The switchable factor was compared to the original GNSS factor explained in 6.3.2, which is implemented in the `GPSFactor()` class in GTSAM. The result of using the original GNSS factor can be seen in figure 8.24a, where it is clear that the trajectory estimate is affected by the simulated multipath artifacts. Notice how the first pose near the origin in figure 8.24a, which is affected the noise, leads the switchable loop constraint to discard the detected loop.

Figure 8.24b shows the same experiment using the switchable factor. The estimate achieves an ATE of $\text{ATE}_{\text{pos}} = 1.35$ meters and $\text{ATE}_{\text{rot}} = 1.47$ degrees, which is very similar, but slightly worse, compared to the results from the experiment with unperturbed GNSS measurements in table 8.5. Figure 8.25 shows the switch values of every switchable GNSS factor added to the graph, which indicates how the optimizer considers the probability of each measurement being valid. We can see that most of the corrupted measurements are optimized out of the graph topology, but some still remains (e.g. measurement 50), which explains the slightly worse performance compared to the experiment with unperturbed measurements.

In the second experiment, I applied GNSS measurements on every keyframe, and



(a) Without the switchable GNSS factor.



(b) With the switchable GNSS factor.

Figure 8.24: Trajectories aligned with the ground truth from the experiment with GNSS measurements corrupted by simulated multipath artifacts. GNSS measurements are provided every 10^{th} keyframe, and every 10^{th} measurement is corrupted with a fixed 30m additive noise in a random horizontal direction.

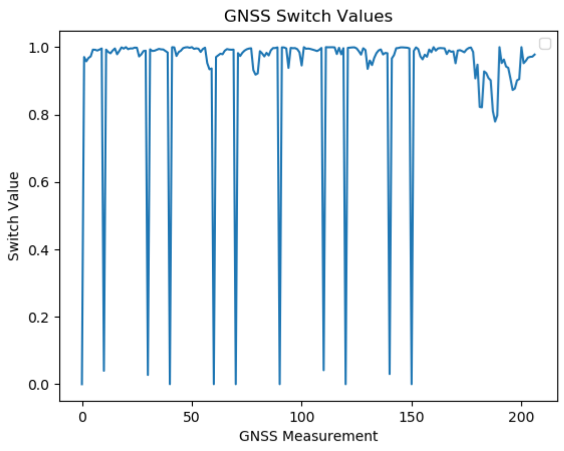


Figure 8.25: Switch values related to the switchable GNSS constraints from the multi-path experiment.

added a diagonal Gaussian noise to every measurement with a standard deviation of $\sigma = 10$ meters in every direction. The noise model on the measurements was also a diagonal Gaussian, but with a standard deviation of $\sigma = 5$ meters in every direction, so the measurements with added noise was not conforming to the noise model. The result from using the original `GPSFactor()` can be seen in figure 8.26a. It is clear that the estimate is severely affected, as the optimizer puts too much confidence in the GNSS measurements. Replacing the GNSS factor with the switchable GNSS factor in figure 8.26b, we can see that the estimate becomes much smoother. Here, many of the measurements are optimized out of the graph topology, as indicated by the switch values in figure 8.27 which has a mean of $\mu = 0.50$ and a standard deviation of $\sigma = 0.40$. The estimate achieves $\text{ATE}_{\text{pos}} = 2.96$ meters and $\text{ATE}_{\text{rot}} = 5.11$ degrees, which is not as good as the result from table 8.6 (estimate with the correct noise model). This is because, as we saw from figure 8.25, not all measurements that falls outside the noise model are optimized out of the topology, and the optimizer puts to much confidence in those remaining measurements. The execution time with and without the switchable

GNSS constraint was very similar to the results in table 8.6, with a mean of 30ms, and a max of 141ms without the switch, and 147ms with.

8.9 Re-localization

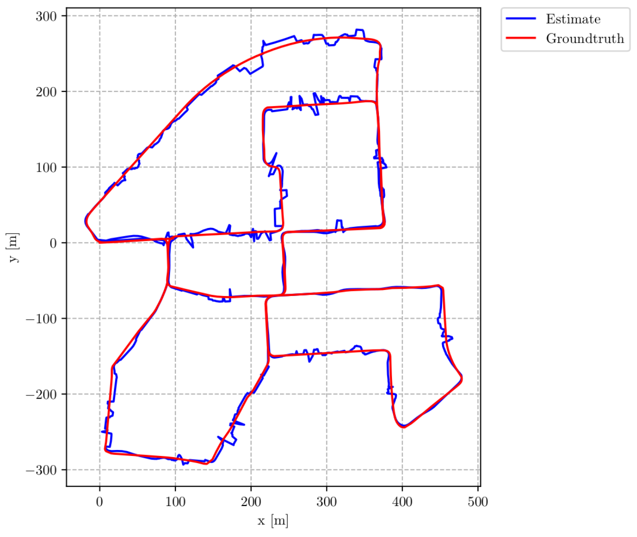
To test the system's ability to recover after a tracking failure, I performed three experiments. One where tracking failed at frame 500 without GNSS, and two experiments where tracking failed every 1000th frame (resulting in 4 tracking failures during the 4541 frame long sequence) with and without GNSS. To simulate tracking failures, images loaded from the dataset was filled with zeros (black).

8.9.1 Without GNSS

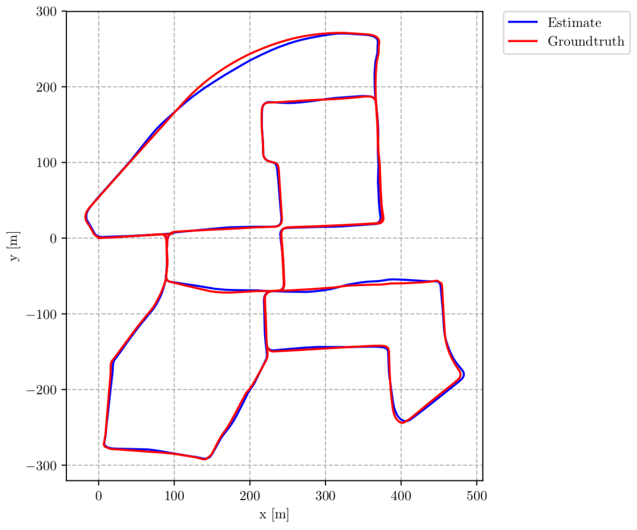
When there are no GNSS measurements, the trajectory is recovered when encountering a loop closure. As explained in section 7.7, there is no constraint inserted at the position where track was lost after recovery. The tracking failure at frame 500 was placed strategically as the robot passes this section twice, and the disconnected point is constrained with loop constraints to the second pass. The resulting trajectory can be seen in figure 8.28, together with the absolute translation and and rotation error in figure 8.29a and 8.29b, respectively. Referring to figure 8.28, the tracking failure happens at approximately $(x, y) = (240, -20)$, but there is no visible signs of the tracking failure at this point, or in the plots in 8.29a, 8.29b. In fact, comparing table 8.7 with table 8.4, we that the ATE in translation and rotation is lower with the tracking failure than without.

		Translation		Rotation		Execution Time	
Tracking Failures	GNSS	ATE_{pos} [m]	σ_{pos} [m]	ATE_{rot} [rad]	σ_{rot} [rad]	Mean [ms]	Max [ms]
1	No	2.70	1.28	1.59	0.69	27	107
4	No	9.44	5.98	3.03	1.11	26	114
4	Yes	1.25	0.50	1.43	0.81	26	100

Table 8.7: Absolute translation and rotation error statistics tracking failure experiments.



(a) Without the switchable GNSS factor.



(b) With the switchable GNSS factor.

Figure 8.26: Trajectories aligned with the ground truth from experiment with nonconforming GNSS noise. GNSS measurements perturbed by a diagonal Gaussian noise with $\sigma = 10$ meters are applied to every keyframe, while the measurement model is a diagonal Gaussian with $\sigma = 5$ meters.

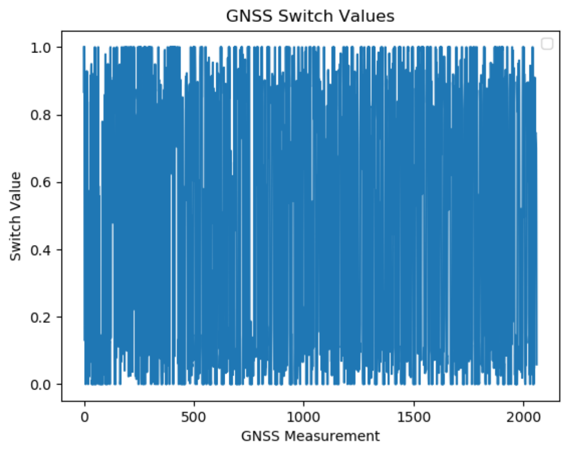


Figure 8.27: Switch values related to the switchable GNSS constraints from the experiment with nonconforming GNSS noise.

In figure 8.30, we see the resulting trajectory when tracking fails at frame 1000, 2000, 3000 and 4000, respectively corresponding to the (approximate) (x, y) coordinates $(340, 190)$, $(20, -280)$, $(390, -240)$ and $(360, 270)$ in the figure. Now, all the disconnected points are clearly visible, as none of these points are passed twice. The plots 8.31a and 8.31b shows how the translation and rotation error is affected. As there are no constraints on the poses following the tracking failures, the trajectory is very sensitive to errors in the loop closure constraints when recovering.

8.9.2 With GNSS

Adding GNSS measurements every 10th keyframe to the last experiment where tracking fails every 1000th frame, the results are drastically improved as can be seen in figure 8.32. Looking at figure 8.33a and 8.33b, we can still see the effects of the tracking failures, especially in translation, but the amplitude is much smaller compared to figure 8.31a and 8.31b.

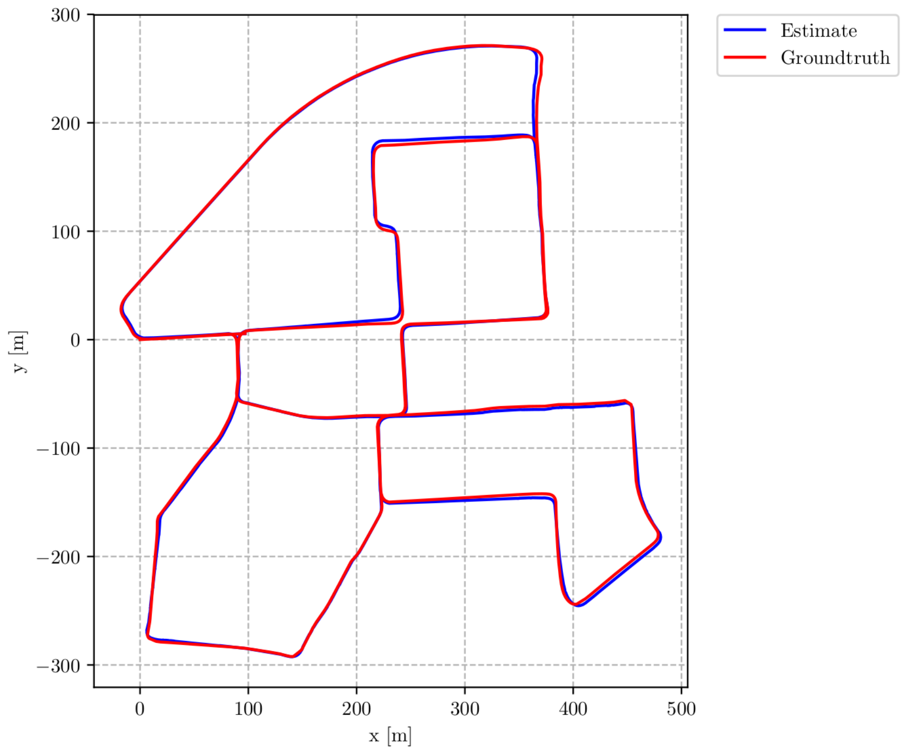
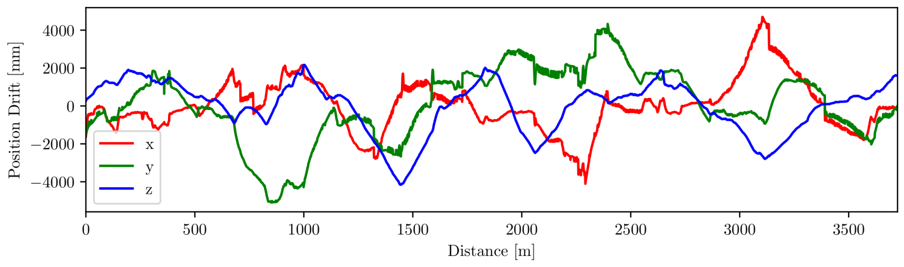
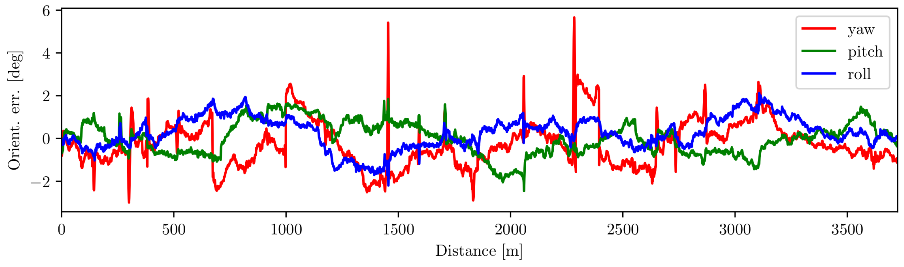


Figure 8.28: Trajectory estimate aligned with ground truth from experiment where a tracking failure occurs at frame 500, corresponding to coordinate $(x, y) = (240, -20)$.



(a) Translation error.



(b) Rotation error.

Figure 8.29: Translation and rotation error from experiment with a tracking failure at frame 500.

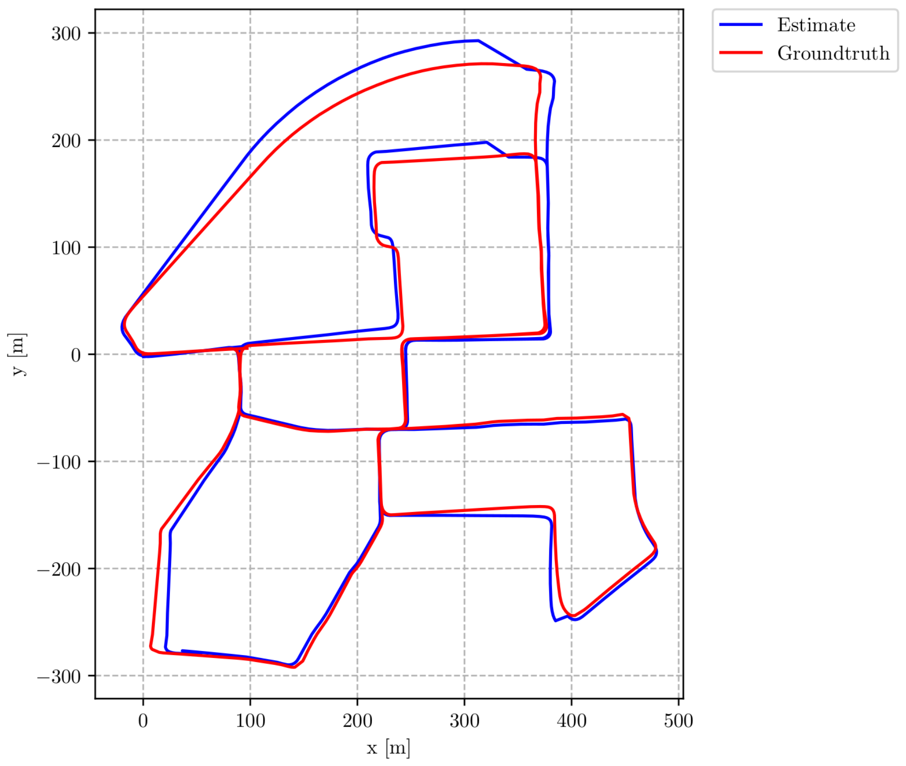
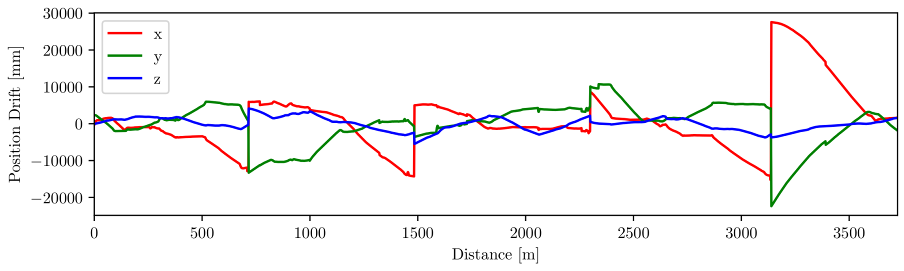
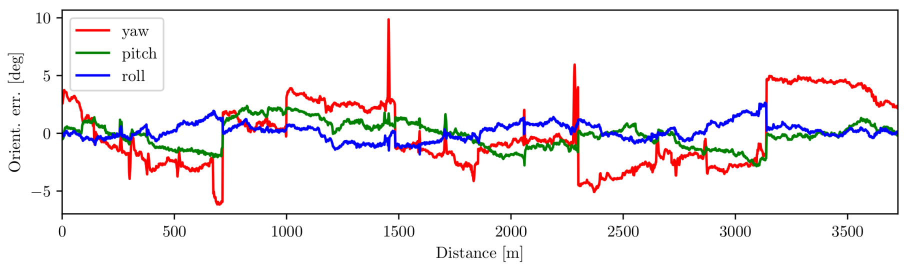


Figure 8.30: Trajectory aligned with ground truth from experiment where tracking failures occure every 1000^{th} frame, without GNSS measurements.



(a) Translation error.



(b) Rotation error.

Figure 8.31: Translation and rotation error from experiment where tracking failures occur every 1000th frame, without GNSS measurements.

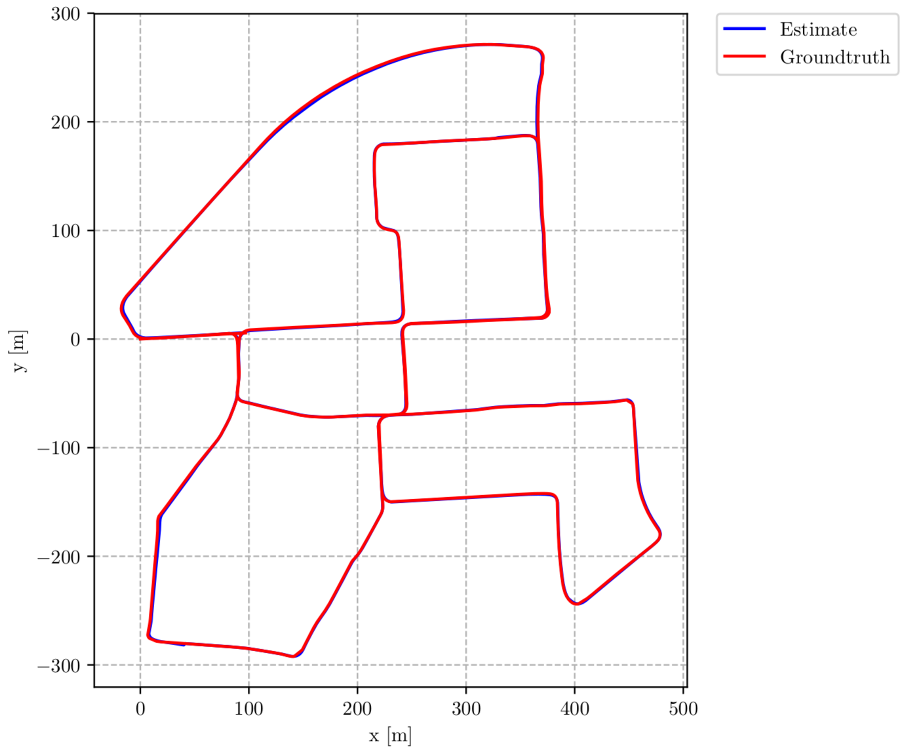
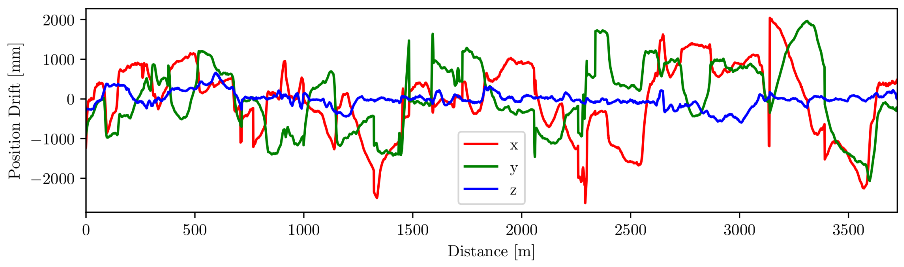
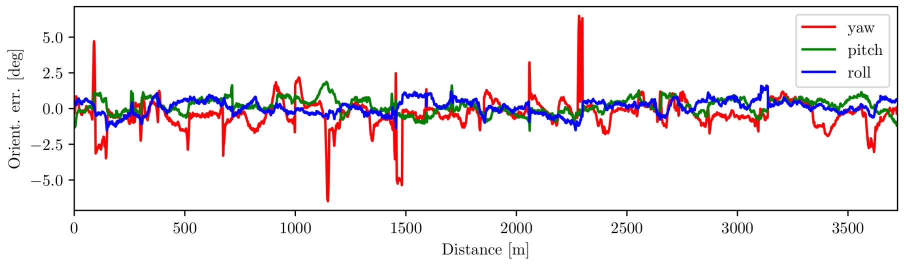


Figure 8.32: Trajectory estimate aligned with the ground truth from experiment where tracking fails every 1000th frame, with GNSS measurements every 10th keyframe.



(a) Translation error.



(b) Rotation error.

Figure 8.33: Translation and rotation error from experiment where tracking failures occurs every 1000th frame with GNSS measurements every 10th frame.

Chapter 9

Conclusions and Future Work

In this project, a GPU-accelerated stereo V-SLAM system was developed, capable of utilizing GNSS sensor data to correct for accumulated drift, and re-localize after tracking failure without discarding information gathered from previously explored areas.

The performance of the VO module of the developed system was able to provide trajectory estimates which in most cases exceeded the performance of the LIBVISO2 algorithm on the KITTI dataset, while operating at a much higher frame-rate of 47.6 fps. By including the DBOW2 loop detection module, the accuracy of the estimates was further improved on all tested datasets which contained loop events, however this also impacted the execution time of the system which had a worst case of 115 ms on a sequence of 4541 frames. I still claim that the system is real-time capable, since the mean execution time was 28 ms which corresponds to a frame rate of 35.7 fps.

The system was able to recover from false positive loop detections, as demonstrated in section 8.7, by using the switchable constraint as proposed in [66]. This improved the robustness of the system compared to using the traditional loop constraint implemented in the GTSAM library, which caused the system to crash in the performed experiment. Using the switchable constraint came with a penalty of slightly higher execution times in the optimization due to oscillations in the switch values, which may be due to inaccuracies in the computed loop constraints.

When provided GNSS measurements, the system was able to increase the precision of the trajectory estimates further. The system was able to handle limited GNSS coverage by estimating the trajectory relative to a locally defined reference frame until enough GNSS measurements was received, and then initialize the trajectory in the global reference frame. The system was able to provide accurate estimates with as little as 5 measurements on the 3724 meter long KITTI 00 sequence, as shown in section 8.6. As shown in section 8.8, the system proved to be robust to GNSS measurements corrupted by simulated multipath artifacts and measurements not conforming to the expected noise model, due to the switchable GNSS factor presented in section 6.5.

The system was also able to recover from simulated tracking failures both with and without GNSS measurements, as shown in section 8.9. Testing showed that the system was able to refine the estimate at the point where tracking was lost by performing a second pass over that section. If the points of failure was not passed a second time, the estimate became less accurate in the case when there were no GNSS measurements.

Unfortunately, I was not able to test the developed system properly on any datasets from Olav due to problems with the stereo calibration. As suggested by my co-supervisor at FFI, a possible contributing factor to the calibration failure is that the stereo cameras may not have been attached rigidly enough. A suggestion on how to possibly improve the calibration procedure is presented in the next section.

9.1 Future Work

These are my thoughts on what remains to be done in order to incorporate cameras into the navigation system on Olav.

- Stereo calibration. In order to utilize stereo cameras for navigation, correct stereo calibration is a necessity. One interesting stereo calibration technique particularly intended for cameras mounted on automobiles is the Camera and Range Sensor Calibration Toolbox by Geiger et al. [26] which calibrates the stereo pair from a single stereo frame. This method requires a set of calibration patterns arranged in different poses such that they cover the whole field of view of both cameras. The single stereo frame can then be uploaded to a web-page [27] where the calibration parameters are computed.

- As shown in the results, the execution time sometimes exceeded the real-time constraint when encountering loop closures. A possible solution to this is to let the optimization module operate continuously in parallel.
- Improve the visual odometry module. The VO module presented in section 5.3 is a frame-to-frame algorithm. Its performance could probably be enhanced by performing windowed bundle adjustment as is done in ORB-SLAM2 or DSO. This way, better relative pose constraints could be estimated by optimization over mutual observations of features over multiple frames instead of consecutive frames only. This would require some extra computation, but only over a limited window. Since the VO system already tracks features over multiple frames, implementing this would require a modification in the optimization module only. This would also a better strategy for inserting keyframes into the pose-graph than just considering the optical flow between images, as is presently done.
- Better estimation of the noise models of the relative pose constraints. In its current state, the noise model, or covariance matrices, of the relative pose constraints used in the optimization is proportional to the cumulative optical flow computed between frames. By using a windowed BA approach, the noise models could be better estimated by only specifying the pixel uncertainty, and let that uncertainty propagate to the relative pose constraints estimated by a windowed BA procedure. This point also applies to the loop constraints, which I suspect would lead to less oscillations in the switch values of the switchable loop constraints and thus lower execution times.
- The performance of the VO algorithm in dense forest and terrain environments remains to be tested.
- Develop a ROS wrapper around the current implementation. Olav uses ROS to pass messages between nodes in its software framework. For the system to be

operational on Olav, the system should be embedded in a ROS framework. As I was not able to obtain a good calibration on the dataset from Olav, and since the KITTI dataset is not dependent on ROS, this was not a priority. However, the system was developed with ROS in mind, as the only input to the system is time-stamped images and GNSS measurements.

- Find loop constraints from stored features instead of stored images. For long term operation, storage of all keyframes in raw format can become expensive.
- Add more sensor measurements as constraints in the optimization. For the navigation system to be complete, IMU measurements should be added as odometry constraints in the factor graph. One interesting approach on how to do this is the method of Forster et al. [19] which summarizes the high frequency inertial measurements from the IMU into a single relative motion constraint (factor) between frames. Thus, instead of adding variable and factor nodes to the factor graph at IMU rate, they are added at a much slower frequency that is determined by other available sensors. Also, Olav is equipped with wheel encoders which can be added as factors as well.
- As discussed in section 5.4, feature-based place recognition is known to lack performance over day/night and seasonal changes in the environment, although this was not part of the test sequences in this project. It would be interesting to look into sequence based place recognition, as those methods have proven able to do place recognition over such changes. By using the DBOW2 as place recognition module, the database may have to be augmented with seasonal and daily variation for each location.
- The DBOW2 module also inherently has the problem that the database structure grows over explored area, and if the size of the explored area is not known before starting the mission, this can cause problems with execution time and memory overflow.
- In the experiments performed in chapter 8, I assumed that the availability of GNSS measurements coincided with the keyframes. This assumption is a simplifi-

cation, and could be changed for example by basing the keyframe decision on the availability of GNSS measurements, or by transforming the GNSS measurements on non-keyframes back to the keyframes, which would have to happen after the estimate is initialized in the global frame. The last option would also work when measurements are available between frames, by assuming an intermediate pose between frames at the time of measurement based on velocity estimates. The GNSS measurement on that intermediate pose could then be transformed back to the keyframe based on the transform between the intermediate pose estimate and the corresponding keyframe.

- As the variance of the switch prior in the switchable constraints impacts their performance, the front-end could potentially inform the back-end about the probability of a loop closure being correct as suggested in [66]. As mentioned in section 8.7, adding a false positive loop constraint becomes more dangerous after long periods of exploration, and the value of the switch prior variance may help mitigate this.

- For very large scale operation, nodes of the pose graph corresponding to poses far away from the current position should not be stored in working memory, as this would eventually overload the system. A solution, such as the one presented by Labbé et al. [41], where distant sections of the map is stored in long term memory should be investigated. Some of the foundations for doing this, such as save and load functionality of the BoW database and factor graph, is present in the current implementation, but was not extensively tested. The implemented re-localization in the developed system is also an important building block which can be extended for this purpose.

References

[1] D. Arthur and S. Vassilvitskii. K-means++: the advantages of careful seeding. In *In Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[2] T. Bailey and H. Durrant-Whyte. Simultaneous localisation and mapping (slam): Part ii state of the art. *IEEE Robotics Automation Magazine*, 13, 01 2006.

[3] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, Inc., 2nd edition, 2013.

[4] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard. Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

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

[6] M. Cummins and P. Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *Int. J. Rob. Res.*, 27(6):647–665, June 2008.

[7] M. Cummins and P. Newman. Appearance-only slam at large scale with fab-map 2.0. *Int. J. Rob. Res.*, 30(9):1100–1123, Aug. 2011.

[8] I. Cvisic and I. Petrovic. Stereo odometry based on careful feature selection and tracking. *2015 European Conference on Mobile Robots (ECMR)*, pages 1–6, 2015.

[9] I. Cvišić, J. Ćesić, I. Marković, and I. Petrovic. Soft-slam: Computationally efficient stereo visual simultaneous localization and mapping for autonomous unmanned aerial vehicles. *Journal of Field Robotics*, 35, 11 2017.

[10] F. Dellaert. Factor graphs and gtsam: A hands-on introduction. Technical Report MSU-CSE-06-2, Center for Robotics and Intelligent Machines, Georgia Institute of Technology, September 2012. Collection: Computational Perception and Robotics Technical Reports.

[11] F. Dellaert and M. Kaess. Square root sam: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Research*, 25(12):1181–1203, Dec. 2006.

[12] F. Dellaert and M. Kaess. Factor graphs for robot perception. *Foundations and Trends in Robotics*, 6(1-2):1–139, August 2017.

[13] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, June 2006.

[14] J. Engel, V. Koltun, and D. Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP(99):1–1, 2017.

[15] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision (ECCV)*, September 2014.

[16] J. Engel, J. Stückler, and D. Cremers. Large-scale direct slam with stereo cameras. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1935–1942, Sept 2015.

[17] F. D. et al. Georgia tech smoothing and mapping library. `https://bitbucket.org/gtborg/gtsam`, 2018. Accessed: 2018-06-10.

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

[19] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. On-manifold preintegration theory for fast and accurate visual-inertial navigation. *CoRR*, abs/1512.02363, 2015.

[20] F. Fraundorfer and D. Scaramuzza. Visual odometry : Part ii: Matching, robustness, optimization, and applications. *IEEE Robotics and Automation Magazine*, 19(2):78–90, 2012.

[21] P. Furgale, J. Maye, J. Rehder, T. Schneider, and L. Oth. Kalibr multiple camera calibration. `https://github.com/ethz-asl/kalibr`. Accessed: 2017-12-31.

[22] D. Gálvez-López and J. D. Tardós. Dbow2: Enhanced hierarchical bag-of-word library for c++. `https://github.com/dorian3d/DBoW2`. Accessed 2018-06-12.

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

[24] D. Galvez-Lopez and J. D. Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, October 2012.

[25] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *Int. J. Rob. Res.*, 32(11):1231–1237, Sept. 2013.

[26] A. Geiger, F. Moosmann, Ömer Car, and B. Schuster. Automatic camera and range sensor calibration using a single shot. In *International Conference on Robotics and Automation (ICRA)*, St. Paul, USA, May 2012.

[27] A. Geiger, F. Moosmann, Ömer Car, B. Schuster, and C. Stiller. Camera and range sensor calibration toolbox. `http://www.cvlibs.net/software/calibration/`, 2018. Accessed: 2018-12-28.

[28] A. Geiger, J. Ziegler, and C. Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *Intelligent Vehicles Symposium (IV)*, 2011.

[29] T. Haavardsholm, I. Dyrdal, and T. Opsahl. Unik4690, maskinsyn (computer vision). `http://www.uio.no/studier/emner/matnat/its/UNIK4690/v17/index.html`, 2017. Accessed: 2017-01-15.

[30] T. V. Haavardsholm, R. Smestad, M. V. Larsen, M. Thoresen, and I. Dyrdal. Scene understanding for autonomous steering. In *STO-MP-IST-127: Intelligence and Autonomy In Robotics*, 127. NATO Science and Technology Organization, 2016. Paper 4.

[31] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Fox, and N. Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *In Proc. of the Intl. Sym. of Robot. Research*, 2011.

[32] A. S. Huang, A. Bachrach, and D. Maturana. libfovis. `https://github.com/fovis/fovis`, 2018. Accessed: 2018-06-10.

[33] V. Indelman, S. Williams, M. Kaess, and F. Dellaert. Factor graph based incremental smoothing in inertial navigation systems. In *International Conference on Information Fusion, FUSION*, pages 2154–2161, Singapore, July 2012.

[34] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *International Journal of Robotics Research*, 31(2):217–236, February 2012.

[35] M. Kaess, A. Ranganathan, and F. Dellaert. isam: Incremental smoothing and mapping. *Trans. Rob.*, 24(6):1365–1378, Dec. 2008.

[36] M. Kaess, S. Williams, V. Indelman, R. Roberts, J. J. Leonard, and F. Dellaert. Concurrent filtering and smoothing. In *2012 15th International Conference on Information Fusion*, pages 1300–1307, July 2012.

[37] C. F. F. Karney. Geographiclib, version 1.49. `https://geographiclib.sourceforge.io/1.49`. Accessed 2018-06-12.

[38] B. Kitt, A. Geiger, H. Lategahn, J. Ziegler, and C. Stiller. `http://www.cvlibs.net/software/libviso/`. Accessed 2018-06-12.

[39] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. G2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613, May 2011.

[40] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o - general graph optimization. https://github.com/RainerKuemmerle/g2o, 2018. Accessed: 2018-06-10.

[41] M. Labbé and F. Michaud. Long-term online multi-session graph-based splam with memory management. *Auton. Robots*, 42(6):1133–1150, Aug. 2018.

[42] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, Sept. 1981.

[43] C. T. Loop and Z. Zhang. Computing rectifying homographies for stereo vision. In *CVPR*, pages 1125–1131. IEEE Computer Society, 1999.

[44] S. Lovegrove. Pangolin - a rapid development library for managing opengl display. `https://github.com/stevenlovegrove/Pangolin`, 2018. Accessed: 2018-06-10.

[45] S. Manthe, A. Carrio, F. Neuhaus, P. Campoy, and D. Paulus. Combining 2d to 2d and 3d to 2d point correspondences for stereo visual odometry. In *VISIGRAPP (5: VISAPP)*, pages 455–463. SciTePress, 2018.

[46] K. Mathiassen, M. Baksaas, L. E. Olsen, M. Thoresen, and B. Tveit. Development of an autonomous off-road vehicle for surveillance missions. In *STO-MP-IST-127: Intelligence and Autonomy In Robotics*. NATO Science and Technology Organization, 2016. Paper 5.

[47] J. Maye, P. T. Furgale, and R. Siegwart. Self-supervised calibration for robotic systems. In *Intelligent Vehicles Symposium*, pages 473–480. IEEE, 2013.

[48] M. J. Milford and G. F. Wyeth. Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. In *2012 IEEE International Conference on Robotics and Automation*, pages 1643–1649, May 2012.

[49] H. P. Moravec. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover.* PhD thesis, Stanford University, Stanford, CA, USA, 1980.

[50] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. Orb-slam2 open-source code. `https://github.com/raulmur/ORB_SLAM2`. Accessed: 2017-12-28.

[51] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, Oct 2017.

[52] R. Mur-Artal and J. D. Tardós. Fast relocalisation and loop closing in keyframe-based slam. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 846–853, May 2014.

[53] D. Nister. An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–770, June 2004.

[54] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'04)*, pages 652–659, 2004.

[55] E. Olson. Apriltag: A robust and flexible multi-purpose fiducial system. Technical report, University of Michigan APRIL Laboratory, May 2010.

[56] T. Pire, T. Fischer, G. Castro, P. De Cristóforis, J. Civera, and J. Jacobo Berlles. S-ptam: Stereo parallel tracking and mapping. *Robotics and Autonomous Systems*, 93, 04 2017.

[57] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[58] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *Proceedings of the Tenth IEEE International Conference on Computer Vision - Volume 2*, ICCV '05, pages 1508–1515, Washington, DC, USA, 2005. IEEE Computer Society.

[59] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*, ECCV'06, pages 430–443, Berlin, Heidelberg, 2006. Springer-Verlag.

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

[61] D. Scaramuzza and F. Fraundorfer. Visual odometry: Part i - the first 30 years and fundamentals. *IEEE Robotics and Automation Magazine*, 18(4):80–92, 2011.

[62] H. Strasdat, J. M. M. Montiel, and A. J. Davison. Real-time monocular slam: Why filter? In *ICRA*, pages 2657–2664. IEEE, 2010.

[63] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar. Robust stereo visual inertial odometry for fast autonomous flight. *CoRR*, abs/1712.00036, 2017.

[64] N. Sünderhauf, M. Obst, S. Lange, G. Wanielik, and P. Protzel. Switchable constraints and incremental smoothing for online mitigation of non-line-of-sight and multipath effects. In *Intelligent Vehicles Symposium*, pages 262–268. IEEE, 2013.

[65] N. Sünderhauf and P. Protzel. Vertigo: Versatile extensions for robust inference using graph optimization. `https://openslam-org.github.io/vertigo.html`. Accessed 2018-06-12.

[66] N. Sünderhauf and P. Protzel. Switchable constraints for robust pose graph slam. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1879–1884, Oct 2012.

[67] B. Vik. Integrated satellite and inertial navigation systems. Ttk5 lecture notes, Engineering Cybernetics Department, NTNU, 2014.

[68] R. Wang, M. Schwörer, and D. Cremers. Stereo DSO: large-scale direct sparse visual odometry with stereo cameras. *International Conference on Computer Vision (ICCV)*, 2017.

[69] R. Watson and J. Gross. Robust navigation in gnss degraded environment using graph optimization. *CoRR*, abs/1806.08899, 2018.

[70] R. M. Watson and J. N. Gross. Robust gnss processing with factor graphs. `https://github.com/wvu-navLab/RobustGNSS`. Accessed 2019-01-15.

[71] J. Weng, P. Cohen, and M. Herniou. Camera calibration with distortion models and accuracy evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(10):965–980, Oct. 1992.

[72] B. Williams, M. Cummins, J. Neira, P. Newman, I. Reid, and J. Tardos. A comparison of loop closing techniques in monocular slam. *Robotics and Autonomous Systems*, 57:1188–1197, 12 2009.

[73] J. Wu, D. Yang, Q. Yan, and S. Li. Stereo-dso open-source code. `https://github.com/HorizonAD/stereo_dso`, 2017. Accessed: 2017-01-15.

[74] J. yves Bouguet. Pyramidal implementation of the lucas kanade feature tracker. *Intel Corporation, Microprocessor Research Labs*, 2000.

[75] Z. Zhang, J. Huber, J. Delmerico, and C. Forster. rpg-trajectory-evaluation - toolbox for quantitative trajectory evaluation of vo/vio. `https://github.com/uzh-rpg/rpg_trajectory_evaluation`, 2018. Accessed: 2018-12-10.

[76] Z. Zhang and D. Scaramuzza. A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry. In *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2018.

Fredrik Bjerkås

NTNU
Norwegian University of
Science and Technology

FFI Forsvarets
forskningsinstitutt