Master's thesis

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

Thomas Schiøler Hellum

Multi-Sensor Stereoscopic Visual SLAM for Autonomous Automotive and Seaborne Vehicles

A Modular Approach to Concurrent Sensor Fusion of Multi-Frame Feature Based Stereo VSLAM, IMU and GNSS using Factor Graph Optimization

Master's thesis in Cybernetics and Robotics
Supervisor: Edmund Førland Brekke
Co-supervisor: Rudolf Mester
July 2021

NTNU
Norwegian University of Science and Technology

Thomas Schiøler Hellum

Multi-Sensor Stereoscopic Visual SLAM for Autonomous Automotive and Seaborne Vehicles

A Modular Approach to Concurrent Sensor Fusion of Multi-Frame Feature Based Stereo VSLAM, IMU and GNSS using Factor Graph Optimization

Master’s thesis in Cybernetics and Robotics
Supervisor: Edmund Førland Brekke
Co-supervisor: Rudolf Mester
July 2021

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

Dedicated to my parents for all their support.

Preface

This report is the final product of the master's thesis course TTK4900 at the Norwegian University of Science and Technology (NTNU). It is written as part of the Autoferry (Autonomous all-electric passenger ferries for urban water transport) project, which is an associated project of the NTNU Centre for Autonomous Marine Operations and Systems. The thesis is written during the spring of 2021 and was carried out at the Department of Engineering Cybernetics and is submitted in partial fulfillment of the requirements for the degree in Master of Technology in Navigation, Vessel Control Systems and Robot Engineering. The thesis serves as a continuation of a specialisation project conducted during the fall of 2020 and is contributing to the field of image based motion estimation and sensor fusion.

I would like to extend my profound gratitude towards my supervisors. My main supervisor, Associate Professor Edmund Førland Brekke has been a terrific support through his deep knowledge within the topic of sensor fusion, and his conscientious follow-up and guidance has been essential for the progression of this thesis. My co-supervisor Professor Rudolf Mester, has been my main pillar of support regarding the subject matter of this thesis. His theoretical expertise within the field of SLAM and computer vision, coupled with his patience with every question and his interest as to my progression have played a key part in making this a great learning experience.

In addition, I would like to thank Martin Eek Gerhardsen, Martin Græsdal and Kristian Auestad for the collaboration preceding, during and succeeding the data logging performed at Brattøra with the autonomous ferry prototype, milliAmpere. Finally, I would like to extend a special thanks to PhD Candidate Andreas Langeland Teigen and, again, to Martin Eek Gerhardsen for discussions and feedback on both theoretical and practical matters throughout the thesis.

[illegible]

Thomas Schiøler Hellum
Trondheim, July 5, 2021

Abstract

For this project, a real-time capable GPU-accelerated feature-based stereo Visual Simultanous Localization And Mapping (VSLAM) solution, capable of fusing measurements from Inertial Measurement Unit (IMU) and Global Navigation Satellite System (GNSS), is developed. Initial motion estimates are produced in one thread while concurrently managing 3D points and performing multi-frame Bundle Adjustment (BA) over short-term windows. Place recognition is successfully performed without false positives, and loop closures are carried out in a final, third thread. The concurrent long-term and short-term optimization is solved over a single factor graph, where iSAM2 is used for the underlying update rule. Preintegrated IMU and GNSS measurements are fused with the short-term VSLAM estimates by optimization. GNSS data are thus available to correct for drift, while the visual-inertial Simultanous Localization And Mapping (SLAM) module provides accurate motion estimates during temporary or permanent loss of GNSS data. This enables the vehicle to report accurate trajectory estimates relative to a global reference frame.

The developed system is validated on real world sensor data recorded on-board the autonomous ferry prototype milliAmpere (mA). However, as a result of insufficient stereo camera calibration, the main portion of the testing is rather performed on the publicly available Karlsruhe Institute of Technology and Toyota Technological Institute (KITTI) dataset. Analysis showed that the Visual-Inertial Odometry (VIO) part of the system outperforms the popular stereo VO system LIBVISO2 on most tested sequences. When loop closures are extended to the VIO module, the gap in performance is increased even further. The resulting performance is also measured up against and compared with one of the current state-of-the-art solutions, ORB-SLAM2, to put the performance of the developed system in perspective.

Sammendrag

I denne oppgaven er det utviklet et navigasjonssystem basert på visuell simultan lokalisering og kartlegging (VSLAM), som er i stand til å inkludere målinger fra IMU og GNSS i en samlet factor graf sensor fusjon. VSLAM systemet er feature-basert og optimalisert for prossessering på GPU. Initielle bevegelsesestimater produseres mens man parallelt utfører multi-frame BA over vinduer med aktive factorer. Stedsgjenkjenning gjennomføres vellykket uten noen falske positive deteksjoner, der lukking av den detekterte sløyfen utføres i en siste, tredje tråd. Et filter og en smoother gjennomfører optimaliseringer parallelt over en samlet factor graf, der iSAM2 brukes som den oppdateringsstrategi når nye målinger legges til. Preintegrerte IMU- og GNSS-målinger fusjoneres sammen med VSLAM estimater i filteret. På den måten vil globale GNSS-data korrigere for drift når de er tilgjengelige, mens den visuell-inertiale SLAM-modulen gir nøyaktige målinger av forflytningen til kjøretøyet i mellomtiden, eller hvis signalet til GNSSen faller ut. På denne måten kan kjøretøyet motta nøyaktige målinger på sin globale posisjon, selv ved lavere oppdateringsrate fra GNSSen.

Det utviklede systemet er validert på innsamlede sensordata fra den autonome ferjeprototypen milliAmpere. Som et resultat av problemer med kamerakalibreringen ble testingen hovedsakelig gjennomført på det offentlig tilgjengelige KITTI-datasettet. Analyse viste at VIO-delen av systemet overgår det populære stereo VO-algoritmen LIBVISO2 på de fleste av de testede sekvensene. Når lukking av sløyfer ble lagt til VIO-modulen, økes gapet i ytelse ytterligere. Presisjonen til den utviklede algoritmen sammenlignes også med en av de nåværende beste SLAM algoritmene, ORB-SLAM2, for å understreke svakheter som utløses i noen spesielle tilfeller.

Contents

CHAPTER 1

Introduction

This introductory chapter serves to give the reader a better understanding of the framework which the thesis is built upon. First, the background and motivation behind the thesis work is presented, before targeted contributions are specified. Related work is then elaborated, followed by the thesis' relation to the specialisation project. Lastly, the structure of the report is outlined.

1.1 Background and Motivation

Autonomous robots have seen considerable advancement in recent years due to an escalating degree of inter-working, increasingly complex system solutions. There are different degrees of autonomy, but an autonomous system should in general be capable of operating with little to no human interaction. A key component of autonomous operations is the ability to accurately perceive and understand both static and dynamic unknown environments so that the robot can localize itself with respect to the surroundings, based on sensory information. Modern state-of-the-art motion estimators differ from classical Bayesian filtering approaches in that they consider a range of measurements (i.e. smoothing) stored in factor graph containers (Cadena *et al.*, 2016). For these approaches it is common to embed exteroceptive sensors such as cameras and Light Detection and Ranging (LiDAR)s using SLAM. With state-of-the-art sensor technology, robust perception systems are more readily available than ever, allowing for more potent motion estimation. Real-Time Kinematic-Global Navigation Satellite System (RTK-GNSS)' are examples of highly accurate localization units, being precise down to a few centimeters. However, the GNSS might not always be available or reliable. For example, it might fail due to hardware errors, jamming/spoofing, limited coverage, or, most commonly, human errors. In case the RTK-GNSS for some reason should fail it is important to have redundancy in a replacement system. One example of such a potent back-up alternative is the aforementioned SLAM approach. Highly accurate motion estimates could be calculated from up to several exteroceptive sensors, followed by fusion with sensor data from interoceptive measurements from for example IMUs or magnetic compasses. Thus, providing a comparable alternative to GNSS in the short term, exampled by both Campos *et al.* (2020) and Skjellaug (2020) for cameras and LiDARs respectively. While perceiving the environment, SLAM provides the additional benefit of creating a three dimensional reconstruction of the scene from exteroceptive sensor data, providing situational awareness of static objects in the scene.

The autonomous potential in the automotive industry have been extensively researched over the years (Bimbraw, 2015). An important reason behind this is that at least 90% of vehicle accidents are estimated to be the result of human error (Singh, 2015). Adopting autonomous vehicles could therefore have the potential of reducing or even eliminating the largest cause of car accidents, while also outperforming human drivers in perception, decision-making and execution. The commercial research has therefore seen a huge spike in interest. An example of a key contribution to this field is the KITTI dataset (Geiger, Lenz, Stiller, and Urtasun, 2013a). This is a publicly available dataset intended to serve as a benchmark for development of computer vision and robotic algorithms, targeted at autonomous driving. This, and many other publications, has thus lead to a rapid progression in the autonomous car industry.

Another application of autonomy that has received increasing attention is autonomous navigation by sea. Ever since Autonomous Surface Vehicle (ASV)s were first introduced by MIT in 1993 (Manley, 2008) extensive research has been conducted and several prototypes have been launched, demonstrating the capabilities of autonomous operation (Liu *et al.*, 2016). One example of the application of ASVs is autonomous ferries for passenger transportation. Golden *et al.* (2016) states that autonomous ferries have the potential to be more sustainable both in terms of cost and environmental footprint. More importantly, Jokioinen *et al.* (2016) explains that autonomous ships have the potential to match, or even improve, the accuracy of manned ships in the coming future. Today, human errors are the reason for more than 60% of ferry accidents and are accountable for more than 70% of the fatalities in these accidents (Golden *et al.*, 2016), hence autonomous ships could also have the potential to be safer than manned vehicles. The Autoferry project (Nilsen, 2017) at NTNU is a research project developing a fully autonomous ferry, called milliAmpere2, which transports passengers between Ravnkloa and Brattøra, in Trondheim. The passage is not easy to cross and milliAmpere2 will serve as a fully electrical replacement being able to transport people with just the push of a button. The ferry prototype, milliAmpere, is displayed in figure 1.1.1, and will be described in more detail in section 6.1.

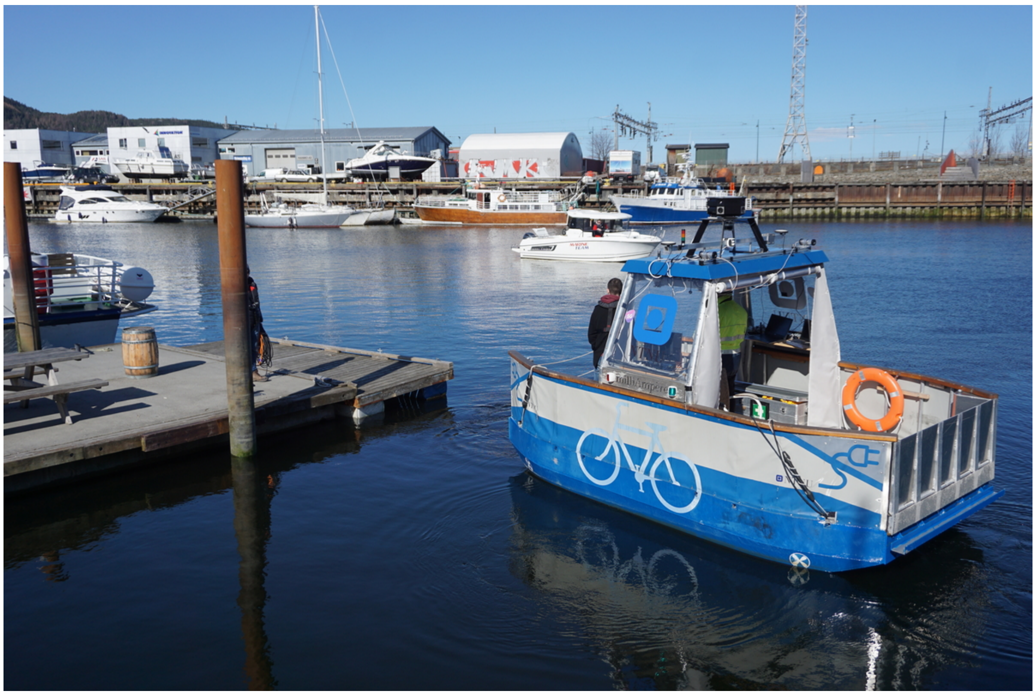


Figure 1.1.1: Image from data logging with milliAmpere in April 2021. Image captured by the author.

1.2 Contributions

The following contributions are listed for this master thesis:

1. A tailor-made data set has been recorded with a large variety of sensors using the autonomous ferry prototype, milliAmpere. The recorded data set was compared to suitable alternative real-world data sets.
2. An efficient GPU-accelerated feature-based VSLAM frontend has been developed for stereo cameras, largely inspired by Library for Visual Odometry 2 (LIBVISO2) (Geiger, Ziegler, *et al.*, 2011) and the Stereo Odometry Based on Feature Selection and Tracking 2 (SOFT2) (Cvišić *et al.*, 2018). The frontend uses bucketed FAST feature detection, Lucas Kanade optical flow feature tracking, circular matching, linear triangulation and both structure-only and motion only BA for initial motion estimates and 3D point detection.
3. Place recognition using DBoW2 (Gálvez-López *et al.*, 2012) was embedded in the frontend making the system capable of detecting loops by comparing ORB descriptors.
4. A concurrent short-term smoother and long term smoother solution (S. Williams *et al.*, 2014) using iSAM2 (Kaess, Johannsson, Roberts, Ila, J. J. Leonard, *et al.*, 2012) as the underlying update rule.
   (a) The short-term smoother performs windowed multi-frame bundle adjustment for stereo measurements. The short-term smoother is capable of fusing both GNSS and preintegrated IMU measurements in a modular approach. The modular strategy makes it easy to replace, improve and add modules, so that other sensors can be incorporated at a later stage.
   (b) The long-term smoother is capable of closing detected loops in a seperate thread, thus correcting for accumulated drift.
5. The complete system is embedded in ROS, fitting with the existing interface already placed on milliAmpere.
6. The developed VSLAM system is tested on milliAmpere, thus being one of the first image based SLAM approaches ever to be tested on a maritime surface vehicle, to the extent of the author's knowledge.

1.3 Related work

Preliminary research on estimating a vehicle's ego-motion using visual input was first described by Moravec (1980) in the early 1980s. This research was motivated by the desire to provide rovers the capability to estimate their 6-Degrees of Freedom (DOF) motion in the presence of wheel slippage and rough terrains. Nistér *et al.* (2004) later formalized this form of motion estimation by the term *Visual Odometry (VO)* in the first real-time implementation of a VO system. VO algorithms can be categorized as either feature-based, which retrieve the relative pose between images by extracting and matching keypoints from them, or direct methods, which directly compare pixel intensities to achieve the same result. Feature-based methods are also referred to as indirect methods.

SLAM is an extension of odometry where drift is removed by recognizing revisited locations and correcting for the accumulated drift to achieve globally consistent pose estimates rather than locally consistent, which is the case for VO. As research topics, VO and SLAM have gained a lot of traction and attention in the last 30 years. Earlier work on SLAM was focused on Bayesian filtering methods, such as Extended Kalman Filter (EKF), particle filters, etc. Durrant-Whyte *et al.* (2006) provide a comparison for methods up til 2006. In the last decade, non-linear filtering based approaches have been replaced by optimization-based approaches, called BA. These have proved to be more accurate and efficient (Strasdat *et al.*, 2010). Cadena *et al.* (2016) provides an in-depth description of the development of SLAM throughout history, as well as a survey of the current state of SLAM together with future directions.

More recent work formulates the SLAM in terms of *factor graphs*, alternatively the closely related *hyper graphs*, which are then optimized in batches. There exists many optimization libraries, such as Georgia Tech Smoothing and Mapping library (GTSAM) (Dellaert, 2012), g^2o (Kummerle *et al.*, 2011), the Ceres-solver (Agarwal *et al.*, n.d.) and miniSAM (Dong *et al.*, 2019). Studies (Doaa *et al.*, 2013), (Youyang *et al.*, 2020), (Grisetti *et al.*, 2020), tend towards GTSAM and g^2o being the most efficient batch algorithms, with g^2o having a slight advantage. However, research conducted by Kaess, Johannsson, Roberts, Ila, J. J. Leonard, *et al.* (2012) created an algorithm for sparse nonlinear incremental optimization, named iSAM2, fitting nicely with the incremental nature of measurement updates in SLAM problems. The factor graph structure simplifies the integration of measurements in the problem formulation. Factor graphs can also quite easily embed measurements from other sensors, making it far more likely that the optimization converges to a consistent and correct estimate. Over the years Dellaert and Kaess (2017) have added several extension to the optimization library GTSAM (Dellaert, 2012), some of which are sliding window optimization (Chiu *et al.*, 2013), concurrent filtering and smoothing (S. Williams *et al.*, 2014) and smart factors (Carlone *et al.*, 2014).

The work by Mouragnon *et al.* (2006) was the first real-time application of VO using BA. This was followed by the ground-breaking Parallel Tracking and Mapping (PTAM) by Klein *et al.* (2007), where tracking and mapping of features were split into two threads. ORB-SLAM, created by Mur-Artal, Montiel, *et al.* (2015), builds on many of the ideas from PTAM. It uses ORB-features (described in section 3.2.1) for tracking, mapping, relocalization, and loop closing, of which all except relocalization are run in parallel threads. At the time of release, ORB-SLAM achieved unprecedented performance with respect to state-of-the-art SLAM systems. Since then, several direct and indirect SLAM systems have been released matching or outperforming ORB-SLAM, including new versions of ORB-SLAM (Mur-Artal and Tardós, 2017) and (Campos *et al.*, 2020). Indirect SLAM methods extract features from images to estimate motion, while direct SLAM methods directly optimize on the pixel intensities between images. Direct methods have proved to accurately estimate motion, but it is recognised that indirect methods often outperform the direct. Current state-of-the-art methods are so-called VIO which include IMU measurements in the pose estimation. ORB-SLAM3 provides a comparison of many of the best VO, VIO, and VSLAM methods available (Campos *et al.*, 2020).

Even though both VSLAM and ASVs have been extensively researched, little attention has been directed towards the application of SLAM in maritime harbor environments. Within the field of pose estimation, preliminary research was conducted by Ødven (2019) and Dalhaug (2019) in their thesis'. The former compared multiple LiDAR-VO and -SLAM algorithms available from open-source in Robot Operating System (ROS); none of which yield sufficiently accurate results for autonomous docking (Ødven, 2019). The latter thesis focused on localization using particle filters. Even though this yield a quite accurate pose estimation, particle filters turned out to be too computationally inefficient for real-time localization (Dalhaug, 2019). Skjellaug (2020) developed a feature-based LiDAR-SLAM system using the iSAM2 framework, incorporating both IMU measurements and the RTK-GNSS. With the RTK-GNSS disabled, this work presented results that achieved higher accuracy than a standard GNSS receiver, both for the two-dimensional xy-plane, and for the z-direction, in the short term. Experiences from this system forms the basis for further development. The most recent addition to the these research topics is the contribution from Gerhardsen (2021) which analyses pose estimation using fiducial markers in marine environments. This research show promising results and should be included as supplementary measurements for future work.

During the specialization project (Hellum, 2020), several state-of-the-art VO and VSLAM algorithms was studied. None of the existing open-source VO/VSLAM systems were considered as viable alternatives, mainly because of their tightly coupled structure and that they were not considered compatible with GTSAM. This would make it difficult to modify and fit such VSLAM systems with inclusion of other exteroceptive SLAM modules. Additionally the pose optimization would have to be performed twice, once for the original system and once where other sensors were included. The additional refinement from the secondary optimization module should then ideally be fed back to the original system. This can cause race conditions. The specialization project therefore concluded that a new VSLAM algorithm rather should be developed.

1.4 Relation to the specialisation project

The practical implementation from the specialisation project (Hellum, 2020) is completely reworked. However, experiences and results obtained from this work is of importance when further investigations are done into the concept of SLAM and sensor fusion in this thesis. Sections from the specialisation project that are found to be relevant for the master's thesis will therefore be included either in their original form or in a modified, redrafted version. Redrafted sections may include new parts added during the work with this thesis or just be updated with the latest information.

If a section is included in its original form or redrafted from the specialisation project it will be clearly stated. The sections these notes applies to are also summarised here for easy identification by the reader:

- Sections included in their original form include: 3
- Sections included in a modified form include: 2.2, 2.4, 4, 5.2 - 5.3

1.5 Outline

This master thesis is organized into 11 chapters. After this introductory chapter, the outline of the report is as follows:

- Chapter 2 introduces background information on pose estimation for both exteroceptive and interoceptive sensors. This includes cameras, IMUs and GNSS'.
- Chapter 3 introduces background information on feature management in images.
- Chapter 4 introduces fundamental statistics that are used in graph-based SLAM problems.
- Chapter 5 presents SLAM problem formulations in general. Backend formulation of factor graphs are further depicted, going into more details on the iSAM2 algorithm and concurrent short-term and long-term smoothing.
- Chapter 6 presents the sensor setup on milliAmpere and on the KITTI dataset, together forming the consolidated datasets. Then, details of the utilized software are presented in more detail.
- Chapter 7 goes into details on the structure of the frontend for the developed VSLAM algorithm. The approach of every developed component are then specified.
- Chapter 8 decribes how concurrent short-term and long-term smoothing is solved in practice, some of which with marginalization and loop closure. This is followed by more details on how the different sensors in practice are connected in a joint factor graph optimization.
- Chapter 9 describes results of data logging with milliAmpere and stereo calibration.
- Chapter 10 discusses results achieved by the developed multi-sensor SLAM algorithm on the consolidated datasets. Here, the results are compared to LIBVISO2 and ORB-SLAM2.
- Chapter 11 concludes the work of the master thesis and presents improvements for future work.

CHAPTER 2

Fundamentals of pose estimation from sensor data

This chapter presents the essential background information on pose estimation using different sensors. First, coordinate frames, rigid-body kinematics and Lie theory are described to explain motion of an object. Next, fundamentals for pose estimation using various sensors are explained.

2.1 Coordinate frames

It is important to understand how different bodies are oriented relative to each other. To understand this, one also have to understand how a vehicle body is oriented with respect to an earth-fixed position. Also, it is often practical to know how a sensor (e.g., a camera) is oriented relative to the vehicle or other sensors. This section describes the various coordinate systems used to describe the position and orientation of the vehicle and its sensors.

World frame: The world origin and coordinate basis vector is an arbitrarily chosen pose in which the state of the vessel is seen in reference to. This center is often referred to as the *world frame*, which will be denoted $\mathcal{F}_W$. To complete the parameterization of the coordinate frames the direction of the axes have to be defined. $\mathcal{F}_W$ is sometimes referred to as a North-East-Down (NED) coordinate frame, for example in seakeeping theory (Fossen, 2011) or for aerial vehicles (Beard *et al.*, 2012a). In this frame, x is pointing towards north, y towards east and z downwards. There are many alternative, for example by inverting the y and z direction compared to the NED frame, as is done for the GNSS/IMU in the KITTI dataset (Geiger, Lenz, Stiller, and Urtasun, 2013b).

Body/Vehicle frame: The *body frame*, $\mathcal{F}_B$, also referred to as the *vehicle frame*, is defined so that all coordinate frames on the vehicle can be described in relation to each other. The reference point of the body frame may be placed on an arbitrarily body fixed position but is often set to the center of orientation. For freestanding objects, this is typically the center of gravity. There are many choices for the orientation of the coordinate frame, but a common choice is placing x-axis in the the direction the vehicle is facing. Following the right-hand rule, the z-axis is typically either pointing upwards or downwards. If z is upwards, then y is pointing to the left, but if z is downwards, then y is pointing to the right.

Odometry frame: Is another frame that often are formulated for VO and SLAM scenarios, which will be denoted $\mathcal{F}_O$. This reference point is set to the initial pose of the body, which does not necessarily have to be the same as the world center.

Common sensor frames: Every sensor perceive the environment with respect to their own coordinate frame. For example, with cameras the coordinate frame place the reference point at the focal point. Following the *OpenCV* convention this frame places the x-axis pointing to the right, y downwards and z in the perceived direction of the camera. This is known as the camera frame, $\mathcal{F}_C$, and will be covered in more detail in chapter 2.4.1. Some other examples are the

LiDAR, which often has the x-axis pointing forward, y to the left, and z upwards, but this is not set in stone. IMUs and GNSSs, on the other hand, may use either the latter coordinate frame or inversing the y and z direction such that y is pointing to the right and z points downwards.

The relation between the different coordinate frames are illustrated in fig. 2.1.1. It is evident that some mathematics have to be formulated in order to relate the different coordinate frames both to perform sensor fusion and state estimation.

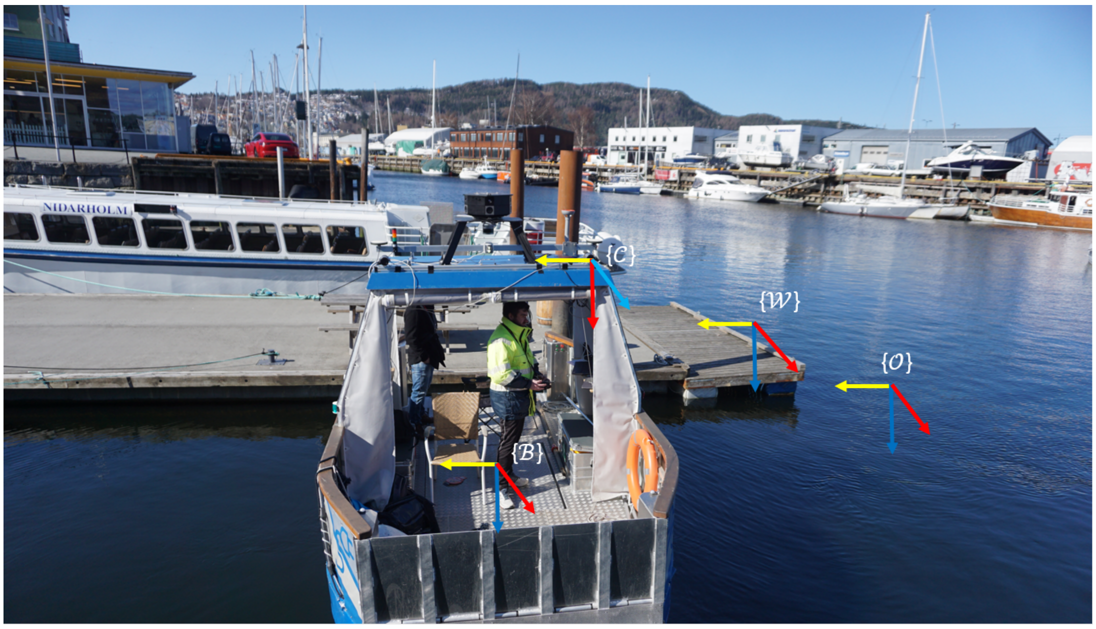


Figure 2.1.1: Illustration of coordinate the different types of frames that are typically referred to in SLAM scenarios. Sensor frames are illustrated in the image. The red arrow depicts the x-direction, while yellow is y-direction and z is marked as blue. Image captured by the author.

2.2 Rigid Body Kinematics

A presentation of the necessary background material related to Rigid body kinematics was included in the specialisation project preceding this thesis. This presentation is deemed valuable also for this thesis, and the presentation from the project report (Hellum, 2020) is therefore included below in a redrafted version.

A homogenous transformation matrix is a 4×4 matrix, $\mathbf{T}_{ab}$, which describes a change in pose from coordinate frame $\mathcal{F}_a$ to $\mathcal{F}_b$. This transformation can be decomposed into a rotation matrix, $\mathbf{R} \in \mathbb{R}^{3\times3}$, and a translation vector, $\mathbf{t} \in \mathbb{R}^{3\times1}$. The transformation matrix is an element of the special Euclidean Lie group in 3D, which can be used to describe motion of vehicles and vessels in 6 Degrees of Freedom (DOF). Lie groups will be covered in more detail in section 2.3. The rotation matrix describes rotational motion for roll, pitch and yaw, while the translation vector describes shift of the specified frame. The rotation matrix has to satisfy a set of properties. This is because the rotation matrix is part of a special orthogonal Lie group in 3D, falling under the following SO(3) constraints:

$$SO(3) = \left\{ \boldsymbol{R} | \boldsymbol{R} \in \mathbb{R}^{3\times3}, \boldsymbol{R}^T \boldsymbol{R} = \boldsymbol{I}, \mathrm{Det}(\boldsymbol{R}) = 1 \right\} \tag{2.2.0.1}$$

A relative pose representation has to satisfy the *SE(3)* property, extending the *SO(3)* according to

$$SE(3) = \left\{ \boldsymbol{T} | \boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4\times4}, \boldsymbol{R} \in SO(3), \boldsymbol{t} \in \mathbb{R}^3 \right\} \tag{2.2.0.2}$$

For SLAM it is common to track motion over time. A useful property that follows from the *SE(3)* constraint is that transformation matrices that are concatenated as the product of a matrix multiplication still satisfies *SE(3)*. A vector, $\boldsymbol{\xi}$, can be transformed from coordinate system $\mathcal{F}_B$ to $\mathcal{F}_A$ by first rotating the vector into the orientation of the new frame, $\mathcal{F}_A$, and then adding a translation, yielding

$$\boldsymbol{\xi}^a = \boldsymbol{R}_{ab}\boldsymbol{x}^b + \boldsymbol{t}^a_{ab}. \tag{2.2.0.3}$$

Updates for rotation is simply the matrix product with another rotation matrix. A combination of such transformations is shown in equation 2.2.0.4. The transformation is described from right to left starting in frame $\mathcal{F}_c$. First, an object described in $\mathcal{F}_c$ is transformed from $\mathcal{F}_c$ to $\mathcal{F}_b$. Then the motion of the object is described in $\mathcal{F}_a$ undergoes a transformation from $\mathcal{F}_b$ to $\mathcal{F}_a$. The overall transformation is from $\mathcal{F}_c$ to $\mathcal{F}_a$.

$$\boldsymbol{T}_{ac} = \boldsymbol{T}_{ab}\boldsymbol{T}_{bc} = \begin{bmatrix} \boldsymbol{R}_{ab}\boldsymbol{R}_{bc} & \boldsymbol{t}^a_{ab} + \boldsymbol{R}_{ab}\boldsymbol{t}^b_{bc} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \in SE(3) \tag{2.2.0.4}$$

If a coordinate frame, $\mathcal{F}_B$, is attached to the moving vehicle body, then the pose of the body frame relative to a fixed world coordinate frame, $\mathcal{F}_W$, can be captured by a set of transformations equal to the ones described in equation 2.2.0.4.

2.3 Lie Theory

Orientations, $SO(3)$, and poses, $SE(3)$, lie on manifolds in higher-dimensional spaces (Sola *et al.*, 2018), which complicates description of perturbations, derivatives and probability distributions since they do not have the same properties as vectors in vector spaces. This can be seen in examples 2.3.0.1a and 2.3.0.1b where the orientation and transformation are pushed outside the special orthogonal/euclidean group after small perturbations, δR and δT, are added. This further means that derivatives cannot properly be expressed by means of perturbations for rotation matrices.

$$\boldsymbol{R} + \delta\boldsymbol{R} \notin SO(3), \quad \boldsymbol{R}, \delta\boldsymbol{R} \in SO(3) \tag{2.3.0.1a}$$

$$\boldsymbol{T} + \delta\boldsymbol{T} \notin SE(3), \quad \boldsymbol{T}, \delta\boldsymbol{T} \in SE(3) \tag{2.3.0.1b}$$

However, as mentioned in section 2.2, *SO(3)* and *SE(3)* falls under matrix Lie groups on the smooth manifold. Lie theory describes the tangent space around elements of a Lie group in order to define the exact mappings between the tangent space and the manifold. The tangent space has the same number of dimensions as the number of degrees of freedom of the group transformations where a separate set of algebraic operations may be applied.

The tangent space at the identity $\mathfrak{m} = \mathcal{TM}_\mathcal{E}$ is called the Lie algebra of the manifold $\mathcal{M}$. Elements on the manifold can be mapped to/from the Lie algebra using the $\exp(\cdot)$ and $\log(\cdot)$ operations in equation 2.3.0.2.

$$\exp : \mathfrak{m} \to \mathcal{M}; \quad \mathcal{Y} = \exp\left(\boldsymbol{\tau}^\wedge\right) \tag{2.3.0.2a}$$

$$\log : \mathcal{M} \to \mathfrak{m}; \quad \boldsymbol{\tau}^\wedge = \log(\mathcal{Y}) \tag{2.3.0.2b}$$

The Lie algebra is a vector space with elements $\boldsymbol{\tau}^\wedge \in \mathfrak{m}$. The $(\cdot)^\wedge$ operator expresses the Lie algebra $\mathfrak{m}$ as a linear combination of some base elements, $\boldsymbol{E}_i$, forming the tangent vector space, $\mathbb{R}^m$. The inverse operation $(\cdot)^\vee$ uses basis vectors, $\boldsymbol{e}_i$ so that $\boldsymbol{e}_i^\wedge = \boldsymbol{E}_i$, to map the tangent vector space back to Lie algebra.

$$\text{Hat } : (\cdot)^\wedge : \mathbb{R}^m \to \mathfrak{m}; \quad \boldsymbol{\tau}^\wedge = \sum_{i=1}^{m} \boldsymbol{\tau}_i \mathbf{E}_i \tag{2.3.0.3a}$$

$$\text{Vee } : (\cdot)^\vee : \mathfrak{m} \to \mathbb{R}^m; \quad \boldsymbol{\tau} = \left(\boldsymbol{\tau}^\wedge\right)^\vee = \sum_{i=1}^{m} \boldsymbol{\tau}_i \mathrm{e}_i \tag{2.3.0.3b}$$

For notational convenience, a vectorized version of the exponential and logarithmic maps are adopted that allows to directly map vector elements $\boldsymbol{\tau} \in \mathbb{R}^m$ to group elements $\mathcal{Y} \in \mathcal{M}$. The direct mapping is expressed by capitalization

$$\text{Exp} : \mathbb{R}^m \to \mathcal{M}; \quad \mathcal{Y} = \text{Exp}(\boldsymbol{\tau}) = \exp\left(\boldsymbol{\tau}^\wedge\right) \tag{2.3.0.4a}$$

$$\text{Log} : \mathcal{M} \to \mathbb{R}^m; \quad \boldsymbol{\tau} = \text{Log}(\mathcal{Y}) = \log\left(\mathcal{Y}^\vee\right) \tag{2.3.0.4b}$$

The proper tools to perform operations such as pertubations on the manifold are now available through Lie theory. After expressing elements on the manifold as tangent space vectors using the Exp mapping they may be concatenated using the $\oplus$ or $\ominus$ operator, and then transform compossition back to the group using the Log mapping. Sola *et al.* (2018) goes into detail on mathematical operations that can be performed once transformed onto the tangent space.

2.4 Camera Geometry

A presentation of the necessary background material related to the Pinhole Model and Perspective Projection and Epipolar Geometry was included in the specialisation project preceding this thesis. This presentation is deemed valuable also for this thesis, and the presentation from the project report (Hellum, 2020) is therefore included below in a redrafted version.

In this section the essential background information on camera geometry and pose estimation using a camera will be presented. First, the geometry of the pinhole camera model explains how information from a 3D scene are related to the 2D image plane. The geometry that describes how motion can be estimated between images is detailed in the section about epipolar geometry. Additionally, the geometry of stereo cameras are explained.

2.4.1 Pinhole Model and Perspective Projection

Cameras are the most important sensor in the field of computer vision. The representation of an object in sensor data can typically be found through a sensor model which transforms a point in the world frame to a point in the sensor frame. There exist multiple models that map points captured from a 3D scene onto a 2D image plane, where the pinhole camera model is the most widely used (Hartley *et al.*, 2003). Another name for this model is the perspective camera model. An illustration of the geometry describing this camera model is shown in figure 2.4.1. In the pinhole model, the camera is imagined as a box with a small hole in the center. Reflected light from the scene passes through the hole illustrated in figure 2.4.1 and creates an inverted reflection on the image plane.

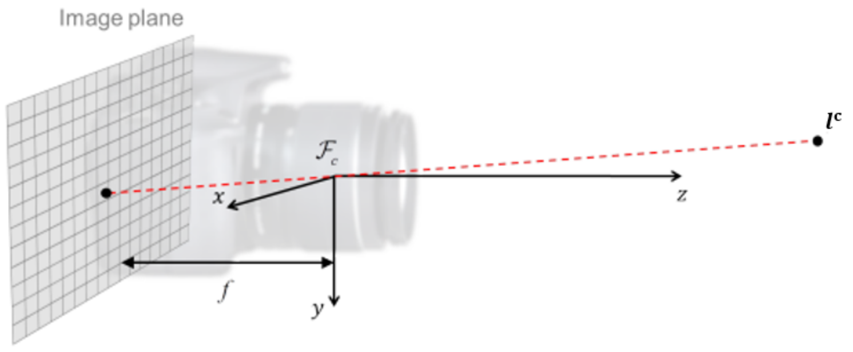


Figure 2.4.1: The perspective camera model. The camera is represented by the camera frame $\mathcal{F}_c$. Points $\mathbf{l}^c$ in the camera frame are projected through the origin and onto the image plane at a distance f behind the projective centre, where f is the camera constant. Reprinted by permission from Haavardsholm (2020). The image is slightly modified to fit the notation of this thesis.

The pinhole model is divided into extrinsic and intrinsic parameters. The extrinsic parameters define the rigid-body motion, T^c_{cw}, between the camera reference frame, $\mathcal{F}_c$ and a known world reference frame, $\mathcal{F}_\text{w}$. In homogeneous coordinates this can be expressed as

$$\begin{bmatrix} x^c \\ y^c \\ z^c \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{R}_{\text{cw}} & \boldsymbol{t}^c_{\text{cw}} \\ \mathbf{0}^T & 1 \end{bmatrix}}_{\boldsymbol{T}_{\text{cw}}} \begin{bmatrix} x^\text{w} \\ y^\text{w} \\ z^\text{w} \\ 1 \end{bmatrix} \tag{2.4.1.1}$$

Next, the intrinsic parameters are necessary to project coordinates given in the camera reference frame, $\mathcal{F}_c$ into pixel coordinates in image plane. The homogeneous representation of the world points in the camera frame, $[x^c, y^c, z^c, 1]^T$, can further be mapped into the Cartesian space using the homogeneous perspective projection matrix, $\mathbf{\Pi}$, i.e.,

$$\boldsymbol{x}^c = \begin{bmatrix} x^c \\ y^c \\ z^c \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{\Pi}} \boldsymbol{T}_{\text{cw}} \begin{bmatrix} x^{\text{w}} \\ y^{\text{w}} \\ z^{\text{w}} \\ 1 \end{bmatrix} \tag{2.4.1.2}$$

The image coordinates are represented by the vector $\boldsymbol{u} = [u, v]^T$. Using the law of similar triangles, the camera coordinates are normalized and further multiplied with the camera constant, f, of the camera to obtain the correct unit length representation of the pixels. The camera constant is the distance from the optical center to the image plane as illustrated in figure 2.4.1. Lastly, the image coordinates are defined with the upper left corner as the origin. The *principal point* $c = [c_u, c_v]^T$ is therefore added to the expression yielding the intrinsic equations below

$$u = f\frac{x^c}{z^c} + c_x \qquad v = f\frac{y^c}{z^c} + c_y \tag{2.4.1.3}$$

These intrinsic equations are often written on matrix form, yielding the camera calibration matrix.

$$\boldsymbol{K} = \begin{bmatrix} f_u & 0 & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4.1.4}$$

Putting the extrinsic and intrinsic equations together yields the pinhole model in homogemous form

$$\tilde{\boldsymbol{u}} = \underbrace{\boldsymbol{K}\left[\boldsymbol{R}_{\text{cw}} | \boldsymbol{t}^c_{\text{cw}}\right]}_{\boldsymbol{P}} \mathbf{l}^{\text{w}} \tag{2.4.1.5}$$

This equation projects 3D points in world coordinates, $\mathbf{l}^{\text{w}}$, onto homogenous image coordinates, where $\mathbf{P}$ is the projection matrix. The projection function can also be expressed in Euclidean form. In the camera frame, $\mathcal{F}_c$, this becomes

$$\boldsymbol{u} = \pi_p(\boldsymbol{l}^c; \boldsymbol{K}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \boldsymbol{K} \frac{1}{z^c} \boldsymbol{l}^c = \begin{bmatrix} f_u \frac{x^c}{z^c} + c_u \\ f_v \frac{y^c}{z^c} + c_v \end{bmatrix}. \tag{2.4.1.6}$$

In the world frame, $\mathcal{F}_{\text{w}}$, the extrinsics have to be included, yielding

$$\boldsymbol{u} = \pi_p(\boldsymbol{l}^{\text{w}}; \boldsymbol{T}_{\text{cw}}, \boldsymbol{K}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \boldsymbol{K} \frac{1}{z^c} \mathbf{\Pi} \boldsymbol{T}_{\text{cw}} \tilde{\boldsymbol{l}}^{\text{w}}. \tag{2.4.1.7}$$

Notice that for the world frame z^c is described by the relation in equation 2.4.1.2.

A camera lens may distort images, making straight lines in a scene appear bent in the image. Radial distortion occurs when light rays are bent closer to the edges of the lens than in the center. An ideal pinhole camera does not have a lens, and thus this is not accounted for in the intrinsic matrix. A distortion model is therefore often applied in addition to equalize distortion caused by the lens. Let (u, v) be the ideal points and (u_d, v_d), the radial distortion expressed in eq. (2.4.1.8).

$$\begin{aligned} u_d &= u + u\left[k_1\left(u^2+v^2\right) + k_2\left(u^2+v^2\right)^2 + k_3\left(u^2+v^2\right)^3 \ldots\right] \\ v_d &= v + v\left[k_1\left(u^2+v^2\right) + k_2\left(u^2+v^2\right)^2 + k_3\left(u^2+v^2\right)^3 \ldots\right] \end{aligned} \tag{2.4.1.8}$$

The radial distortion coefficients k_n express the degree of the radial distortion (Z. Zhang, 2000).

2.4.2 Epipolar geometry

Consider two perspective cameras, represented by the camera frames $\mathcal{F}_a$ and $\mathcal{F}_b$. The cameras are related by a relative transformation, $\{\mathbf{R}_{ab}, \mathbf{t}^a_{ab}\} \in \mathbf{T}_{ab}$, as described in section 2.2. Observing the same world point, $\mathbf{l}$, from the two camera frames puts a geometric constraint on the point correspondence $\tilde{\boldsymbol{u}}_a \leftrightarrow \tilde{\boldsymbol{u}}_b$ in the two normalized image planes $\mathcal{I}_a$ and $\mathcal{I}_b$. This is called the epipolar constraint, and can be expressed as

$$(\tilde{\boldsymbol{u}}^a)^T \boldsymbol{E}_{ab} \tilde{\boldsymbol{u}}^b = 0, \quad \text{where} \quad \boldsymbol{E}_{ab} = [\boldsymbol{t}^a_{ab}]^\times \boldsymbol{R}_{ab} \in \mathbb{R}^{3\times3} \tag{2.4.2.1}$$

where $[\cdot]^\times$ denotes the skew-symmetric operator. Equation 2.4.2.1 represents the epipolar constraint by the essential matrix, $\boldsymbol{E}_{ab}$. Another representation is by means of the fundamental matrix, $\boldsymbol{F}_{ab} = \boldsymbol{K}_a^{-T} \boldsymbol{E}_{ab} \boldsymbol{K}_b^{-1}$, where $\mathbf{K}$ represent the calibration matrix of camera frame $\mathcal{F}_a$ and $\mathcal{F}_b$. By this relationship, the essential matrix depends only on extrinsic parameters, thus representing the epipolar constraint in normalized image coordinates. The fundamental matrix on the other hand express correspondence relation in pixel coordinates.

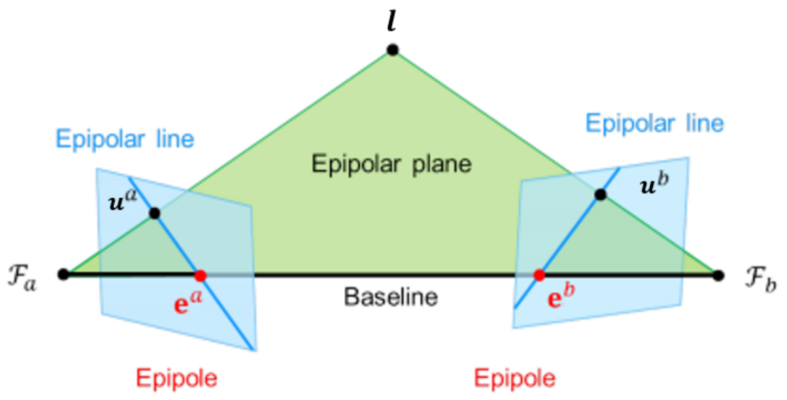


Figure 2.4.2: The image illustrates the epipolar geometry relating two perspective camera frames $\mathcal{F}_a$ and $\mathcal{F}_b$. $\mathbf{l}$ describes a world point observed from both camera frames. The camera centers and the world point forms a triangle called the epipolar plane. The baseline is the line drawn between the two camera centers, while the epipoles, $\mathbf{e}^a$ and $\mathbf{e}^b$, are the intersection of the baseline in the image planes. The epipolar lines fall where the epipolar plane intersects the image planes. The epipolar constraint says that the world point projected onto the image plane, i.e. $\mathbf{u}^a$ and $\mathbf{u}^b$, must lie on this epipolar line. Reprinted by permission from Haavardsholm (2020). The image is slightly modified to fit the notation of this thesis.

Figure 2.4.2 illustrates the epipolar geometry between the two camera frames. The *epipolar plane* describes the plane containing the 3D point $\mathbf{l}$. The *baseline* is the line joining the two camera centres. Furthermore, the *epipoles* are the intersection of the baseline in each image plane. The *epipolar lines* are then found by drawing a line segment where the epipolar plane intersects the image planes. The epipolar line in image $\mathcal{I}_a$ intersects its epipole e^a and the point $\mathbf{u}^a$. The corresponding point in image $\mathcal{I}_b$ must then lie on the epipolar line for image $\mathcal{I}_a$. Search for correspondences is then reduced from a region to the epipolar line. Furthermore, the depth is directly proportional to the length of the baseline and the distance between the two pixels capturing the world object.

The fundamental matrix can be estimated from the 7- or 8-point algorithms, while the essential matrix can be estimated from 5 point correspondences using Nistérs five point algorithm (Nistér, 2004).

2.5 Stereo Vision

When a point in 3D space is projected onto the 2D image plane, the depth dimension is completely lost. This is intuitively similar to human vision, where it becomes difficult to determine the distance to objects with one eye kept closed. As a consequence of the lost depth information will monocular SLAM, i.e. using a single camera, be unable to recover the scale of the scene and thus the scale of the traversed trajectory. By using stereo cameras, i.e. two rigidly mounted cameras observing the same scene, the depth information can be recovered by triangulating stereo matched features at every frame. Consequently, challenges related to scale drift is eliminated.

Two stereo images observing an overlapping scene is related through epipolar geometry described in the previous section. This way, all point correspondences between the images are constrained to lie on the epipolar lines associated with the observed 3D point. There follows an uncertainty for the reconstructed depth of a 3D point. This uncertainty is mainly dependent on the baseline and image resolution, which generally decreases with an expanding baseline and a higher pixel intensity. A rule of thumb is that the working distance, i.e. the distance for which the position of the 3D point can accurately be determined, is 30 times the baseline (Curtis, 2011). However, more advanced methods can be applied for depth estimating which greatly increases the accuracy for larger distances (Pinggera *et al.*, 2014).

When stereo cameras are calibrated, extrinsic parameters describing the interrelating transformation in the camera setup is calculated. In reality there is usually an unwanted vertical rotation and transformation component in the stereo setup. This can be compensated for by applying the calibration result in a stereo *rectification* procedure for captured image pairs. In a rectified stereo setup the two image planes are perfectly aligned so that the epipolar plane, depicted in fig. 2.4.2, is horizontal. Additionally are the focal lengths f and optical centers (c_u, c_v) constrained to be equivalent for both cameras. As a result of rectification there should be no vertical discrepancies between two rectified image points, $\mathbf{u}_l = (u_l, v_l)^T$ and $\mathbf{u}_r = (u_r, v_r)^T$, in the left and right image respectively corresponding to the same landmark, $\mathbf{l}$. Consequently, is the search space for stereo point matches now only restricted to a search along the epipolar lines, where the correspondences rather can be represented with three parameters (u_l ,u_r ,v).

2.6 Inertial Measurement Unit (IMU)

IMUs are interoceptive sensors used for navigation and state estimation, which measure relative motion in the body frame. An IMU includes a gyroscope and an accelerometer to measure the body's angular rate, $\tilde{\boldsymbol{\omega}}^{\mathrm{b}}_{\mathrm{wb}}(t)$, and linear acceleration, $\tilde{\boldsymbol{a}}^{\mathrm{b}}(t)$, in equations 2.6.0.1a and 2.6.0.1b respectively. These measurements are often supplied by magnetic or gyroscopic compasses, $\tilde{\boldsymbol{m}}^{\mathrm{b}}(t)$, to obtain the orientation of the body as described in equation 2.6.0.1c. These measurements are affected by additive white noise, $\boldsymbol{\eta}$, and a slowly varying sensor bias, $\boldsymbol{b}$, both separately specified for the gyro, accelerometer and magnetometer. A separate superscript convention is used for the bias and noise, where g, a, m is used to denote measurements for gyro, accelerometer and magnetometer respectively; All of which are described in the body frame, where the $\boldsymbol{R}_{\mathrm{bw}}$ is the rotation from the world to body frame.

$$\tilde{\boldsymbol{\omega}}^{\mathrm{b}}(t) = \boldsymbol{\omega}^{\mathrm{b}}(t) + \mathbf{b}^{g}(t) + \boldsymbol{\eta}^{g}(t) \tag{2.6.0.1a}$$

$$\tilde{\boldsymbol{a}}^{\mathrm{b}}(t) = \boldsymbol{R}_{\mathrm{bw}}(t)\left(\boldsymbol{a}^{\mathrm{w}}(t) - \boldsymbol{g}^{\mathrm{w}}\right) + \boldsymbol{b}^{a}(t) + \boldsymbol{\eta}^{a}(t) \tag{2.6.0.1b}$$

$$\tilde{\boldsymbol{m}}^{\mathrm{b}}(t) = \boldsymbol{R}_{\mathrm{bw}}(t)\boldsymbol{m}^{\mathrm{w}}(t) + \boldsymbol{b}^{m}(t) + \boldsymbol{\eta}^{m}(t) \tag{2.6.0.1c}$$

The classic gyroscope is a spinning wheel that utilizes conservation of momentum to detect rotation, however, modern solutions come in several forms. Optical gyros are a popular choice for high accuracy strapdown inertial systems. The Microelectromechanical Systems (MEMS) technology have however made it possible to place an IMU on a small electrical chip, making the technology available for low and medium cost applications. Accelerometers are either *mechanical* or *vibratory*. The mechanical accelerometer uses Newton's second law to measure a force acing on the body, for example a pendulum. The vibratory accelerometer measures frequency shifts in a string, due to increased or decreased tension caused external forces acting on the body. A stand-alone IMU solution for pose estimation, where acceleration measurements are integrated twice and gyro outputs are integrated once to obtain positions and attitude respectively, is insufficient for long term navigation as the estimates will drift due to sensor biases, misalignments and temperature variations. However, using sensor fusion, IMU measurements can be a valuable addition in the overall system configuration by constraining the outcome space of the joint state estimate.

2.6.1 IMU preintegration

Lupton *et al.* (2011) introduced IMU preintegration as a method of combining measurements between two frames into one relative motion constraint, thus being able to calculate the motion a vehicle over time from a series of IMU measurements. Forster, Carlone, *et al.* (2015) later extended this theory to the SO(3) rotation group and fit the mathematics into a factor graph representation. This will be discussed in section 5.2.1. This representation is the solution that is currently implemented in GTSAM (Dellaert, 2012). The derivation discussed in this chapter follows from Forster, Carlone, *et al.* (2015).

Before applying the preintegration, the following kinematic model is introduced to describe motion between frames:

$$\dot{\boldsymbol{R}}_{\text{wb}} = \boldsymbol{R}_{\text{wb}} \cdot \left[\boldsymbol{\omega}^{\text{b}}\right]^{\times} \tag{2.6.1.1}$$

$$\dot{\boldsymbol{v}}^{\text{w}} = \boldsymbol{a}^{\text{w}} \tag{2.6.1.2}$$

$$\dot{\boldsymbol{p}}^{\text{w}} = \boldsymbol{v}^{\text{w}} \tag{2.6.1.3}$$

Here, $[\cdot]^{\times}$ describes the skew symetric matrix. To find the relative motion at time $t+\Delta t$, between two IMU measurements, equations 2.6.1.1 are integrated. If $\boldsymbol{a}^{\text{w}}$ and $\boldsymbol{\omega}^{\text{b}}$ are assumed constant in the time interval $[t, t+\Delta t]$, the states can be written as a function of the measurements as shown in equation 2.6.1.4.

$$\begin{aligned}
\boldsymbol{R}_{\text{w}b}(t+\Delta t) &= \boldsymbol{R}_{\text{w}b}(t)\,\text{Exp}\left\{\boldsymbol{\omega}^{\text{b}}(t)\Delta t\right\} && \text{(2.6.1.4a)}\\
&= \boldsymbol{R}_{\text{wb}}(t)\,\text{Exp}\left\{\left(\tilde{\boldsymbol{\omega}}^{\text{b}}(t) - \mathbf{b}^{g}(t) - \boldsymbol{\eta}^{g}(t)\right)\Delta t\right\}\\
\boldsymbol{v}^{\text{w}}(t+\Delta t) &= \boldsymbol{v}^{\text{w}}(t) + \boldsymbol{a}^{\text{w}}(t)\Delta t && \text{(2.6.1.4b)}\\
&= \boldsymbol{v}(t) + \boldsymbol{g}^{\text{w}}\Delta t + \boldsymbol{R}_{\text{wb}}(t)\left(\tilde{\boldsymbol{a}}^{\text{b}}(t) - \boldsymbol{b}^{a}(t) - \boldsymbol{\eta}^{a}(t)\right)\Delta t\\
\boldsymbol{p}^{\text{w}}(t+\Delta t) &= \boldsymbol{p}^{\text{w}}(t) + \boldsymbol{v}^{\text{w}}(t)\Delta t + \frac{1}{2}\boldsymbol{a}^{\text{w}}(t)\Delta t^{2} && \text{(2.6.1.4c)}\\
&= \boldsymbol{p}^{\text{w}}(t) + \boldsymbol{v}^{\text{w}}(t)\Delta t + \frac{1}{2}\boldsymbol{g}^{\text{w}}\Delta t^{2} + \frac{1}{2}\boldsymbol{R}_{\text{wb}}(t)\left(\tilde{\boldsymbol{a}}^{\text{b}}(t) - \boldsymbol{b}^{a}(t) - \boldsymbol{\eta}^{a}(t)\right)\Delta t^{2}
\end{aligned}$$

where the measurement equations in 2.6.0.1 are inserted after the second equality of each subequation in 2.6.1.4. Exp$\{\cdot\}$ denotes the lie exponential. The relative motion between two measurements are now known, hence the next step is to concatenate all measurements with Δt intervals between two consecutive keyframes at times $k = i$ and $k = j$.

$$\boldsymbol{R}_{j} = \boldsymbol{R}_{i}\prod_{k=i}^{j-1}\text{Exp}\left(\left(\tilde{\boldsymbol{\omega}}_{k} - \boldsymbol{b}_{k}^{g} - \boldsymbol{\eta}_{k}^{g}\right)\Delta t\right), \tag{2.6.1.5a}$$

$$\boldsymbol{v}_{j} = \boldsymbol{v}_{i} + \boldsymbol{g}\Delta t_{ij} + \sum_{k=i}^{j-1}\boldsymbol{R}_{k}\left(\tilde{\boldsymbol{a}}_{k} - \boldsymbol{b}_{k}^{a} - \boldsymbol{\eta}_{k}^{a}\right)\Delta t \tag{2.6.1.5b}$$

$$\boldsymbol{p}_{j} = \boldsymbol{p}_{i} + \sum_{k=i}^{j-1}\left[\boldsymbol{v}_{k}\Delta t + \frac{1}{2}\boldsymbol{g}\Delta t^{2} + \frac{1}{2}\boldsymbol{R}_{k}\left(\tilde{\boldsymbol{a}}_{k} - \boldsymbol{b}_{k}^{a} - \boldsymbol{\eta}_{k}^{a}\right)\Delta t^{2}\right]. \tag{2.6.1.5c}$$

In equation 2.6.1.5 the sub- and superscripts for frame description are dropped for readability. Also, $\Delta t_{ij} = \sum_{k=i}^{j-1}\Delta t$ and $(\cdot)_i = (\cdot)(t_i)$. While providing an estimate of the relative motion between t_i and t_j, equation 2.6.1.5 has the drawback that the integration has to be repeated whenever the linearization point at time t_i changes. To avoid this recomputation, the relative motion increments are assumed to be approximately independently of the pose and velocity at t_i giving the expression in eq. (2.6.1.6).

$$\Delta \mathbf{R}_{ij} = \boldsymbol{R}_i^T \boldsymbol{R}_j = \prod_{k=i}^{j-1} \operatorname{Exp}\left(\left(\tilde{\boldsymbol{\omega}}_k - \boldsymbol{b}_k^g - \boldsymbol{\eta}_k^g\right)\Delta t\right) \tag{2.6.1.6a}$$

$$\Delta \mathbf{v}_{ij} = \boldsymbol{R}_i^T \left(\boldsymbol{v}_j - \boldsymbol{v}_i - \boldsymbol{g}\Delta t_{ij}\right) = \sum_{k=i}^{j-1} \Delta \mathbf{R}_{ik}\left(\tilde{\boldsymbol{a}}_k - \boldsymbol{b}_k^a - \boldsymbol{\eta}_k^a\right)\Delta t \tag{2.6.1.6b}$$

$$\begin{aligned} \Delta \mathbf{p}_{ij} &= \boldsymbol{R}_i^T \left(\boldsymbol{p}_j - \boldsymbol{p}_i - \boldsymbol{v}_i \Delta t_{ij} - \frac{1}{2}\sum_{k=i}^{j-1} \boldsymbol{g}\Delta t^2\right) \\ &= \sum_{k=i}^{j-1}\left[\Delta \mathbf{v}_{ik}\Delta t + \frac{1}{2}\Delta \mathbf{R}_{ik}\left(\tilde{\boldsymbol{a}}_k - \boldsymbol{b}_k^a - \boldsymbol{\eta}_k^a\right)\Delta t^2\right] \end{aligned} \tag{2.6.1.6c}$$

These equations require knowledge of the bias, however, consider that the bias is slow-varying, the bias can be assumed to remain constant between two keyframes.

$$\boldsymbol{b}_i^g = \boldsymbol{b}_{gyro,i+1} = \ldots = \boldsymbol{b}_{gyro,j-1}, \quad \boldsymbol{b}_i^a = \boldsymbol{b}_{acc,i+1} = \ldots = \boldsymbol{b}_{acc,j-1}$$

Lie algebra, discussed in (Forster, Carlone, *et al.*, 2015), may further be applied for updating the rotational group SO(3). An example is the first-order approximation $\operatorname{Exp}(\zeta + \Delta\boldsymbol{\zeta}) \approx \operatorname{Exp}(\boldsymbol{\zeta})\operatorname{Exp}\left(\mathbf{J}_r(\boldsymbol{\zeta})\Delta\boldsymbol{\zeta}\right)$, here expressed with a random variable ζ. The jacobian $\mathbf{J}_r$ is the right Jacobian of SO(3) computed using Lie algebra.

Equation 2.6.1.6 should be modified further to isolate the noise. Therefore, starting with the rotation increment $\Delta\mathbf{R}_{ij}$, a first-order approximation is used to rearrange the terms by "moving" the noise to the end.

$$\begin{aligned} \Delta \mathbf{R}_{ij} &\simeq \prod_{k=i}^{j-1}\left[\operatorname{Exp}\left(\left(\tilde{\boldsymbol{\omega}}_k - \mathbf{b}_i^g\right)\Delta t\right)\operatorname{Exp}\left(-\mathbf{J}_{r,k}\boldsymbol{\eta}_k^g \Delta t\right)\right] \\ &= \Delta\tilde{\mathbf{R}}_{ij}\prod_{k=i}^{j-1}\operatorname{Exp}\left(-\Delta\tilde{\mathbf{R}}_{k+1j}^{\top}\mathbf{J}_{r,k}\boldsymbol{\eta}_k^g\Delta t\right) \\ &\doteq \Delta\tilde{\mathbf{R}}_{ij}\operatorname{Exp}\left(-\delta\phi_{ij}\right) \end{aligned} \tag{2.6.1.7}$$

with $\mathrm{J}_r^k \doteq \mathrm{J}_r^k\left(\tilde{\boldsymbol{\omega}}_k - \mathbf{b}_i^g\right)$. The *preintegrated rotation measurement* is defined as $\Delta\tilde{\mathrm{R}}_{ij} \doteq \prod_{k=i}^{j-1}\operatorname{Exp}\left(\left(\tilde{\boldsymbol{\omega}}_k - \mathbf{b}_i^g\right)\Delta t\right)$, and its noise $\delta\phi_{ij}$. Substituting equation 2.6.1.7 back into equation 2.6.1.6b and dropping higher-order noise yields

$$\begin{aligned} \Delta \mathbf{v}_{ij} &\simeq \sum_{k=i}^{j-1}\Delta\tilde{\mathbf{R}}_{ik}\left(\mathbf{I} - \delta\boldsymbol{\phi}_{ik}^{\wedge}\right)\left(\tilde{\mathbf{a}}_k - \mathbf{b}_i^a\right)\Delta t - \Delta\tilde{\mathbf{R}}_{ik}\boldsymbol{\eta}_k^a\Delta t \\ &= \Delta\tilde{\mathbf{v}}_{ij} + \sum_{k=i}^{j-1}\left[\Delta\tilde{\mathbf{R}}_{ik}\left(\tilde{\mathbf{a}}_k - \mathbf{b}_i^a\right)^{\wedge}\delta\phi_{ik}\Delta t - \Delta\tilde{\mathbf{R}}_{ik}\boldsymbol{\eta}_k^a\Delta t\right] \\ &\doteq \Delta\tilde{\mathbf{v}}_{ij} - \delta\mathbf{v}_{ij} \end{aligned} \tag{2.6.1.8}$$

where the *preintegrated velocity measurement* is defined as $\Delta\tilde{\mathbf{v}}_{ij} \doteq \sum_{k=i}^{j-1}\Delta\tilde{\mathbf{R}}_{ik}\left(\tilde{\mathbf{a}}_k - \mathbf{b}_i^a\right)\Delta t$, and its noise $\delta\mathbf{v}_{ij}$ Similarly, substituting equation 2.6.1.7 into equation 2.6.1.6c, and using a first-order approximation the following relation is obtained

$$\begin{aligned}
\Delta\mathbf{p}_{ij} &\simeq \sum_{k=i}^{j-1} \frac{3}{2}\Delta\tilde{\mathbf{R}}_{ik}\left(\mathbf{I} - \delta\phi_{ik}^{\wedge}\right)\left(\tilde{\mathbf{a}}_k - \mathbf{b}_i^a\right)\Delta t^2 - \sum_{k=i}^{j-1}\frac{3}{2}\Delta\tilde{\mathbf{R}}_{ik}\boldsymbol{\eta}_k^a\Delta t^2 \\
&= \Delta\tilde{\mathbf{p}}_{ij} + \sum_{k=i}^{j-1}\left[\frac{3}{2}\Delta\tilde{\mathbf{R}}_{ik}\left(\tilde{\mathbf{a}}_k - \mathbf{b}_i^a\right)^{\wedge}\delta\phi_{ik}\Delta t^2 - \frac{3}{2}\Delta\tilde{\mathbf{R}}_{ik}\boldsymbol{\eta}_k^a\Delta t^2\right] \\
&\doteq \Delta\tilde{\mathbf{p}}_{ij} - \delta\mathbf{p}_{ij}
\end{aligned} \tag{2.6.1.9}$$

Here, $\Delta\tilde{\mathbf{p}}_{ij}$ defines the *preintegrated position measurement* and its noise $\delta\mathbf{p}_{ij}$. Substituting all of the expressions 2.6.1.7, 2.6.1.8, 2.6.1.9 back in the original definition in equation 2.6.1.6, the final expression for the preintegrated measurement model becomes

$$\begin{aligned}
\Delta\tilde{\mathbf{R}}_{ij} &= \mathbf{R}_i^{\top}\mathbf{R}_j \operatorname{Exp}\left(\delta\boldsymbol{\phi}_{ij}\right) \\
\Delta\tilde{\mathbf{v}}_{ij} &= \mathbf{R}_i^{\top}\left(\mathbf{v}_j - \mathbf{v}_i - \mathbf{g}\Delta t_{ij}\right) + \delta\mathbf{v}_{ij} \\
\Delta\tilde{\mathbf{p}}_{ij} &= \mathbf{R}_i^{\top}\left(\mathbf{p}_j - \mathbf{p}_i - \mathbf{v}_i\Delta t_{ij} - \frac{1}{2}\mathbf{g}\Delta t_{ij}^2\right) + \delta\mathbf{p}_{ij}
\end{aligned} \tag{2.6.1.10}$$

So far, the bias $\mathbf{b}_i$ used to compute the preintegrated measurements has been assumed given. However, the bias term will likely change during optimization. One solution would be to re-compute the delta measurements when the bias changes, but that would be computationally expensive. Instead, given a bias update $\mathbf{b} \leftarrow \bar{\mathbf{b}} + \delta\mathbf{b}$, the delta measurements in equation 2.6.1.10 can rather be updated using a first-order expansion, giving

$$\begin{aligned}
\Delta\tilde{\mathbf{R}}_{ij}\left(\mathbf{b}_i^g\right) &\simeq \Delta\tilde{\mathbf{R}}_{ij}\left(\bar{\mathbf{b}}_i^g\right)\operatorname{Exp}\left(\frac{\partial\Delta\bar{\mathbf{R}}_{ij}}{\partial\mathbf{b}^g}\delta\mathbf{b}^g\right) \\
\Delta\tilde{\mathbf{v}}_{ij}\left(\mathbf{b}_i^g, \mathbf{b}_i^a\right) &\simeq \Delta\tilde{\mathbf{v}}_{ij}\left(\bar{\mathbf{b}}_i^g, \bar{\mathbf{b}}_i^a\right) + \frac{\partial\Delta\bar{\mathbf{v}}_{ij}}{\partial\mathbf{b}^g}\delta\mathbf{b}_i^g + \frac{\partial\Delta\bar{\mathbf{v}}_{ij}}{\partial\mathbf{b}^a}\delta\mathbf{b}_i^a \\
\Delta\tilde{\mathbf{p}}_{ij}\left(\mathbf{b}_i^g, \mathbf{b}_i^a\right) &\simeq \Delta\tilde{\mathbf{p}}_{ij}\left(\bar{\mathbf{b}}_i^g, \bar{\mathbf{b}}_i^a\right) + \frac{\partial\Delta\bar{\mathbf{p}}_{ij}}{\partial\mathbf{b}^g}\delta\mathbf{b}_i^g + \frac{\partial\Delta\bar{\mathbf{p}}_{ij}}{\partial\mathbf{b}^a}\delta\mathbf{b}_i^a
\end{aligned} \tag{2.6.1.11}$$

where $\left\{\frac{\partial\Delta\mathbf{R}_{ij}}{\partial\mathbf{b}^g}, \frac{\partial\Delta\bar{\mathbf{v}}_{ij}}{\partial\mathbf{b}^g}, \ldots\right\}$ is the jacobians computed at $\mathbf{b}_i$. The final formulation in equation 2.6.1.11 is important to fit the factor graph optimization structure described in section 5.2.1. A description of how preintegrated measurements are added to factor graphs will be covered in section 8.3.

2.7 Global Navigation Satellite System (GNSS)

There are several GNSSs that together provide autonomous geospatial positioning with global coverage. In the United States, the Global Positioning System (GPS) was created in 1995, followed by the Russian Global Navigation Satellite System (GLONASS) shortly after. The more recent Galileo by the EU and BeiDou by China provide an improved accuracy to the old systems. In addition there exists several regional systems. The GNSS system uses satellites for absolute global positioning. The constellation of the satellites is designed such that any point on the earth's surface is observable by at least four satellites at all times. GNSS receivers then uses the time of flight from a minimum of four satellites the receiver can triangulate, and thus determine its latitude, longitude and altitude. The fourth satellite is necessary because of clock synchronization errors between the satellites. The accuracy of the GNSS position is affected by the geometry of the satellites and the accuracy of the satellite pseudorange measurements. It is therefore evident that the accuracy of the integrated GNSS receivers benefit from combining signals from one or more systems. A regular receiver is low-cost, and typically has a specified Root Square Error (RSE) of about 4 meters (Fossen, 2011). As described by Beard *et al.* (2012), the accuracy of the GNSS measurements are affected by:

- *Ephemeris data*: The mathematical description of its orbit.
- *Satellite clock error*: Internal clock error of satellite.
- *Ionospheric delay* of the signal caused by the presence of free electors.
- *Troposphere disturbances* caused by variations in temperature pressure and humidity affect the time of flight.
- *Multipath reception* occurs when signals are reflected on surfaces before reaching the receiver.
- *Receiver measurement errors* stem from the computational limits with which the timing of the satellite signal can be resolved.

Additional types of sensors may also be available to ensure reliability of the positioning system, forming an Inertial Navigation System (INS). Such sensors may include IMUs, hydro acoustic position sensors, taut wires and laser sensors. There also exist other types of GNSS systems that improves the accuracy of the GNSS receiver positioning. This includes the *Differential and Augmented GNSS (DGNSS)* which uses stationary stations on the earth surface with known position to correct for errors, and the even more accurate RTK-GNSS which will be described in the next section.

The GNSS measurement equation is given by

$$\mathbf{z}_t^{gnss} = h^{gnss}\left(\mathbf{x}_t\right) + \eta_{gnss} \tag{2.7.0.1}$$

where η_{gnss} is the measurement noise which is assumed Gaussian distributed. $\mathbf{h}^{gnss}$ is the measurement function that relates the measurement $\mathbf{z}_t^{gnss}$ to the robot's position.

2.7.1 Real-Time Kinematic-Global Navigation Satellite System (RTK-GNSS)

A RTK-GNSS receiver is a highly accurate solution which provides accuracy down to a few centimeters, however the RTK-GNSS is significantly more expensive. The increased accuracy is achieved by tracking the phase shift of the signal's carrier wave and output the fractional phase measurement at each epoch. In order to lock on to this track the RTK-GNSS first has to determine the integer ambiguity, that is the unknown number of carrier cycles from the time a satellite signal is placed to the receiver begins an active track. These position measurements of the RTK-GNSS are, however, not as robust as GNSS and DGNSS (Fossen, 2011).

CHAPTER 3

Fundamentals of Feature Management

A presentation of the necessary background material related to feature management was included in the specialisation project preceding this thesis. The applied methods are the same as in the specialisation project and the theoretical description from the project report (Hellum, 2020) is therefore included below in its original version.

In this chapter the relevant background information on feature management is presented. These are essential topics within feature-based VO and SLAM systems. First, detection of features in images are described, with Features from Accelerated Segment Test (FAST) as the method of choice. Then, feature descriptors and more specifically Oriented FAST and Rotated BRIEF (ORB) is covered. Lastly, pyramidal Lucas-Kanade describes how features are tracked from image to image using optical flow.

3.1 Feature Extraction

As mentioned in section 5.1.1 features is an essential component in indirect SLAM methods. Features are significant image points that stand out in the texture of a scene. These points typically include corners or edges. What separates these points from planar areas in an image is that the pixel intensity typically changes. The intensity typically refers to the brightness or gradients of the pixel. These types of image points may be used for various tasks such as object classification, face recognition, etc., but popular demand is feature-based SLAM. The reason being that features that stand out in the scene can more easily be tracked and compared to other images capturing features of the same scene from another pose.

3.1.1 FAST

Feature detectors such as SIFT and Harris are accurate methods that yield high-quality features, however, they are computationally expensive which limits them from real-time applications. The FAST algorithm was first proposed by Rosten *et al.* (2006) to provide a faster feature detector aimed at real-time applications.

The FAST algorithm selects a pixel p in the image. For this interest-point the pixel brightness is compared to the surrounding 16 pixels, forming a Bresenham circle around p as shown in figure 3.1.1. The 16 pixels in the circle is classified as either lighter than p, darker than p or similar to p. The interest-point is selected as a keypoint if more than 8 pixels are either darker or brighter than p. Non-maximum Suppression is applied to reduce the number of interest points in adjacent locations. Amongst two adjacent keypoints the interest-point which inhabits a lower score function, i.e. the sum of the absolute difference between p and the 16 surrounding pixels, is discarded.

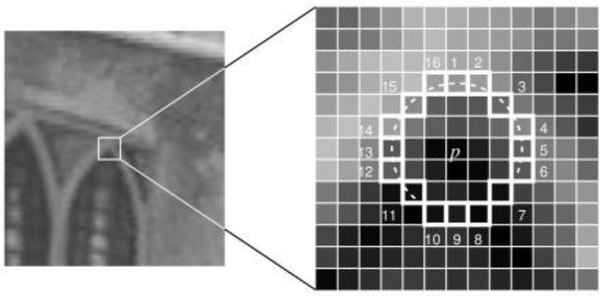


Figure 3.1.1: The image illustrates the image patch used by FAST to determine if an interest point satisfy the segment test. The pixel, p, is the centre of a candidate corner, while the 16 highlighted pixels forming the Bresenham circle is the compared pixels. Image courtesy of (Rosten *et al.*, 2006)

3.2 Feature Descriptors

Feature descriptors encode interesting information about features into a series of numbers. These descriptions act as numerical "fingerprint" so that features may be differentiated from one another. One alternative is to describe these features in the form of binary bit strings, forming binary feature vectors for the set of features. There exist a huge variety of feature detectors and descriptors, and Pire *et al.* (2017) provide a comparison of combinations applied to a VO framework.

3.2.1 ORB

ORB was developed by OpenCV Labs (Rublee *et al.*, 2011) as an efficient and viable alternative to SIFT and SURF. Both of the latter approaches are patented, thus ORB was developed as a free alternative to these algorithms. In short ORB is a fusion of the FAST keypoint detector, following the procedure described in section 3.1.1, and the BRIEF descriptor (Calonder *et al.*, 2010) with some modifications to enhance the performance. For the FAST algorithm robust features are selected using either FAST or Harris response, while using an image pyramid to produce multiscale-features. An image pyramid is a multi-scale representation of a single image at different resolution. As FAST isn't orientation invariant ORB adds this ability by using first-order moments. The measure of corner orientation is determined by computing angle between the intensity weighted centroid and the center of the corner. To determine center of mass for the patch, that is, the centroid C, the moments of a patch first have to be defined as

$$m_{pq} = \sum_{u,v} u^p v^q \mathcal{I}(u,v), \tag{3.2.1.1}$$

where $\mathcal{I}(u,v)$ expresses pixel intensities at a image coordinate, (u,v). p and q furthermore express the order of the moments as an analogue of the mechanical moments. The centroid C is then given by

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right). \tag{3.2.1.2}$$

By constructing a vector between the centroid and the center of the corner, the orientation of the patch will be given by

$$\alpha = \text{atan2}(m_{01}, m_{10}). \tag{3.2.1.3}$$

When the orientation of the patch is calculated, it can be rotated to a canonical rotation, obtaining some rotation invariance. Now, a modified version of BRIEF can be applied to obtain a description of the keypoints. The features are converted into a binary feature vector so that they together represent an object. By using a Gaussian kernel the image is smoothed, making the binary descriptors insensitive to high-frequency noise. Finally, by using the patch orientations the binary tests produced by BRIEF are rotated such that the descriptions are rotation invariant.

3.3 Pyramidal Lucas-Kanade Feature Tracker

The optical flow feature tracker by Lucas-Kanade (Bouguet *et al.*, 2001) tries to find point correspondences between two grayscale images $\mathcal{I}_a$ and $\mathcal{I}_b$. Consider a pixel $\boldsymbol{u}_a = [u, v]^T$ in the first image $\mathcal{I}_a$. The algorithm tries to find the location of $\boldsymbol{u}_b = \boldsymbol{u}_a + \boldsymbol{d} = [u + d_u, v + d_v]^T$ in the second image $\mathcal{I}_b$, such that $\mathcal{I}_a(\boldsymbol{u}_a)$ and $\mathcal{I}_b(\boldsymbol{u}_b)$ are similar. The vector $\boldsymbol{d} = [d_u, d_v]^T$ describes the image velocity and is known as an optical flow vector. The optical flow algorithm tries to find the vector solution for $\boldsymbol{d}$ that minimizes the residual function ϵ defined as:

$$\epsilon(\boldsymbol{d}) = \epsilon(d_u, d_v) = \sum_{x=u-w_u}^{u+w_u} \sum_{y=v-w_v}^{v+w_v} (\mathcal{I}_a(x, y) - \mathcal{I}_b(x + d_u, y + d_v))^2 \tag{3.3.0.1}$$

The similarity function is evaluated over an integration window in an image neighbourhood of size $(2w_v + 1) \times (2w_v + 1)$. Notice that a smaller integration window will provide a higher local accuracy, while a larger window would be more robust to larger displacement following from higher velocity or a lower frame rate. Therefore it follows a natural trade off in this.

A solution to this problem is the pyramidal implementation of the classical iterative Lucas-Kanade algorithm. The idea of the pyramidal implementation is illustrated in figure 3.3.1.

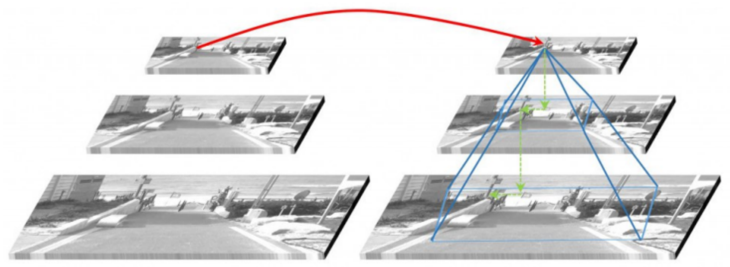


Figure 3.3.1: Image pyramid of the optical flow algorithm. The image at top of the pyramid describes the image with the lowest resolution, which estimates the initial optical flow. The estimate is refined for every level down to the original image.

For each image the pyramidal implementation recursively smoothes and down-samples the original image, such that the input image can be evaluated at different scales. The recursive form computes an initial estimate for optical flow vector, $\boldsymbol{d}$, for the image at the lowest scale. This estimate serves as an initial guess when the track is computed the higher resolution images. Thus, the optical flow vector is continuously refined through the higher resolution levels down to the original image as shown in image 3.3.1. Going down the pyramid small motions are removed and large motions becomes small motions, thus returning an optical flow that is more robust to larger displacements.

CHAPTER 4

Fundamentals of statistics

A presentation of the necessary background material related to statistical inference was included in the specialisation project preceding this thesis. This presentation is deemed valuable also for this thesis, and the presentation from the project report (Hellum, 2020) is therefore included below in a redrafted version.

This chapter covers fundamental statistics that is essential to understand the SLAM problem formulation. First, the multivariate gaussian describes the uncertainty model that is used for measurements. Then, MAP optimization describes how variable assignment may be refined to find the most likely outcome from a set of condtitional states.

4.1 Multivariate Gaussian Distribution

The multivariate Gaussian generalizes the univariate Gaussian, and is one of the key constructions that underlies SLAM and virtually all of sensor fusion. The multivariate Gaussian distribution is given by its expectation vector, $\boldsymbol{\mu}$, and a symmetric positive definite covariance matrix, $\boldsymbol{\Sigma}$. The univariate Gaussian is distributed according to a bell curve with the peak located at the expectation value and the spread given by the covariance. The distribution follows the same logic when this is extended to multiple dimensions, and is mathematically described according to

$$\begin{aligned}\mathcal{N}(\boldsymbol{\xi};\boldsymbol{\mu},\boldsymbol{\Sigma}) &= \frac{1}{(2\pi)^{\frac{n}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\boldsymbol{\xi}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\xi}-\boldsymbol{\mu})\right) && (4.1.0.1a)\\ &= \exp\left(a+\boldsymbol{\eta}^T\boldsymbol{\xi}-\frac{1}{2}\boldsymbol{\xi}^T\boldsymbol{\Lambda}\boldsymbol{\xi}\right). && (4.1.0.1b)\end{aligned}$$

Here, equation 4.1.0.1b describes the canonical form of a multivariate gaussian Brekke, 2020. The canonical form is parametrized using the information matrix, $\boldsymbol{\Lambda}$, which is the inverse covariance matrix, and the information vector, $\boldsymbol{\eta}$. The parameters of the canonical form is distributed as

$$\begin{aligned}\boldsymbol{\Lambda} &= \boldsymbol{\Sigma}^{-1}\\ \boldsymbol{\eta} &= \boldsymbol{\Lambda}\boldsymbol{\mu}\\ a &= -\frac{1}{2}n\ln(2\pi)-\ln|\boldsymbol{\Lambda}|+\boldsymbol{\eta}^T\boldsymbol{\Lambda}\boldsymbol{\eta}\end{aligned} \qquad (4.1.0.2)$$

The canonical form is widely used in graph based SLAM, because the canonical form is able preserve a factor-graph structure that will be explained in section 5.2.1. Another advantage is that the information matrix remains sparser than the corresponding covariance form (Dellaert and Kaess, 2017) when the number of variables grows. However, the problem still grows, which makes memory usage and computation grow unbounded in time. One way to handle this is to remove older variables without removing information. This process is called marginalization.

Stated in terms of probability densities, given a joint density $p(\xi, \gamma)$ for two variables ξ and γ, then marginalizing out the variable ξ corresponds to integrating over ξ, i.e.,

$$p(\gamma) = \int_x p(\xi, \gamma) d\xi \tag{4.1.0.3}$$

If the two variables ξ and γ are expressed on the canonical form, their joint distribution can be partitioned according to

$$p(\xi, \gamma) = \mathcal{N}\left(\Lambda^{-1}\begin{bmatrix} \eta_\xi \\ \eta_\gamma \end{bmatrix}, \begin{bmatrix} \Lambda_{\xi\xi} & \Lambda_{\xi\gamma} \\ \Lambda_{\xi\gamma}^\top & \Lambda_{\gamma\gamma} \end{bmatrix}^{-1}\right) \tag{4.1.0.4}$$

The information matrix of y after marginalization can then be obtained by taking the *Schur complement* of $\Lambda_{\xi\xi}$, i.e. $\Lambda_{\gamma\gamma} - \Lambda_{\xi\gamma}^\top \Lambda_{\xi\xi}^{-1} \Lambda_{\xi\gamma}$ (Brekke, 2020).

4.2 Maximum a Posteriori Optimization

A Maximum a Posteriori (MAP) estimator finds an the most likely value of a state, ξ_i, given the mode of its posterior distribution. This mode will be the one that yield the most likely value based conditionally on a prior distribution. The MAP estimator thus provides a powerful tool for nonlinear state estimation. This is valueable for SLAM systems, where the true states of the system are unknown, and conditional on a set of measurements. In order to represent the uncertainty of the measurements, multivariate Gaussians are normally used, yielding $z_i = h_i(\xi_i) + \eta_i \sim \mathcal{N}(\mu, \sigma_i^2)$. Here $h_i(\xi_i)$ describes the measurement process of all quantities, ξ_i, that contribute to the measurement result, z_i. The associated noise, η_i, is often assumed to be zero-mean Gaussian, so that the measurement error can be expressed as

$$e_i(\xi_i) = h_i(\xi_i) - z_i \tag{4.2.0.1}$$

From the expression one can see that the error function expresses the state, x_i, as a conditional of the measurement, z_i. For SLAM problems, the objective function is typically set to be the squared Mahalanobis distance (Mahalanobis, 1936), or the Huber Norm (Huber, 1992) of all the errors. Using the Mahalanobis distance, this yields the MAP problem formulation

$$\boldsymbol{\xi}^* = \arg\min_{\boldsymbol{\xi}} g(\boldsymbol{\xi}) = \arg\min_{\boldsymbol{\xi}} \sum_i \|e_i\|_{\Sigma_i}^2 \tag{4.2.0.2}$$

The nonlinear problem, $g(\boldsymbol{\xi})$, is then solved by applying nonlinear optimization methods such as Gauss-Newton, Levenberg-Marquardt, or Powell's Dogleg (Nocedal *et al.*, 2006). These solvers repeatedly perform perturbations on a succession of linear approximations to equation 5.2.1.4 in order to approach a minimum. As the number of states grow, the problem grows unbounded. This may be solved efficiently by graph optimization methods, which we will study in greater detail in section 5.2.

CHAPTER 5

Simultanous Localization and Mapping (SLAM)

In this chapter relevant background information related to SLAM and factor graph optimization will be presented. First, the structure of the SLAM problem formulation is detailed. Next, graph structures and the incremental update and optimization approach iSAM2 is described. Lastly, followed by a concurrent approach for short-term and long-term smoothing is explained.

5.1 The SLAM Problem Formulation

Visual odometry is the process of mapping an environment and estimating the motion of which a robot travels within this local map, typically by using exteroceptive sensors such as LiDAR, EO- or IR-camera. Customary, VO systems are divided into two main components: the front end and the back end. The front end associates structures in the image/frame, either in terms of extracted keypoints, or pixel intensities over the image directly. The back end performs inference on this data to find the relative poses that yields the most likely association of the aforementioned structures. The keypoints may also be included in this refinement. These pose estimates altogether forms the robot's odometry. Additional measurements from IMU, GNSS, etc. may be included to improve this estimate. An extension of VO is SLAM. Visual odometry treats the whole world as an infinite corridor, but SLAM enables place recognition so that the true topology of the environment can be restored. The accumulated drift are thus corrected within the loop-enclosed area. The problem formulation of VO and SLAM is commonly divided into pose estimation and structure estimation. It is common to formulate pose estimation problems as either direct or indirect. The latter is also referred to as feature-based. Whereas structure estimation problems is categorized as either dense or sparse. Sparse methods only use a carefully selected subset of the information in a scene, while dense methods attempt to reconstruct the entire scene. Direct and indirect methods are discussed in the following subsections.

5.1.1 Indirect Methods

Indirect methods start by extracting a geometric representation of the scene. This typically includes feature extraction and association of the keypoints through optical flow feature tracking or descriptor matching. Indirect methods are typically referred to as feature-based methods because of this intermediate features extraction step. The feature observations can then be used to estimate the relative motion based on the point correspondences. It is common to estimate the motion of the body by project 3D points in the world, $\boldsymbol{l}^w$, onto the image plane using eq. (2.4.1.6) and comparing the reprojected point position to the original corresponding feature position, $\boldsymbol{u}$. This is referred to the reprojection error, which minimizes the geometric error in eq. (5.1.1.1)

$$e_{geometric}(\boldsymbol{T}_{\text{wc}}, \boldsymbol{l}^{\text{w}}) = \pi_p(\boldsymbol{l}^w; \boldsymbol{T}_{\text{wc}}^{-1}, \boldsymbol{K}) - \boldsymbol{u} \tag{5.1.1.1}$$

The world points are related to the camera pose by the initial transformation estimate from the world frame $\mathbf{T}_{wc}$. The reprojection error is calculated for multiple 3D-2D correspondences and the overall error is minimized using equation 4.2.0.2. This is called BA. If the scale of the 3D points are known, then the actual scale of the motion can be retrieved. The optimization may be performed by only refining an initial pose estimate, $\mathbf{T}_{wc}$. This is called motion-only bundle adjustment. Another alternative is structure-only bundle adjustment, which rearranges the world points $\boldsymbol{l}^w$ to minimize the error. The last alternative is full bundle adjustment, which both optimize the poses and the world points.

There are typically two approaches to keypoint matching before bundle adjustment, either by sequential frames over a window, or by mutual observability constraints following repojection from a local map. Respectively, this is exampled by SVO2 (Forster, Z. Zhang, *et al.*, 2016) and ORB-SLAM (Mur-Artal, Montiel, *et al.*, 2015), where ORB-SLAM uses a combination of the two. Windowed approaches are often faster, while projecting co-visible points from a local map can provide a slight boost in performance because more potential correspondences are mapped from the entire trajectory.

Indirect methods provide robustness to photometric and geometric distortions. However, a disadvantage of indirect methods is that extraction and matching of features in low textured and poorly illuminated environments is difficult. Direct methods partially tackle this problem.

5.1.2 Direct Methods

Direct methods skip the feature extraction step that is considered for indirect methods. Indirect methods rather optimize directly on pixel intensities. This problem formulation is the same as minimization equation 4.2.0.2 over the photometric error in equation 5.1.2.1

$$e_{photometric}(\boldsymbol{u}^b, z^b, \boldsymbol{T}_{ab}) = \mathcal{I}_a(w(\boldsymbol{u}^b, z^b, \boldsymbol{T}_{ab})) - \mathcal{I}_b(\boldsymbol{u}^b) \tag{5.1.2.1}$$

where the warp function

$$\boldsymbol{u}^a = w(\boldsymbol{u}^b, z^b, \boldsymbol{T}_{ab}) = \pi_p(\boldsymbol{T}_{ab} \cdot \pi_p^{-1}(\boldsymbol{u}^b, z^b)) \tag{5.1.2.2}$$

maps pixels $\boldsymbol{u}^b$ from image $\mathcal{I}_b$ to pixels $\boldsymbol{u}^a$ in image $\mathcal{I}_a$. The transformation $\mathbf{T}_{ab}$ describes a motion from camera pose a to pose b, while z describes the depth for the inverse projection.

Well known examples of direct SLAM methods are LSD-SLAM by Engel, Schöps, *et al.* (2014), SVO by Forster, Z. Zhang, *et al.* (2016), and DSO Engel, Koltun, *et al.* (2017). This work shows that direct methods are more robust to blurred images and areas that are sparsely textured since these methods use the image intensity values directly to optimise a photometric error, and are not dependent on distinct features such as indirect methods. Direct methods may also provide a denser map reconstruction compared to feature-based methods. There are however some downsides to these methods as they require photometrically calibrated images and are more vulnerable to geometric distortions that can originate from bad calibration or rolling shutter effects.

5.2 Optimization over Graph Structures

At their core SLAM, and many other estimation problems, are searching for a MAP estimate, i.e., trying to maximize the posterior probability of some variables, given a set of measurements. This operation is typically performed in the part of the SLAM system referred to as the *backend*, where initial estimates from the frontend or other sensor data from IMUs, GNSS', etc. are included. When attempting to act optimally, a performance index is maximized, or conversely a penalty function is minimized.

The optimized terms of these problem formulations are usually local in nature, meaning that the terms only depend on a minor subset of the entire set of variables. A flexible and intuitive way of modelling this locality structure is using the concept of factor graphs. Factor graphs are a class of graphical models composed of variable and factor nodes. Variables represent unknown quantities to be estimated. Dependencies between variables are indicated by connecting edges, where factors represent functions on subsets of the variables for each connecting edge. Aside from the insightful modelling benefit, they are efficiently solved and provide an easily modular interface.

In the next subsections different graph structures that are associated to factor graphs and are of specific importance to the Incremental Smoothing and Mapping 2 (iSAM2) algorithm (section 5.3) are described. Details on how the graph structures are related and the act of performing inference on the graph structures are also explained.

A presentation of the necessary background material related to the graph structures in section 5.2.1, section 5.2.2 and section 5.2.3 was included in the specialisation project preceding this thesis. This presentation is deemed valuable also for this thesis, and the presentation from the project report (Hellum, 2020) is therefore included below in a redrafted version for the next subsections.

5.2.1 Factor Graphs

Factor graphs have become the de facto standard for formulating SLAM problems (Cadena *et al.*, 2016). One of the reasons is that the graph can be factorized if the measurement errors are assumed statistically independent. New measurements often only have a local effect on the graph, which enables efficient update algorithms to only evaluate the affected parts of the graph. iSAM2, described in section 5.3, is an example of an algorithm that efficiently utilizes factor graph structure.

An example of a factor graph containing poses, $\boldsymbol{x}$, and landmarks, $\boldsymbol{l}$, is depicted in fig. 5.2.1. In addition to being capable of embedding poses and landmarks for state estimation are factor graphs also suitable for sensor fusion, where measurements from other sensors can be added as separate nodes to the factor graph. Embedding new sensors in the optimization will not greatly increase the complexity of the problem, as all measurements are considered independent. It will, however, make it easier for the optimization to converge to the globally correct solution rather than a local maximum. The IMU is a good example on how other sensors can provide additional information on for example the velocities in addition to the poses. IMU measurements can be inserted as connecting factors between two variable nodes. GNSS is another sensor that easily can be fused into a factor graph, adding a prior of the pose nodes. Details on practical examples of how sensors can be added to the graph is depicted in fig. 8.0.2 and will be discussed in more detail in section chapter 8. Minimizing the MAP objective function formulated over the factor graph, with multiple sensors enables, elegantly performs sensor fusion. Another variant to the connecting factors from odometry measurements is the loop-closing constraints in figure 5.2.1, which zero out drift for a revisited location. In order of performing all of these operations, the a general problem formulation for factor graph statistical inference have to be defined.

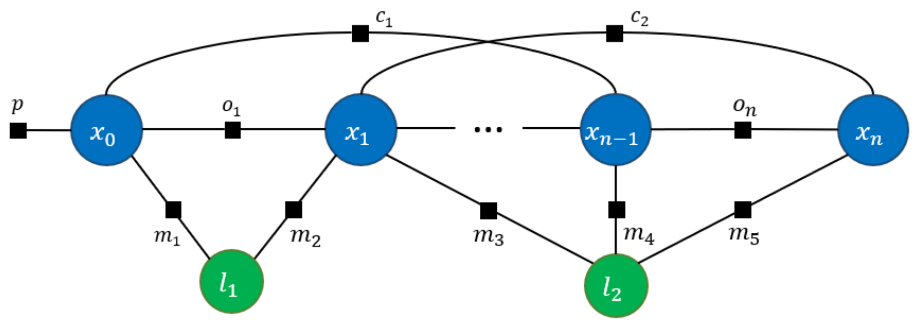


Figure 5.2.1: Factor graph (Kschischang *et al.*, 2001) formulation of a basic SLAM problem, only composed of poses and landmarks. Variable nodes are shown as large coloured circles, and factor nodes (measurements) as small solid squares. The factors shown are odometry measurements o, a prior p, loop-closing constraints c and landmark measurements m. Inspired by fig. 2 from Kaess, Johannsson, Roberts, Ila, J. J. Leonard, *et al.* (2012).

A factor graph is an bipartite probabilistic graphical model that consists of two types of nodes: variable nodes and factor nodes. Variable nodes, denoted $\mathcal{X}_j \in \mathcal{X}$, represent the states that are estimated. For SLAM problems these states are typically the poses and landmarks $\mathcal{X} = \{\mathbf{x}, \mathbf{l}\}$ respectively, as shown in figure 5.2.1. Additional variable states such as velocities and biases are examples of potential extensions to the problem formulation. Factor nodes, denoted $f_i \in f$, commonly represent measurements that connect the conditional probabilities between states. Each factor f_i is defined as a function of the set of its adjacent variables nodes $\mathcal{X}_i$. Edges e_{ij} are always between factor nodes and variable nodes.

There are two main inference problems for factor graphs, computing marginals, and computing the mode. Computing marginals is the process of removing older variables without removing their information, such that computational complexity is reduced. This process is applied during the smoothing process of SLAM problems. The update step boils down to computing the mode, which is estimating the most likely variable assignment over a chosen set of states. A common choice for this estimation is the maximum a posteriori (MAP) estimator, explained in section 4.2, which maximizes the posterior density of the variables with respect to the factors

$$\mathcal{X}^* = \arg\max_{\mathcal{X}} f(\mathcal{X}) \tag{5.2.1.1}$$

where $f(\mathcal{X})$ is the factorization of the factor graph

$$f(\mathcal{X}) = \prod_i f_i(\mathcal{X}_i). \tag{5.2.1.2}$$

For SLAM problems it is common to assume Gaussian measurement models, as described in section 4.1. This equals

$$f_i(\mathcal{X}_i) \propto exp(-\frac{1}{2}\left\|h_i(\mathcal{X}_i) - z_i\right\|^2_{\mathbf{\Sigma}_i}). \tag{5.2.1.3}$$

The estimated posterior from equation 5.2.1.1 is rewritten as to the nonlinear least-squares objective function

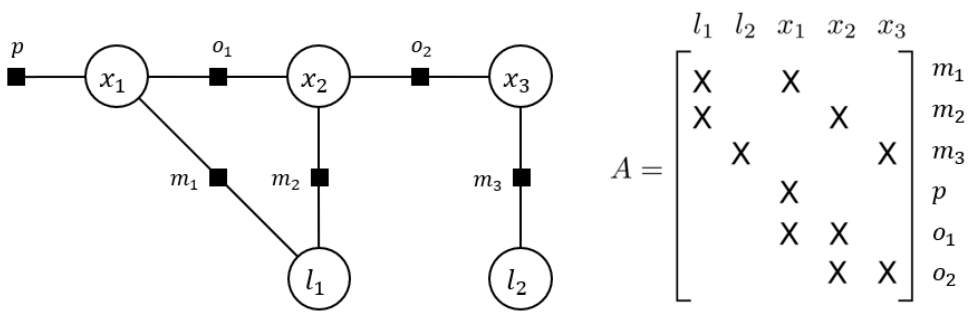
$$\mathcal{X}^* = \arg\min_{\mathcal{X}}(-\log f(\mathcal{X})) = \arg\min_{\mathcal{X}}(\frac{1}{2}\sum_i \left\|h_i(\mathcal{X}_i) - z_i\right\|^2_{\mathbf{\Sigma}_i}) = \arg\min_{\mathcal{X}}\sum_i \left\|\mathbf{e}_i\right\|^2_{\mathbf{\Sigma}_i} \tag{5.2.1.4}$$

where $h_i(\mathcal{X}_i)$ is a measurement function and z_i is a measurement. Furthermore, $\|\boldsymbol{e}\|^2_{\mathbf{\Sigma}} = \boldsymbol{e}^T\mathbf{\Sigma}^{-1}\boldsymbol{e}$ is the squared Mahalanobis distance which weights the residual errors, $\boldsymbol{e}$, according to the covariance $\mathbf{\Sigma}$ of the respective measurement, z_i.

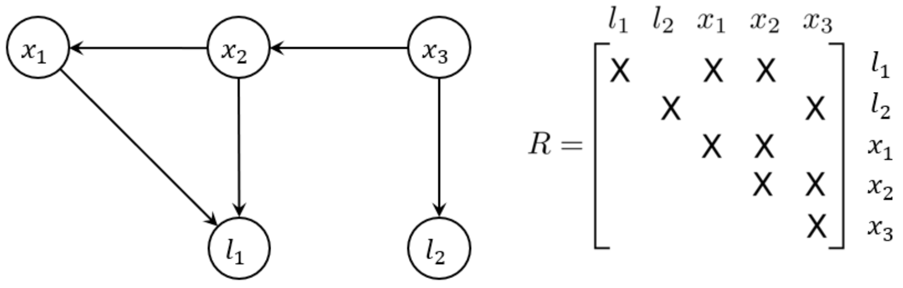
The MAP estimate is now in a form that can be solved by using nonlinear optimization methods such as Gauss-Newton, Levenberg-Marquardt, or Powell's Dogleg. These solvers repeatedly perform perturbations on a succession of linear approximations to equation 5.2.1.4 in order to approach a minimum. Therefore, one typically approximates equation 4.1.0.1 using a Taylor expansion. With some rewriting this becomes

$$-\log f(\mathbf{\Delta}) = \sum_i \frac{1}{2}\left\|\boldsymbol{A_i}\Delta_i - b_i\right\|^2 \tag{5.2.1.5}$$

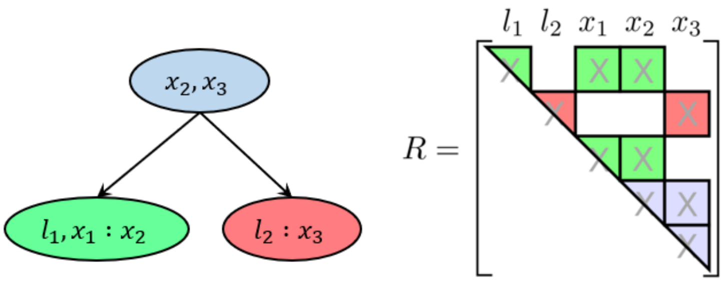
where $A_i = \Sigma_i^{-1/2}J_i$ is the weighted measurement Jacobian, $\boldsymbol{J}$, Δ_i is the linearized states and $b_i = \Sigma_i^{-1/2}(z_i - h_i(\Delta_i))$ is the weighted prediction error. In SLAM problems updates arrive in an incremental order which allows for more efficient solutions; One of which transforms the factor graph into a Bayes net and further into the Bayes tree as illustrated in figure 5.2.2. Thus, inference on the factor graph can be understood as converting the factor graph onto the form of these related graph structures and computing the joint density over these structures instead. These graph structures are covered in the following sections.



(a) Factor graph and the associated Jacobian matrix **A**. Xs mark connections from variable to factor nodes.



(b) Bayesian network and the associated square root information matrix **R**. Xs mark connections between variable nodes.



(c) Bayes tree and the associated square root information matrix **R**. Colors are used to describe cliques represented both in terms of the bayes tree structure and the matrix equivalent.

Figure 5.2.2: The graph structures used in the iSAM2 algorithm. The same naming convention as for fig. 5.2.1 is used. All figures are inspired by fig. 3 from (Kaess, Johannsson, Roberts, Ila, J. J. Leonard, *et al.*, 2012), but modified to fit conventions for this thesis.

5.2.2 Bayesian network

Bayesian networks, also called Bayes nets, are directed acyclic graphs (DAGs) that aim to model probabilistic causality through conditional dependence. In figure 5.2.2b the nodes, $\mathcal{X}_j$, represent variables while each edge represent a conditional dependency. Furthermore, Bayesian networks satisfy the local Markov property (Rabiner, 1989), which states that a variable is only conditionally dependent on its prior state. Here, prior state refers to previous poses or associated landmarks. In other words, if there exist an edge between nodes $\mathcal{X}_j$ and $\mathcal{X}_{j-1}$ their conditional dependence $f(\mathcal{X}_j|\mathcal{X}_{j-1})$, is a factor in the joint distribution over the network. A variable $\mathcal{X}_j$ may have multiple priors, thus forming a set of variables $\boldsymbol{S}_j$ that the node is conditional dependent on. Bayesian inference may be applied to analyze the joint distribution over the entire set of random variables

$$f(\mathcal{X}) \triangleq \prod_j f(\mathcal{X}_j|\boldsymbol{S}_j). \tag{5.2.2.1}$$

The Bayes net can be obtained from a factor graph by QR- or Cholesky factorization, or equivalently by means of a bipartite elimination game, as described by (Heggernes *et al.*, 1996). For iSAM2 (section 5.3) the latter approach is taken, which proceeds by eliminating one variable $\mathcal{X}_j$ at a time, and follows algorithm 1 in converting it into a node of the Bayes net. The nodes represent conditionals $P(\mathcal{X}_j|\boldsymbol{S}_j)$ for every factor node on its connected variables.

```
Algorithm 1: Eliminating a variable $\mathcal{X}_j$ from the factor graph.
1 Remove from the factor graph all factors $f_i(\mathcal{X}_i)$ that are adjacent to $\mathcal{X}_j$. Define the
    separator $\boldsymbol{S}_j$ as all variables involved in those factors, excluding $\mathcal{X}_j$.
2 Form the (unnormalized) joint density $f(\mathcal{X}_j, \boldsymbol{S}_j) = \prod_i f_i(\mathcal{X}_j)$ as the product of those
    factors.
3 Using the chain rule, factorize the joint density $f(\mathcal{X}_j, \boldsymbol{S}_j) = P(\mathcal{X}_j|\boldsymbol{S}_j)f(\boldsymbol{S}_j)$. Add the
    conditional $P(\mathcal{X}_j|\boldsymbol{S}_j)$ to the Bayes net and the factor $f(\boldsymbol{S}_j)$ back into the factor graph.
```

An example of a factor graph converted to a Bayes net is shown in figure 5.2.2b. The chain rule $f(\mathcal{X}_j, \boldsymbol{S}_j) = P(\mathcal{X}_j|\boldsymbol{S}_j)f_{new}(\boldsymbol{S}_j)$ in step 3 of the elimination algorithm can be implemented using Householder reflections or Gram-Schmidt orthogonalization. Every incoming factor-product is on the form

$$f(\Delta_j, \mathbf{s}_j) \propto \exp\left(-\frac{1}{2}\left\|\boldsymbol{a}\Delta_j + \boldsymbol{A}_S\mathbf{s}_j - \boldsymbol{b}\right\|^2\right) \tag{5.2.2.2}$$

where $\mathbf{A}_j = [a|\mathbf{A}_S]$ is obtained by concatenating all variables $\boldsymbol{s}_j$ that are connected to the variable that is currently undergoing conversion from factor graph to Bayes net, Δ_j. The factor product can be rewritten according to (Kaess, Ila, *et al.*, 2010) by marginalization of $\boldsymbol{s}_j$ as described by Brekke (2020). Equation 5.2.2.2 is rewritten as

$$f(\Delta_j, \mathbf{s}_j) = P(\Delta_j|\mathbf{s}_j)f(\mathbf{s}_j) \tag{5.2.2.3}$$

where

$$P(\Delta_j|\mathbf{s}_j) \propto \exp\left\{-\frac{1}{2}(\Delta_j + \mathbf{r}\mathbf{s}_j - d)^2\right\} \qquad f(\mathbf{s}_j) \propto \exp\left(-\frac{1}{2}\left\|\boldsymbol{A}^{'}\mathbf{s}_j - \boldsymbol{b}^{'}\right\|^2\right) \tag{5.2.2.4}$$

and

$$\mathbf{r} = (\mathbf{a}^T\mathbf{a})^{-1}\mathbf{a}^T\mathbf{A}_S \qquad d = (\mathbf{a}^T\mathbf{a})^{-1}\mathbf{a}^T\mathbf{b} \qquad \mathbf{A}^{'} = \mathbf{A}_S - \mathbf{a}\mathbf{r} \qquad \mathbf{b}^{'} = \mathbf{b} - \mathbf{a}d$$

The new factor $f(\mathbf{s}_j)$ in equation 5.2.2.3 is obtained by substituting $\Delta_j = d - \boldsymbol{r}\boldsymbol{s}_j$ into 5.2.2.2. This yields one iteration of Gram-Schmidt. Thus, the MAP estimate of $\boldsymbol{\Delta}$ is found by recursively solving $\Delta_j = d - \mathbf{r}\mathbf{s}_j$ for every variable Δ_j.

The Bayes net resulting from algorithm 1 holds the important property of being chordal. A chordal graph means that any undirected cycle of length greater than three has an additional edge that is not part of the cycle but connects two of the vertices. This is a necessary property if the Bayes net is to be further converted into a Bayes tree, as will be discussed in next.

5.2.3 Bayes tree

Marginalization and optimization of Bayes nets is not easy in general. However, the related tree-based structure, the Bayes tree, enables new recursive algorithms that simplify and streamlines this process. A Bayes tree is a directed tree where the nodes represent cliques. This structure was introduced with the iSAM2 algorithm (Kaess, Johannsson, Roberts, Ila, J. J. Leonard, *et al.*, 2012) described in section 5.3. If the underlying Bayes net is chordal then the Bayes tree can be constructed by discovering its cliques using the maximum cardinality search algorithm (Tarjan *et al.*, 1984). Each clique consists of its separators $\boldsymbol{S}_k$ being the intersection $\mathcal{C}_k \cap \Pi_k$ of the current clique and its parent respectively, and frontal variables $\boldsymbol{F}_k$ containing the remaining variables. An example of a Bayes tree is shown in figure 5.2.2c, where the frontal elements of the left leaf node is $\{l_1, x_1\}$ and x_2 is the separator. This restructuring of the graph yields the joint distribution from equation 5.2.2.1 to be rewritten for the Bayes tree accordingly

$$P(\mathcal{X}) \triangleq \prod_k P(\boldsymbol{F}_k|\boldsymbol{S}_k). \tag{5.2.3.1}$$

From section 5.2.2 it is clear that every node of the Bayes net represent the conditional $P(\mathcal{X}_j|\boldsymbol{S}_j)$. The cliques of the Bayes tree therefore contains as set of these nodes of size greater than one as shown in figure 5.2.2c. The optimal assignment of $\mathcal{X}^*$ can be computed using algorithm 4.

New measurements often only have a local effect on the Bayes tree, which enables efficient update algorithms to only evaluate the affected branches of the tree. The update rule of the Bayes tree is described in algorithm 2 and illustrated in figure 5.2.3. From this update rule it becomes clearer why iSAM2 uses the elimination game from algorithm 1 over factorization. Due to the local update it is efficient to only perform partial state update over the affected nodes using algorithm 4.

Algorithm 2: Updating the Bayes tree inclusive of fluid relinearization (section 5.3.3) by recalculating all affected cliques.

```
Input: Bayes tree $\mathcal{T}$, nonlinear factors $f$, affected variables $\mathcal{X}'$
Output: modified Bayes tree $\mathcal{T}'$
1 Remove top of Bayes tree, convert to a factor graph:
      (a) For each affected variable in $\mathcal{X}'$ remove the corresponding clique and all parents
          up to the root.
      (b) Store orphaned sub-trees $\mathcal{T}_{orph}$ of removed cliques
2 Relinearize all factors required to recreate top.
3 Add cached linear factors from orphans $\mathcal{T}_{orph}$.
4 Re-order variables of factor graph.
5 Eliminate the factor graph and create a new Bayes tree.
6 Insert the orphans $\mathcal{T}_{orph}$ back into the new Bayes tree.
```

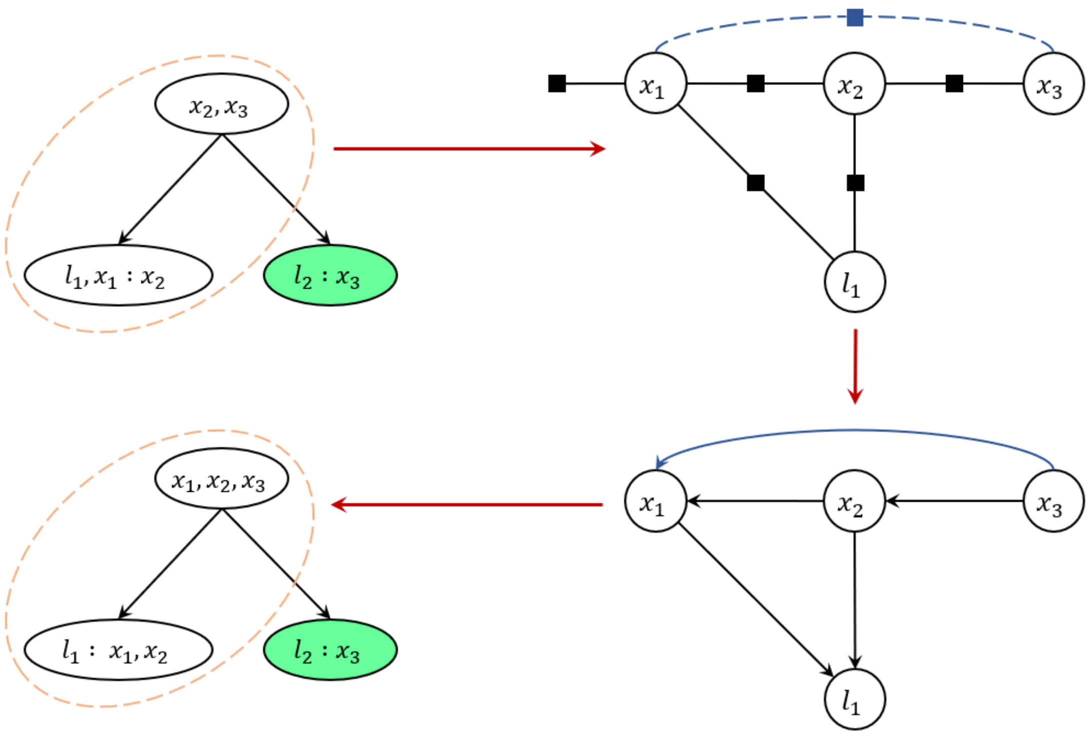


Figure 5.2.3: The image depicts the Bayes tree update sequence described in algorithm 2. Based on example from fig. 5.2.2, the nodes that are affected by the update (encircled by a dotted orange oval) are first converted to a factor graph. A new factor is inserted in the extracted graph, between pose variables x_1 and x_3 (dotted blue). The factor graph nodes are then eliminated using algorithm 1 to obtain the Bayes net representation. Lastly are the updated Bayes tree created from the chordal Bayes net added back into the original tree. Consequently, the right "orphan" sub-tree (light green) from the original Bayes tree remains untouched, while the affected section is updated. The figure is inspired by fig. 3 from (Kaess, Johannsson, Roberts, Ila, J. J. Leonard, *et al.*, 2012), but modified to fit the notation used in this thesis.

5.3 Incremental Smoothing and Mapping 2 (ISAM2)

A presentation of the necessary background material related to iSAM2 was included in the specialisation project preceding this thesis. This presentation is deemed valuable also for this thesis, and the presentation from the project report (Hellum, 2020) is therefore included below in a redrafted version.

Many inference problems, such as SLAM, are incremental by nature as new measurements arrive sequentially. Naturally the most efficient solution to a sequential problem is an incremental algorithm. For SLAM problems the current state is dependent on prior knowledge and reuse is therefore possible, which allows for more efficient solutions. The iSAM algorithms are examples of this.

The original iSAM algorithm follows the derivation described in section 5.2.1 and minimizes equation 5.2.1.4 by applying fast linear solvers. This allows iSAM1 to repeatedly solve the square root information matrix, $\boldsymbol{R}$ by backsubstitution. Updates are performed using Givens rotation (Gentleman, 1973) to maintain the efficient upper triangular structure of the $\boldsymbol{R}$ matrix. However, as new measurements are added the square root information matrix will gradually drift further away from the true state. iSAM1 solves this problem by periodically execute re-linearization and variable reordering at batch steps. This is a sub-optimal solution as refactoring the whole matrix is expensive.

iSAM2 is a fully incremental, graph-based version of the iSAM algorithm. iSAM2 avoids the whole issue of periodic batch refraction by introducing the Bayes tree and then utilizes incremental reordering, partial state updates, and fluid re-linearization. The Bayes tree is discussed in section 5.2.3, while the remaining topics are covered in the following subsections. By introducing these topics iSAM2 combines the advantages of the graphical model and sparse linear algebra to obtain one of the fastest full graph-SLAM methods used today. The iSAM2 algorithm is captured in algorithm 3.

Algorithm 3: One step of the iSAM2 algorithm, following the general structure of a smoothing solution.

```
Input: New nonlinear factors $f^{'}$, new variables $\mathcal{X}'$
Output: Bayes tree $\mathcal{T}$, nonlinear factors $f$, linearization point $\mathcal{X}$, update $\mathbf{\Delta}$
Initialization: $\mathcal{T} = \emptyset$, $f = \emptyset$, $\mathcal{X} = \emptyset$
1 Add any new factors $f := f \cup f^{'}$.
2 Initialize any new variables $\mathcal{X}'$ and add $\mathcal{X} := \mathcal{X} \cup \mathcal{X}'$.
3 Fluid relinearization with Alg. 5 yields affected variables.
4 Redo top of Bayes tree with Alg, 2.
5 Solve for delta $\mathbf{\Delta}$ with Alg. 4.
6 Current estimate given by $\mathcal{X} \oplus \mathbf{\Delta}$.
```

From algorithm 3 it can be seen that iSAM2 use several additional strategies to solve specific sub tasks, all of which are addressed in the next subsections.

5.3.1 Incremental Variable Ordering

It is important to choose a good variable ordering to efficiently find the sparse matrix solution. An optimal ordering minimizes the fill-in, where fill-in can be seen as the size of the cliques. Finding the variable ordering that leads to the minimum fill-in is NP-hard. Heuristic methods such as Column Approximate Minimum Degree (COLAMD) (Davis *et al.*, 2004) provide close to optimal approximations of the ordering for batch problems. COLAMD was used in the original iSAM algorithm, but an incremental variable reordering strategy is desirable for the iSAM2 algorithm to allow for faster updates in subsequent steps. This is achieved with the Constrained COLAMD (CCOLAMD), where the most recently accessed variables to the end of the ordering. This makes sense for the SLAM problems because a new set of measurements are expected to connect to the most recent observed states. It is therefore most likely that the most recent accessed variables are the ones that need reordering.

5.3.2 Partial State Updates

New measurements often only have a local effect of the Bayes tree, which is utilized by the partial state update. Instead of performing full backsubstitution on the square root information matrix, $\boldsymbol{R}$, iSAM2 only include variables that change when computing the update. More specifically, iSAM2 begins at the root of the tree and recursively updates all descendant cliques that change beyond a given threshold. The remaining nodes are marginalized. Except for large loop closures, this results in a less expensive update rule.

Algorithm 4: Partial state update: Solving the Bayes tree in the nonlinear case returns an update $\mathbf{\Delta}$ to the current linearization point $\mathcal{X}$.

Input: Bayes tree $\mathcal{T}$

Output: Update $\mathbf{\Delta}$

```
1 Starting from the root clique $\boldsymbol{\mathcal{C}}_r = \boldsymbol{F}_r$:
2     For current clique $\mathcal{C}_k = \boldsymbol{F}_k : \boldsymbol{S}_k$, compute update $\mathbf{\Delta}_k$ of frontal variables $\boldsymbol{F}_k$ using
        already computed values of parent $\boldsymbol{S}_k$ and the local conditional density $P(\boldsymbol{F}_k|\boldsymbol{S}_k)$.
3     For all variables $\Delta_{kj}$ in $\mathbf{\Delta}_k$ that change by more than threshold $\alpha$: recursively
        process each descendant containing such a variable.
```

5.3.3 Fluid Relinearization

Fluid relinearization is added to the iSAM2 algorithm so that relinearization only is performed when needed. In order of doing so, the algorithm keeps track of the validity of the linearization point for each variable. When a variable drift beyond a threshold all relevant information is removed from the Bayes tree and replaced by relinearizing the corresponding original nonlinear factor. For all cliques that are relinearized, the marginal factors from their sub-trees also have to be taken into account.

Algorithm 5: Fluid relinearization: The linearization points of select variables are updated based on the current delta $\mathbf{\Delta}$

Input: Linearization point $\mathcal{X}$, delta $\mathbf{\Delta}$

Output: Updated linearization point $\mathcal{X}$, marked cliques $\mathcal{M}$

```
1 Mark variables in $\mathbf{\Delta}$ above threshold $\beta$: $J = \{\Delta_j \in \mathbf{\Delta} | \Delta_j \geq \beta\}$.
2 Update linearization point for marked variables: $\mathcal{X}_j := \mathcal{X}_j \oplus \Delta_j$
3 Mark all cliques $\mathcal{M}$ that involve marked variables $\mathcal{X}_j$ and all their ancestors.
```

5.4 Concurrent Filtering and Smoothing

In contrast to filtering over a single state, even the iSAM2 smoothing solution discussed in section 5.3 generally is not a constant time operation over a growing number of states (Kaess, Johannsson, Roberts, Ila, J. J. Leonard, *et al.*, 2012). This is even more evident with occasional inclusion of loop closures where large amounts of factors have to be recalculated. Smoothing solutions can however be parallelized, allowing the problem formulation to be split into a high speed navigation component and a higher latency loop closure component. S. Williams *et al.* (2014) proposed an approach that combines short-term filtering and long-term smoothing within a single Bayes tree, while formulating it in such a way that both are performed concurrently. This approach enables the filter to operate at constant time where new sensor data are integrated real time, while updates from the slower smoother are integrated whenever once become available, while still achieving an optimal state estimate at any time. Such smoother updates typically refer to loop closures. While being named a filter, it is in practice rather a short-term smoother as will be detailed later. The concurrent structure is achieved by splitting the posterior factors in eq. (5.2.3.1) into three components

$$P(\mathcal{X} \mid \boldsymbol{Z}) = P\left(\mathcal{X}^R \mid \mathcal{X}^S\right) P\left(\mathcal{X}^S\right) P\left(\mathcal{X}^t \mid \mathcal{X}^S\right). \tag{5.4.0.1}$$

The factors from left to right represent the smoother, the separator and the filter. With this representation the smoothing over past states is decoupled from filtering on current states. The posterior $p(\mathcal{X} \mid \boldsymbol{Z})$ in eq. (5.4.0.1) is equivalent to a Bayes tree with the separator as root, as illustrated in fig. 5.4.1. By definition, the new factor are directly inserted into the filter clique $\mathcal{X}^t$. Changing a clique also affects all ancestors, which can be seen from the elimination algorithm (alg. 1), where information is passed upwards towards the root. Because the root is the separator clique $\mathcal{X}^S$, this also has to be recalculated. By the same argumentation, the smoother cliques $\mathcal{X}^R$ are eliminated independently up to the separator. Consequently, the smoother clique is unaffected by updates in the filter. The smoother and filter can therefore be solved in parallel, with the filter typically performing multiple steps during one smoother iteration, and then joined at the separator.

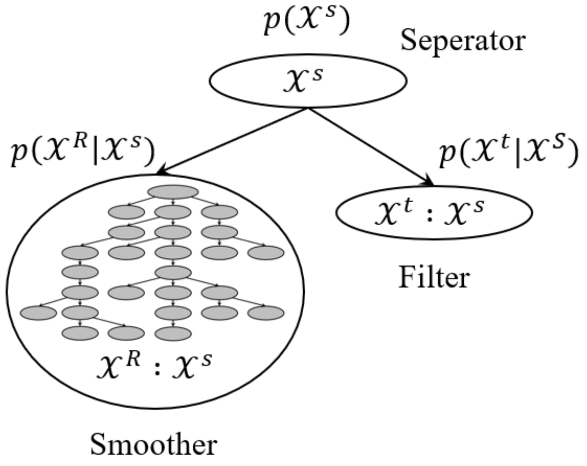

$p(\mathcal{X}^s)$
Seperator
$\mathcal{X}^s$
$p(\mathcal{X}^R|\mathcal{X}^s)$
$p(\mathcal{X}^t|\mathcal{X}^S)$
$\mathcal{X}^t : \mathcal{X}^s$
Filter
$\mathcal{X}^R : \mathcal{X}^s$
Smoother

Figure 5.4.1: Smoother and filter combined in a single optimization problem and represented as a Bayes tree. A separator is selected so as to enable parallel computations. The image is inspired by S. Williams *et al.* (2014), but modified to fit the notation of this thesis.

To keep the filter operating in constant time, an approximately constant number of factors have to be maintained. Hence, intermediate states that no longer is referred to by future factors should be removed. Some factors are marginalized out, while others are moved to the smoother. Marginalization is equivalent to simply dropping the respective conditionals from the chordal Bayes net. The smoother updates could be solved by simply performing batch optimization, however it is cheaper to update the factor graph using the iSAM2 algorithm (Kaess, Johannsson, Roberts, Ila, J. Leonard, *et al.*, 2011). The iSAM2 update strategy is also chosen for the filter, but for a smaller window of factors.

Concurrent operations require synchronization of the two independently running processes. Synchronization happens after each iteration of the smoother. In order to keep the filter run time constant, it can never wait for the smoother. Hence, upon finishing an iteration, the smoother waits for the filter to finish its current update. When the processes are synchronized the orphaned subtree of the filter is merged back into the updated tree of the smoother. However, the original separator has changed independently in both the smoother and the filter. This is solved by only incorporating the change from the filter with respect to the original separator. In other terms, exactly the information the filter would add had it instead been run in sequence with the smoother. The inactive key states that are not removed from the filter is transferred to the smoother; this is done during synchronization by re-eliminated with a variable ordering (section 5.3.1) that changes the separator closer to the filter and thus transfers the previous separator and potential intermediate states from the filter into the smoother.

CHAPTER 6

Overview over Sensor and Software Setup

In this chapter datasets and their sensor setup are first described, followed by a description of the software that are employed to envision the developed system covered in sections 7 and 8.

6.1 milliAmpere

milliAmpere is a prototype of the autonomous ferry being designed as part the Autoferry project (Nilsen, 2017). An in-action image of milliAmpere can be found in fig. 1.1.1. The ferry is intended to transport pedestrians and cyclists between *Ravnkloa* and *Vestre Kanalkai at Brattøra* in Trondheim. This is a $\sim 110m$ wide passage shown in figure 6.1.1.

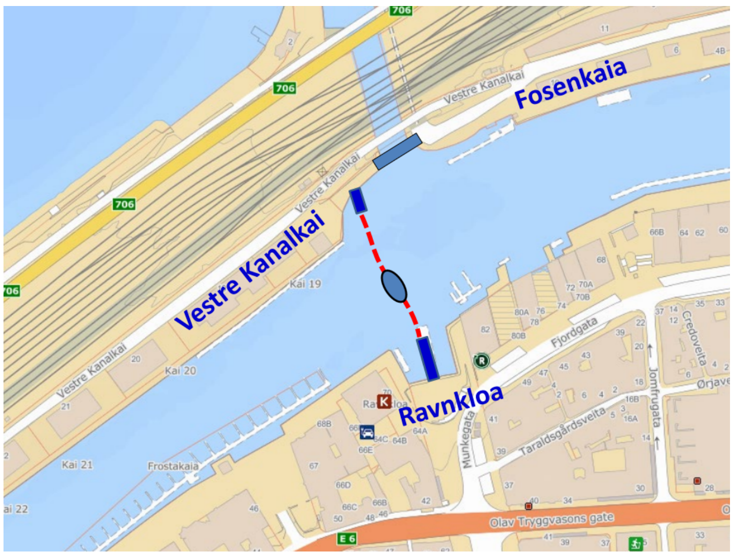


Figure 6.1.1: Intended operational region for mA2. The image shows an overview of Brattøra in Trondheim. The blue oval illustrates mA2 traveling along the striped lines, between the highlighted dark blue docking areas on Ravnkloa and Vestre Kanalkai.

milliAmpere is equipped with a sensor platform placed on top of the approximately 3-meter tall vessel. The platform consist of a camera setup of five EO and five IR cameras mounted to cover a 360° Field of view (FOV). The sensor platform additionally has a 360° LiDAR and an INS (RTK-GNSS and IMU). The specifications of each sensor are listed below.

- 5 × EO cameras (FLIR BlackFly 2), resolution: 2448 × 2048, max framerate: 22 fps, Kowa LM6JC lens, focal length 6mm, HFOV: 81.9°
- 5 × IR cameras (FLIR Boson 640), resolution: 640 × 512, max framerate: 9 fps, Kowa LM6JC lens, focal length 4.9mm, HFOV: 95°
- 1 × Velodyne VLP-16 rotating 3D laser scanner, update rate of 5-20 Hz, 16 channels, 0.1°-0.4°horizontal angular resolution, 2.0°vertical angular resolution, 3 cm distance accuracy,

field of view: 360°horizontal, 30°vertical, range: 100 m

- 1 × RTK-GNSS (Hemisphere Vector VS330), gyro stabilizer to measure heading at an accuracy of 0.05 degrees, horizontal accuracy 0.30m without RTK and 0.01m with RTK, update rate of 20Hz
- 1 × IMU (Xsens MTI-G-710), update rate of 100Hz

The sensor setup on milliAmpere was supplied by the stereo rig described in Theimann *et al.*, 2020. The rig uses two identical electro-optical cameras in the stereo setup. They are delivered by FLIR, and the camera model is Blackfly S GigE. The selected lens was bought from Edmund

- 2 × EO cameras (FLIR Blackfly S GigE) resolution: 2448 × 2048, max framerate: 24 fps, Edmund Optics C Series lens, focal length 8.5mm

The mounted full sensor setup can be seen in figure 6.1.2

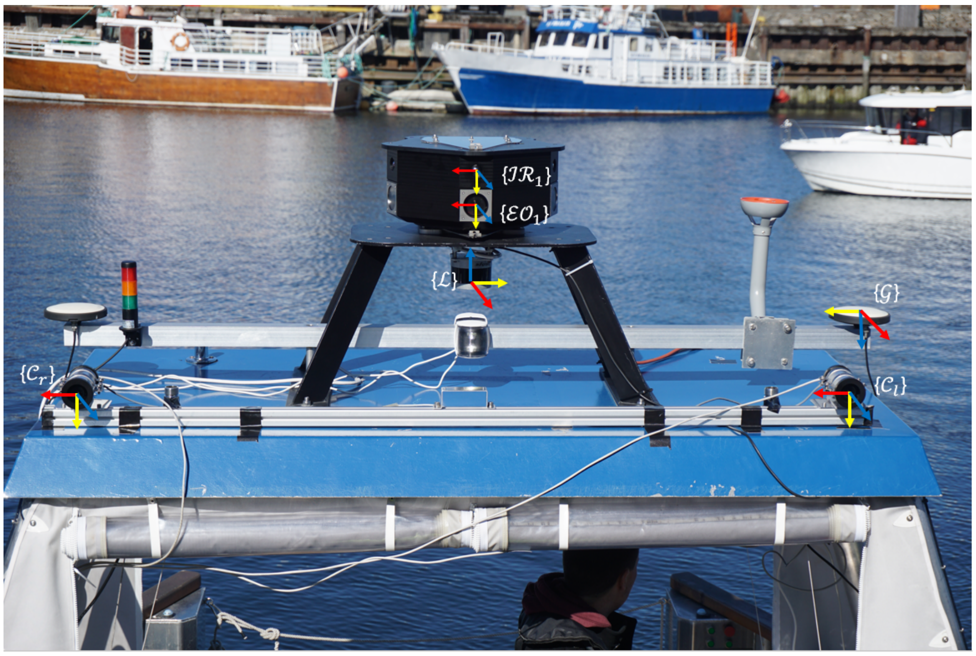


Figure 6.1.2: Illustration of the milliAmpere sensor setup listed below. Sensor frames are illustrated in the image. The red arrow depicts the x-direction, while yellow is y-direction and z is marked as blue. The sensor coordinate frames are denoted by: $\mathcal{IR}$ = infrared cameras, $\mathcal{EO}$ = electro optical cameras, $\mathcal{L}$ = LiDAR, $\mathcal{G}$ = GNSS, and $\mathcal{C}_i$ = stereo cameras which is further denoted left *(l)* and right*(r)*. Image captured by the author.

6.2 The KITTI Dataset as an Alternative to the milliAmpere Dataset

The KITTI dataset is included as an alternative benchmark for testing. The KITTI dataset (Geiger, Lenz, Stiller, and Urtasun, 2013a) is a publicly available dataset recorded from a car driving around urban areas in Karlsruhe, Germany. The dataset is split into 11 sequences (sequence 00-10) where the amount of details and number of objects vary with different locations. All of the sequences provide a ground truth of the trajectory, while 6 of which contain one or more loop closure events (revisits a location). The main purpose of the dataset is to provide a benchmark for development of computer vision and robotic algorithms targeted at autonomous driving. The variety in content between the sequences allows developers to test algorithms for stereo vision, visual odometry, 3D/2D object detection and much more, for different scenarios. The sensor platform is equipped with both grayscale and color stereo configured cameras, LiDAR, and an INS. Sensor setup with their internal coordinate frame on the vehicle is displayed in figure 6.2.1, while sensor specifications are listed below.

- 2 × PointGray Flea2 grayscale cameras (FL2-14S3M-C), 1.4 Megapixels, 1/2" Sony ICX267 CCD, global shutter
- 2 × PointGray Flea2 color cameras (FL2-14S3C-C), 1.4 Megapixels, 1/2" Sony ICX267 CCD, global shutter
- 4 × Edmund Optics lenses, 4mm, opening angle 90°, vertical opening angle of region of interest (ROI) 35°
- 1 × Velodyne HDL-64E rotating 3D laser scanner, 10 Hz, 64 beams, 0.09 degree angular resolution, 2 cm distance accuracy, collecting 1.3 million points/second, field of view: 360°horizontal, 26.8°vertical, range: 120 m
- 1 × OXTS RT3003 inertial and GPS navigation system, 6 axis, 100 Hz, L1/L2 RTK, resolution: 0.02m / 0.1°

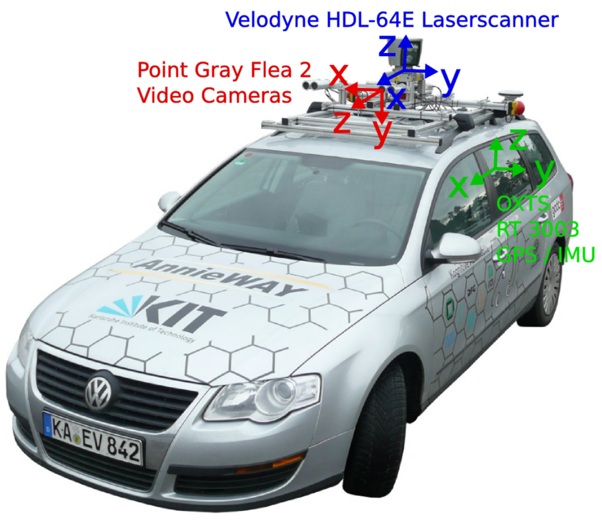


Figure 6.2.1: The sensor platform used in the KITTI dataset. The location of the cameras, lidar and INS system is illustrated together with their coordinate frame. Reprinted by permission from Geiger, Lenz, Stiller, and Urtasun (2013).

6.3 Software

6.3.1 Robotic Operating System (ROS)

Robot Operating System (ROS) (Quigley *et al.*, 2009) is a distributed and modular open-source middleware for robotic software development. ROS follows a peer-to-peer communication structure, embedding several different styles of communication such as synchronous services, asynchronous actions, and a publisher-subscriber alternative where nodes communicate over topics. These nodes are the core of the ROS communication structure. Every node can embed multiple publishers and subscribers. The subscribers are called whenever new information is available on the topic that they subscribe to. A callback function is registered to every subscriber where a set of logical operations are executed every time they are called. The result of the callback function can then yet again be published to a specified topic. Nodes support both sequential and multi-thread behaviour by applying either the single-threaded `spin()` looper or the `MultiThreadedSpinner()`. ROS utilizes a set of software libraries, tools, and conventions to simplify and standardize the task of complex and robust robot behavior. This includes visualization packages such as rqt that can visualize the overall system structure as a graph and Rviz for visualization of data that is published through standard ROS messages. This may include the pose of the vehicle, point clouds and much more. Another valuable tool included with ROS is the logging package called *rosbags*. Rosbags allows the user to store published data, for example sensor data and robot behavior, with time stamps for each measurement. This is a useful tool to collect and organize real world sensor data, such that testing may be performed later.

6.3.2 kitti2bag

kitti2bag is a third party python package developed specifically for transforming the raw data from the KITTI dataset into a rosbag. The library is credited on the KITTI dataset webpage (Geiger, Lenz, Stiller, and Raquel, 2013).

6.3.3 Trajectory Evaluation Tool

The trajectory evaluation toolbox by Zhang and Scaramuzza (Z. Zhang and Scaramuzza, 2018) was used to quantify the quality of the estimated trajectory. The toolbox has support for Absolute Trajectory Error (ATE) and Relative/Odometry Error (RE). ATE is widely used to evaluate visual odometry/SLAM algorithms because it produces a single number metric which makes it easy to compare performance. ATE is computed by first aligning the estimated trajectory to the ground truth, and then calculating the Root-Mean-Square Error (RMSE) over the aligned estimate and the ground truth. The alignment are typically done either at the beginning of the trajectory, or more commonly by computing the $SE(3)$ transformation that minimizes the overall ATE of the trajectory.

6.3.4 OpenCV

Open Source Computer Vision Library (OpenCV) is an open-source software library mainly aimed at real-time computer vision. OpenCV lists a comprehensive set of more than 2500 algorithms for both classic and state-of-the-art computer vision. This includes optimized algorithms for feature detection and tracking, and calculation of essential matrix to mention some of which are of relevance to this thesis. OpenCV has C++, Python, Java and MATLAB interfaces, and supports Windows, Linux, Android and Mac OS. Additionally, OpenCV has built-in GPU support for many of its functions allowing boosted computational performance.

6.3.5 Eigen

Eigen is an open source C++ library for linear algebra. It contains templated headers for matrices, vectors, numerical solvers, and related algorithms. The library is fast and well-suited for a wide range of tasks, while being targeted at operations related to transformation matrices in this thesis.

6.3.6 Georgia Tech Smoothing and Mapping (GTSAM)

The GTSAM toolbox is an open-source C++ library that performs large scale optimization. GTSAM uses factor graphs and Bayes networks as the underlying computing paradigm rather than sparse matrices. The factor graphs can be converted to a, typically, sparse matrix equivalent which can be solved using multiple different approaches. What separates GTSAM from many other optimization libraries is that it has multiple predefined factors for standard application which simplifies the implementation. In addition to the predefined factors, GTSAM also supports implementation of customized factors. GTSAM uses *Values* as initial estimates for the state variables in the factors. However, what most uniquely seperates GTSAM from other optimization libraries is the inclusion of algorithms such as iSAM and iSAM2 for back end optimization in SLAM problems. GTSAM do also support filtering, smoothing and batch optimization for sequences of factors. Building on this, the library has support for concurrent behaviour of batch filtering and smoothing solution, where the iSAM2 algorithm is used to update the factor graph (S. Williams *et al.*, 2014).

6.3.7 Pitch Yaw Roll (PYR)

PYR is an approach developed by Barnada *et al.* (2015), that estimate angular changes from monocular visual data, by analysing distant points. This strategy is based on the fact that the motion of distant points is not dependent on translation, but only on the rotation perceived by the camera. PYR does not require features to be extracted, but rather use phase correlation to estimate the optical flow. The algorithm also estimate the illumination changes between the compared frames, which allows to largely stabilize the estimation of image correspondences and motion vectors. Phase correlation can also be made robust against multiple motion, occlusions and other typical sources of failure for feature-based or photometric matching.

In practice are several predefined sub-regions of the image extracted where distant points are expected to be found, i.e. not in the sky or at the bottom of the image. Phase correlation is then applied independently to each sub-regions of the image to obtain a sparse set of optical flow vectors. Then, the relative rotation angles of the camera are inferred from the individual components of the flow vectors. PYR is significantly less complex and much faster than a full egomotion computation from features. PYR may therefore provide a useful prior to reduce search spaces of optimization schemes.

6.3.8 Joint Epipolar Tracking (JET)

Joint Epipolar Tracking is a sparse direct bundle adjustment algorithm developed by Bradler *et al.* (2017), for optimizing relative pose transformations and feature correspondences. Traditionally, pose estimation is considered as a two step problem. First, feature correspondences are determined and in a second step the relative pose is estimated, often by minimizing the reprojection error. JET rather introduce a loss function that allows to simultaneously optimize the unscaled relative pose, as well as the set of feature correspondences directly in a one-step approach. This is solved by considering the image intensities of the feature correspondences rather than their position only. At the time this algorithm was introduced it outperformed the classical reprojection error optimization on two synthetic datasets and on the KITTI dataset. This, while running in real-time on a single CPU thread.

The developed VSLAM frontend algorithm described in section 7 would greatly benefit from this software. However, after spending some time trying to integrate the software into the system it was in the end not accomplished. The reason being that JET uses older versions of OpenCV and other libraries, and even more importantly is based on an old Microsoft C++ compiler.

6.3.9 DBoW2 and DLoopClosure

A key feature that separates SLAM from VO is the ability to re-localize whenever previously mapped environments are recognized. This is useful both to correct for accumulated drift and re-localize in case of tracking failures. Storing, and comparing entire images to previous entries is very computationally inefficient, but using image regions surrounding features reduce the complexity. Still, a naive brute force approach which matches features against all previously detected features quickly becomes impractical. As the system has to perform in real-time, a place recognition module has to be very effective.

Dynamic Bags of Words 2 (DBoW2) is a visual place recognition module developed by Gálvez-López *et al.* (2012), which uses binary descriptors such as ORB (Rublee *et al.*, 2011) or Binary Robust Independent Elementary Features (BRIEF) (Calonder *et al.*, 2010), with an efficient search structure to find correspondences. Bag of Words (BoW)-algorithms uses a tree structure called visual vocabulary to convert an image into a sparse numerical vector. The visual vocabulary is constructed in an offline training step by discretizing the descriptor space into $\mathcal{W}$ visual words. Training a good vocabulary requires thousands of images captured in a wide range of conditions. At run time, DBoW2 builds database of images from previously visited locations and uses the vocabulary for fast matching and retrieval of potential correspondences. An implementation of the DBoW2 database and vocabulary that is built on OpenCV is available open-source (Gálvez-López *et al.*, 2014a). This implementation is furthermore templated, so it can work with any type of descriptor at the users convenience.

In practice visual vocabularies are constructed by first discretizing the descriptor space into $k_{\mathcal{W}}$ binary clusters by performing k-means clustering using k-means++ seeding (Arthur *et al.*, 2006). These clusters form the first level of nodes in the vocabulary tree. Subsequent levels are created by repeating this operation with the descriptors associated with each node, up to $L_{\mathcal{W}}$ times. This process generates a tree with $\mathcal{W}$ leaves, called the vocabulary words. Each word is weighted according to its relevance in the training set, decreasing the weight of very frequent and, thus, less discriminative, words. This weighting strategy is called the Term Frequency-Inverse Document

Frequency (tf-idf).

An image $\mathcal{I}_t$ taken at time t is converted into a BoW vector $\mathbf{v}_t \in \mathbb{R}^{\mathcal{W}}$ by traversing the binary descriptors of the features from the root to the leaves. At each level the intermediate nodes that minimize the Hamming distance is selected.

Each BoW vector is stored in a database together with an *inverse index* and a *direct index*. The inverse index stores, for every word $\mathcal{W}_i$, a list of images $\mathcal{I}_t$ in which it appears, together with the corresponding weight of the word in the image v_t^i. The inverse index is updated when a new image $\mathcal{I}_t$ is added to the database, and accessed when the database is searched for some image. This allows to perform comparisons only against those images that have some word in common with the query image. The direct index is not used as often, and stores the nodes which are ancestors to the words present in $\mathcal{I}_t$, as well as the list of local features $\mathbf{u}_{tj}$ 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.

Loop Detection Algorithm
Gálvez-López *et al.* (2012) also describes a four step algorithm to verify loop closing candidates, which is available open-source (Gálvez-López *et al.*, 2014b). Using ORB features, this algorithm has been tested on several real datasets, yielding an execution time of 9 ms to detect a loop a in a sequence with more than 19000 images (without considering the feature extraction).

The algorithm starts by querying the database with the BoW vector $\mathbf{v}_t$, resulting in a list of potential matches $\{< \mathbf{v}_t, \mathbf{v}_{t_1} >, < \mathbf{v}_t, \mathbf{v}_{t_2} >, ...\}$ associated with their similarity scores $s(\mathbf{v}_t, \mathbf{v}_{t_j}) \in [0, 1]$ are measured with the L1-score

$$s\left(\mathbf{v}_t, \mathbf{v}_{t_j}\right) = 1 - \frac{1}{2}\left|\frac{\mathbf{v}_t}{|\mathbf{v}_t|} - \frac{\mathbf{v}_{t_j}}{\left|\mathbf{v}_{t_j}\right|}\right| \tag{6.3.9.1}$$

The potential match scores are then normalized with the best score that could be expected to obtain in this sequence for the vector $\mathbf{v}_t$. This is approximated with the previous image processed, yielding the normalized similarity score

$$\eta\left(\mathbf{v}_t, \mathbf{v}_{t_j}\right) = \frac{s\left(\mathbf{v}_t, \mathbf{v}_{t_j}\right)}{s\left(\mathbf{v}_t, \mathbf{v}_{t-1}\right)}. \tag{6.3.9.2}$$

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

The second step is to group images that are close in time into *islands*, and treat them as one match, to prevent them competing among themselves whenever the database is queried. I.e. if $\mathcal{I}_t$ and $\mathcal{I}_{t'}$ represent a real loop closure, then $\mathcal{I}_t$ is very likely to be similar to $\mathcal{I}_{t'+\Delta t}, \mathcal{I}_{t'+2\Delta t}, ...$ The notation $T_i = \{t_{n_i}, ..., t_{m_i}\}$ is therefore introduced to represent the interval of timestamps for an island $V_{T_i} = \left\{\mathbf{v}_{t_{n_i}}, ..., \mathbf{v}_{t_{m_i}}\right\}$. Several matches, $\{< \mathbf{v}_t, \mathbf{v}_{t_1} >, < \mathbf{v}_t, \mathbf{v}_{t_2} >, ...\}$, is thus converted into a single match $< \mathbf{v}_t, V_{T_i} >$ ranked according to the score

$$H\left(\mathbf{v}_t, V_{T_i}\right) = \sum_{j=n_i}^{m_i} \eta\left(\mathbf{v}_t, \mathbf{v}_{t_j}\right). \tag{6.3.9.3}$$

The island yielding the highest score is selected as a matching group and continue to the the third step, called the 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,
$\{< \mathbf{v}_t - 1, V_{T_1} >, ..., \mathbf{v}_t - k, V_{T_k}\}$ to be accepted. If so, the vector $\mathbf{v}_{t'} \in V_{T'}$ that maximizes the score η is considered a loop closing candidate and passes to the final step, the geometrical verification stage.

The geometrical consistency checks applies Random Sample Consensus (RANSAC) to find a fundamental matrix, $\mathbf{F} \in \mathbb{R}^{3\times3}$, between images $\mathcal{I}_t$, and the matching candidate $\mathcal{I}_{t'}$ supported by at least 12 correspondences. Local features of the query image and the matched image must be compared and $\mathbf{F}$ can then be computed as described in section 2.4.2. Using the direct index in the database, the features correspondences of $\mathcal{I}_t$ and $\mathcal{I}_{t'}$ is quickly extracted. Only the features associated with the same nodes at level l in the vocabulary tree needs to be compared, thus speeding up the process. Finally, if the fundamental matrix was successfully computed, the match is accepted.

CHAPTER 7

Implementation of Frontend

This chapter will cover the frontend of the stereo VSLAM solution. An illustration of the pipeline is illustrated in figure 7.0.1, while an algorithmic description is detailed in algorithm 6. The initial outline of the implementation was to develop an efficient multiframe semi-direct SLAM algorithm, closely linked to SVO2 (Forster, Z. Zhang, *et al.*, 2016). However, due to issues described in section 6.3.8, the JET library that was going to be used for sparse photometric optimization was not successfully integrated. Therefore, a sequential multiframe fully feature based SLAM algorithm was developed. *OpenCV* is used for feature management and *Eigen* is used for pose representation and transformations. *GTSAM* is used for bundle adjustment, while *ROS* connects the complete system.

The system detects FAST features (Rosten *et al.*, 2006) in the left stereo image using a bucketed approach and matches them using the circular matching algorithm (Geiger, Ziegler, *et al.*, 2011). Features in the two images are then linearly triangulated Hartley *et al.*, 2003, ch.12.2 and the landmark positions are refined using structure-only bundle adjustment with a known stereo baseline from calibration. An initial estimate of the relative transformation between the two sequential image pairs is obtained up-to-scale by using Nistér's five-point algorithm (Nistér, 2004) while also employing RANSAC to remove feature outliers. It was observed that Nistér sometimes struggle when there are small feature displacements from frame to frame. Consequently, whenever the scaled motion relative from previous frames drops significantly, the rotation is replaced by PYR (Barnada *et al.*, 2015). The initial relative transformation is then refined using motion-only bundle adjustment, returning a scaled estimate of the relative transformation. Keyframes are decided if the body has moved more three meters, rotated more than 20 degrees, or more than 20 frames have passed. ORB descriptors are computed whenever a keyframe is decided. DBoW2 is then queried for loop closure matches using the ORB descriptors. If a loop is detected, the relative transformation between the current keyframe and the loop frame is found. The relative transformation is published to the backend for every frame, while stereo features, landmarks and loop closures are published every keyframe.

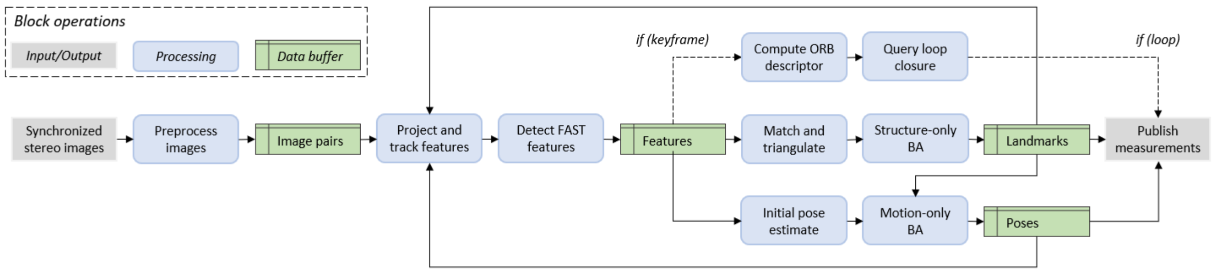


Figure 7.0.1: Pipeline for visual odometry algorithm. Boxes indicate algorithmic processes and arrows shows the data flow. Red boxes indicate input or output, while blue boxes describe algorithmic processes. Green boxes indicate data buffers of different types.

Algorithm 6: Main body of VSLAM frontend pipeline

Input: Image pair from both previous and current left and right stereo camera, $\mathcal{I}_{l,n-1}$, $\mathcal{I}_{r,n-1}$, $\mathcal{I}_{l,n}$ and $\mathcal{I}_{r,n}$ respectively. Features in previous left frame ($\mathbf{u}_{l,n-1}$). Triangulated refined features from previous frame $\{\mathbf{l}_{n-1}\}^*$. The former refined relative transformation $\left\{\mathbf{T}_{n-1/n-2}\right\}^*$.

Output: Refined relative transformation estimate $\left\{\mathbf{T}_{n/n-1}\right\}^*$. Refined landmarks position $\{\mathbf{l}_n\}^*$. Features for stereo image pair $\{\mathbf{u}_{l,n}, \mathbf{u}_{r,n}\}$. Loop detection container with *boolean* if loop closure is detected and the relative transformation $\left\{\mathbf{T}_{i_{loop}/n}\right\}^*$.

Nomenclature: The features in the current frame: $\mathbf{u}_{j,n} = \mathbf{u}^-_{j,n} \cup \mathbf{u}^+_{j,n}$.

- $\mathbf{u}^-_{j,n}$: Features that were matched to frame n. Subscript j denotes left or right image.
- $\mathbf{u}^+_{j,n}$: New detections in the current frame.

Nomenclature: i_{loop} denotes the iterator value for the keyframe loop closure match.

Nomenclature: $\mathbf{T}_{n/n-1}$ subscripts are separated by forward slash for clarity.

Initializaiton:

```
1  PreprocessImages($\mathcal{I}_{l,n}$, $\mathcal{I}_{r,n}$)
2  $\mathbf{u}_{l,n} \leftarrow$ BucketedFeatureDetection($\mathcal{I}_{l,n}$, _)
3  $\mathbf{u}_{r,n} \leftarrow$ CircularMatching(_, _, $\mathcal{I}_{l,n}$, $\mathcal{I}_{r,n}$, $\mathbf{u}_{l,n}$)
4  $\mathbf{l}_n \leftarrow$ Triangulate($\mathbf{u}_{l,n}$, $\mathbf{u}_{r,n}$)
5  $\{\mathbf{l}_n\}^* \leftarrow$ StructureOnlyBundleAdjustment($\mathbf{u}_{l,n}$, $\mathbf{u}_{r,n}$, $\mathbf{l}_n$)
```

Locked in state:

```
1  while $\mathcal{I}_{l,n}$, $\mathcal{I}_{r,n} \leftarrow$ ReadSyncronizedImagePair() do
2      PreprocessImages($\mathcal{I}_{l,n}$, $\mathcal{I}_{r,n}$)
3      $\mathbf{u}^-_{l,n} \leftarrow$ ProjectLandmarks($\{\mathbf{l}_{n-1}\}^*$, $\left\{\mathbf{T}_{n-1/n-2}\right\}^*$)
4      $\mathbf{u}^-_{l,n} \leftarrow$ calcOpticalFlowPyrLK($\mathcal{I}_{l,n-1}$, $\mathcal{I}_{l,n}$, $\mathbf{u}_{l,n-1}$, $\mathbf{u}^-_{l,n}$)
5      $\mathbf{T}_{n/n-1} \leftarrow$ Nistér5Point($\mathbf{u}_{l,n-1}$, $\mathbf{u}^-_{l,n}$)
6      if norm($\mathbf{t}_{n-1/n-2}$) $< \alpha$ then
7          $\mathbf{T}_{n/n-1} \leftarrow \mathbf{R}_{n/n-1} \leftarrow$ PYR($\mathcal{I}_{l,n-1}$, $\mathcal{I}_{l,n}$)
8      $\mathbf{u}^+_{l,n} \leftarrow$ BucketedFeatureDetection($\mathcal{I}_{l,n}$, $\mathbf{u}^-_{l,n}$)
9      $\mathbf{u}_{r,n} \leftarrow$ CircularMatching($\mathcal{I}_{l,n-1}$, $\mathcal{I}_{r,n-1}$, $\mathcal{I}_{l,n}$, $\mathcal{I}_{r,n}$, $\mathbf{u}_{l,n}$)
10     $\mathbf{l}_n \leftarrow$ Triangulate($\mathbf{u}_{l,n}$, $\mathbf{u}_{r,n}$, $\mathbf{P}_l$, $\mathbf{P}_r$)
11     $\{\mathbf{l}_n\}^* \leftarrow$ StructureOnlyBundleAdjustment($\mathbf{u}_{l,n}$, $\mathbf{u}_{r,n}$, $\mathbf{l}_n$)
12     $\left\{\mathbf{T}_{n/n-1}\right\}^* \leftarrow$ MotionOnlyBundleAdjustment($\mathbf{T}_{n/n-1}$, $\mathbf{u}_{l,n}$, $\mathbf{u}_{r,n}$, $\{\mathbf{l}_{n-1}\}^*$)
13     $\mathbf{T}_{keyframe} = \mathbf{T}_{keyframe} \cdot \left\{\mathbf{T}_{n/n-1}\right\}^*$
14     if IsKeyframe($\mathbf{T}_{keyframe}$) then
15         $\mathbf{d}_{l,n} \leftarrow$ computeDescriptor($\mathcal{I}_{l,n}$, $\mathbf{u}_{l,n}$)
16         $i_{loop} \leftarrow$ queryDLoopDetector($\mathbf{u}_{l,n}$, $\mathbf{d}_{l,n}$)
17         if $i_{loop} \neq -1$ then
18             $\mathbf{u}_{l,i_{loop}}$, $\mathbf{u}_{r,i_{loop}} \leftarrow$ CircularMatching($\mathcal{I}_{l,n}$, $\mathcal{I}_{r,n}$, $\mathcal{I}_{l,i_{loop}}$, $\mathcal{I}_{r,i_{loop}}$ $\mathbf{u}_{l,n}$)
19             $\mathbf{T}_{i_{loop}/n} \leftarrow$ Nistér5Point($\mathbf{u}_{l,n}$, $\mathbf{u}_{l,i_{loop}}$)
20             $\left\{\mathbf{T}_{i_{loop}/n}\right\}^* \leftarrow$MotionOnlyBundleAdjustment($\mathbf{T}_{i_{loop}/n}$, $\mathbf{u}_{l,i_{loop}}$, $\mathbf{u}_{r,i_{loop}}$, $\{\mathbf{l}_n\}^*$)
21         $\mathbf{T}_{keyframe} = \mathbf{I}$;
       return: {Output}
```

7.1 Preprocessing

The preprocessing of the image ensures that new images have a format that fits for analysis. Firstly, the stereo image pairs are rectified if this was not done beforehand. To obtain the rectified images, two different rectification algorithms were tried: OpenCV's `StereoRectify()` and ROS' `stereo_image_proc` software. The latter achieved the best results. Then, the stereo image pair are syncronized using the ROS `message_filters::Synchronizer` to ensure that the incoming image pair match. These two operations combined equals `ReadSyncronizedImagePair()` in algorithm 6. Next, the `PreprocessImages()` is defined. The images are converted to grayscale mainly because the algorithms used for feature extractors and descriptors are designed to analyse the intensity of pixels. If necessary, for example to remove static body-fixed items, the images are then cropped. A mask would serve the same purpose, but are harder to define correctly. For high resolution images, the resolution was considered dropped for two reasons. Firstly to improve computational time, and secondly to improve feature detection because an unnecessarily large amount of features could then be detected within a small region. Adjusting the resolution was ignored because the images at the end were read at maximum resolution of 1224×1024. Lastly, it was experimented with adjusting the brightness of the images, but this showed small to no improvement for the tested scenarios, while increasing processing time. This adjustment was therefore skipped.

7.2 Feature management

7.2.1 Bucketed feature detection

In this section the developed procedure for `BucketedFeatureDetection()` is described. The preprocessed left image is divided into a grid of 10×10 grid cells. Before new features are detected, an existing set of features are first stored in buckets. The bucket index of each feature is determined by the grid for which the image coordinate is placed. For each grid features are extracted using the FAST feature extractor provided by OpenCV. This is to ensure that the extracted features get an approximately uniform distribution over the entire image domain. By analysis, this shows that features are matched better following the bucketed approach than when they are placed closer to each other. By experimentation the default parameters of FAST yielded good result. The detected features in each grid are then sorted based on their *response*, a metric that compares the feature's quality. Non-max suppression is then performed where the tracked features are given the highest priority. For each newly detected feature it was check whether the coordinates of the feature are too close, in the image space, to any existing features coordinates. If not, the feature are accepted and added to the respective bucket. The feature acceptance procedure is repeated while the number features in the grid cell bucket is lower than a threshold, set to three. An example of the uniformly distributed feature detections is displayed in the left current image of fig. 7.2.1.

Algorithm 7: Bucketed feature detection

Input: Image $\mathcal{I}$. A set of existing features $\mathbf{u}^-$.

Output: New feature detections in image $\mathbf{u}^+$.

Nomenclature: Subscript, $\{\cdot\}_{x,y}$, denotes vector or matrix extracted at sliced image, $\mathcal{I}_{x,y}$, where $[x,\ y]$ is the location in the original image, $\mathcal{I}$.

```
Function BucketedFeatureDetection($\mathcal{I}$, $\mathbf{u}^-$):
1    Allocate $N \cdot M$ buckets for $u^-_{N,M}$ and $u^+_{N,M}$
2    for $u_i \in \mathbf{u}^-$ do
3        Place $u^-_i$ in bucket of $\mathbf{u}^-_{N,M}$
4    for $\mathcal{I}_{n,m} \leftarrow$ SliceImage($\mathcal{I}$, $n \in N$, $m \in M$) do
5        $\mathbf{u}^+_{n,m} \leftarrow$ extractor$\rightarrow$DetectFASTFeatures($\mathcal{I}_{n,m}$)
6        SortByRespone($\mathbf{u}^+_{n,m}$)
7        for $u^+_j \in \mathbf{u}^+_{n,m}$ do
8            if $u^+_j \notin \mathbf{u}^-_{n,m}$ and $u^+_j \notin \mathbf{u}^+_{n,m}$ then
9                $\mathbf{u}^+_{n,m} \leftarrow u^+_j$
10   return $\mathbf{u}^+$
```

7.2.2 Feature tracking

From the specialisation project (Hellum, 2020) it was experienced that the Lucas Kanade optical flow yielded better matches between sequential images than descriptor based matching. The Lucas Kanade optical flow tracking algorithm described in section 3.3 benefits from having initial estimates of the features coordinates. An initial estimate can potentially reduce the geometrical search space for the correct displacement of the optical flow vector for the correspondences. These initial guesses of the feature location can be obtained by projecting triangulated 3D points onto the image using equation 2.4.1.6 where the previous relative transformation are used for extrinsic parameters. This equals `ProjectLandmarks()` in algorithm 6. The features are then tracked from the previous left image to the current left image using the OpenCV function `calcOpticalFlowPyrLK()`.

7.2.3 Feature Matching in Stereo Image Pairs

For feature matching between stereo images, both descriptor based matching and optical flow based feature tracking were explored. For descriptor based matching, features and their ORB descriptor were detected in both stereo images. ORB was chosen because the descriptors are efficient to work with, which is essential for SLAM systems. This is argued by Mur-Artal, Montiel, *et al.*, 2015. The other reason that the ORB descriptor were used is that ORB-SLAM provides a well defined pre-trained vocabulary for loop closure detection based on ORB descriptors, which will be discussed in section 7.4. ORB uses binary descriptors, therefore the Hamming distance, d_h, is recommended distance metric for feature matching. This distance is equivalent to count the number of different elements for binary strings. The shorter the distance, the better the features match.

Feature matches the stereo images were found using the *cross check* approach, i.e. the descriptor was matched both from the left image to the right and the opposite. The match is accepted if the descriptor is matched both ways. The other descriptor based matching method that was explored is called the *ratio test*, by Lindeberg, 2012. This test algorithm compares a descriptor d_a in image $\mathcal{I}_a$ to the two closest descriptors d_{b1} and d_{b2} for image $\mathcal{I}_b$. Then the ratio test, $d_h(d_a, d_{b1})/d_h(d_a, d_{b2})$ is performed. For a low distance ratio d_{b1} is set to be a good match, while for a high distance ratio d_{b2} may be incorrect or ambiguous. By experimentation, Lindeberg (2012) found 0.8 to be a good distance threshold in order to accept a feature. Without more advanced procedures for prediction the descriptor position of the correspondence it was experienced that a large amount of features have to be detected in both images to ensure that correct matches in fact are present. This greatly increase the computational complexity, not only for the detection and matching, but also for operations such as bundle adjustment. Another downside is that a large amount of features that are closely spaced, by experimentation, showed small false displacements for the tracked features in section 7.2.2. Descriptor based matching was therefore ignored, and descriptors were rather only used for loop closure detection as will be detailed in section 7.4.

Visual odometry and VSLAM aims towards real-time operations. For real-time use-cases the goal of the feature management is rather to maintain a minimal set of robust features while providing sufficient information for the motion estimation. Here, robust refers to strong corners or edges that are uniformly spread across the image. The Lucas Kanade optical flow algorithm is a matching algorithm that fits this set of specifications. Geiger, Ziegler, *et al.* (2011) proposed an efficient sparse matching procedure called *circular matching*. This approach has also been used in the work of Cvišić *et al.* (2018), which is the current top contender for VO performed on the KITTI VO benchmark. The circular matching algorithm uses the sparse Lucas Kanade optical flow algorithm to match features between consecutive images pairs as detailed and illustrated in figure 7.2.1. This provides an efficient matching solution for stereo cameras that also maintain consistent features. From experimentation it was experienced that the circular matching procedure always yielded correct matches. Additionally, because it benefits for the uniformly spaced feature detections the `CircularMatching()` procedure was chosen to be implemented for the developed system.

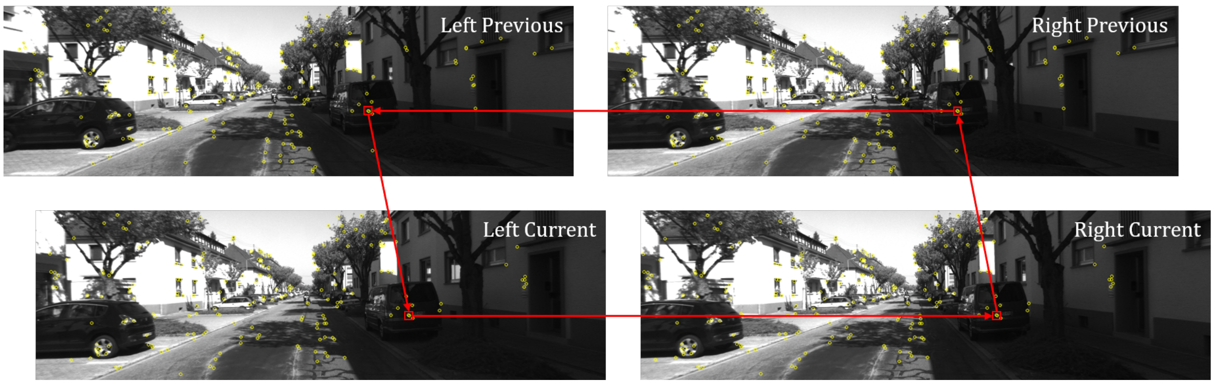


Figure 7.2.1: Circular feature matching. The procedure starts with an initial feature in the current left image. The feature is tracked consecutively in the previous left image, the previous right image, then to the current right image, and lastly to the current left image. Only if the tracked feature coordinates coincides with the initial location the match is accepted.

Algorithm 8: Circular matching

Input: Image pair from both previous and current left and right stereo camera, $\mathcal{I}_{l,n-1}$, $\mathcal{I}_{r,n-1}$, $\mathcal{I}_{l,n}$ and $\mathcal{I}_{r,n}$ respectively. Features in the current left image, $\mathbf{u}_{l,n}$.

Output: New feature detections in the right image $\mathbf{u}_r$.

```
Function CircularMatching($\mathcal{I}$, $\mathbf{u}^-$):
1    $\mathbf{u}_{l,n-1} \leftarrow$ calcOpticalFlowPyrLK($\mathcal{I}_{l,n}$, $\mathcal{I}_{l,n-1}$, $\mathbf{u}_{l,n}$)
2    $\mathbf{u}_{r,n-1} \leftarrow$ calcOpticalFlowPyrLK($\mathcal{I}_{l,n-1}$, $\mathcal{I}_{r,n-1}$, $\mathbf{u}_{l,n-1}$)
3    $\mathbf{u}_{r,n} \leftarrow$ calcOpticalFlowPyrLK($\mathcal{I}_{r,n-1}$, $\mathcal{I}_{r,n}$, $\mathbf{u}_{r,n-1}$)
4    $\hat{\mathbf{u}}_{l,n} \leftarrow$ calcOpticalFlowPyrLK($\mathcal{I}_{r,n}$, $\mathcal{I}_{l,n}$, $\mathbf{u}_{r,n}$)
5    CheckDisplacement($\mathbf{u}_{l,n}$, $\hat{\mathbf{u}}_{l,n}$)
6    return $\mathbf{u}_{r,n}$
```

7.2.4 Triangulation

In this section the `Triangulate()` process is described. Triangulation is the problem of determining a point's 3D position from a pair of feature correspondences and a known relative transformation of the camera poses. The 3D point is often referred to as a *landmark*. There are multiple ways of performing triangulation. A popular choice is the linear triangulation method described in Hartley *et al.*, 2003, ch.12.2, which is used by SLAM algorithms such as ORB-SLAM (Mur-Artal and Tardós, 2017). This method takes a feature that appears in both images, $\mathbf{u}_l$ and $\mathbf{u}_r$, and tries to find a landmark that satisfy equation 2.4.1.5 for the two views. For stereo configurations the projection matrices, $\mathbf{P}_l$ and $\mathbf{P}_r$, are calculated beforehand during calibration, while for monocular cases they have to be computed with the relation in equation 2.4.1.5. In this context subscipts l and r are used to reference the left and right camera respectively. The actual image coordinate of the feature is found by dividing by the third component of the projection

$$u = \frac{\boldsymbol{p}_1^T \tilde{\mathbf{l}}}{\boldsymbol{p}_3^T \tilde{\mathbf{l}}} \quad \text{and} \quad v = \frac{\boldsymbol{p}_2^T \tilde{\mathbf{l}}}{\boldsymbol{p}_3^T \tilde{\mathbf{l}}} \tag{7.2.4.1}$$

where $\mathbf{p}_i$ is the i'th row of P, $\tilde{\mathbf{l}}$ is the homogenous coordinates of the landmark, while u and v are the horizontal and vertical image coordinate respectively. This can be done for both views, giving a total of four equations. By multiplying both sides with the denominator and collecting terms, this can be written as a linear homogeneous system of equations

$$\mathbf{A} \cdot \tilde{\mathbf{l}} = \begin{bmatrix} u_l \boldsymbol{p}_{l,3}^T - \boldsymbol{p}_{l,1}^T \\ v_l \boldsymbol{p}_{l,3}^T - \boldsymbol{p}_{l,2}^T \\ u_r \boldsymbol{p}_{r,3}^T - \boldsymbol{p}_{r,1}^T \\ v_r \boldsymbol{p}_{r,3}^T - \boldsymbol{p}_{r,2}^T \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \tilde{w} \end{bmatrix} = \mathbf{0} \tag{7.2.4.2}$$

which can be solved using the Singular Value Decomposition (SVD) (Golub *et al.*, 1965). The solution corresponds to the smallest singular value of $\mathbf{A}$. This method is closely related to the Direct Linear Transformation algorithm (Hartley *et al.*, 2003).

The linear triangulation is performed for each feature match. Because not all landmarks are correctly triangulated, their validity have to be checked before they are accepted. The checks that are used to confirm that the landmarks are the following:

- Check that the reprojection error does not surpass 0.5 pixels in either images.
- Check that the depth of the landmarks in fact is positive.
- Check that the depth of the landmark does not surpass a threshold, given by a scale factor times the baseline. The reason being that when the baseline decreases the accuracy of distant/*far away* landmarks decreases. Furthermore, far away landmarks only provide information about the rotation of the body. For a generalized solution where landmarks are used to estimate both rotation and translation these have to be filtered out.

Algorithm 9: Triangulation procedure

Input: Matched features in left and right image, $\mathbf{u}_l$ and $\mathbf{u}_r$. Projection matrix for left and right camera, $\mathbf{P}_l$ and $\mathbf{P}_r$.

Output: Triangulated landmarks, $\mathbf{l}$.

```
Function DLT($\mathbf{u}_l$, $\mathbf{u}_r$, $\mathbf{P}_l$, $\mathbf{P}_r$):
1    Construct $\mathbf{A}$ matrix in equation 7.2.4.2
2    $\tilde{\mathbf{l}}_i \leftarrow$ smallest value of SVD($\mathbf{A}$)
3    return HomogenousToCartesian($\tilde{\mathbf{l}}_i$)
Function Triangulate($\mathbf{u}_l$, $\mathbf{u}_r$, $\mathbf{P}_l$, $\mathbf{P}_r$):
1    for $\mathbf{u}_{l,i} \in \mathbf{u}_l$, $\mathbf{u}_{r,i} \in \mathbf{u}_r$) do
2        $\mathbf{l}_i \leftarrow$ DLT($\mathbf{u}_{l,i}$, $\mathbf{u}_{r,i}$, $\mathbf{P}_l$, $\mathbf{P}_r$)
3        $\{\hat{\mathbf{u}}_{l,i}, \hat{\mathbf{u}}_{r,i}\} \leftarrow$ ReprojectLandmark($\mathbf{l}_i$, $\mathbf{P}_l$, $\mathbf{P}_r$)
4        if ValidTriangulation($\mathbf{u}_{l,i}$, $\mathbf{u}_{r,i}$, $\hat{\mathbf{u}}_{l,i}$, $\hat{\mathbf{u}}_{r,i}$) then
5            $\mathbf{l} \leftarrow \mathbf{l}_i$
6    return $\mathbf{l}$
```

7.2.5 Structure-only Bundle Adjustment

In this section the algorithmic process for `StructureOnlyBundleAdjustment` is explained. Because the extrinsic of the stereo pinhole model is known from calibration, it is not necessary to include the stereo transformation in the bundle adjustment. Therefore, structure-only bundle adjustment is used rather than local full bundle adjustment, to refine only the landmark positions.

Using the intrinsic and extrinsic parameters obtained from calibration, the triangulated landmarks in the current frame are re-projected onto the current left and right image using equation 2.4.1.7. The *reprojection error* is formulated as the difference between the re-projected feature position and the original feature position. This essentially describes one iteration of the structure-only bundle adjustment in equation 7.2.5.1. Using a non-linear solver such as the Levenberg-Marquardt, the re-projection error is iteratively minimized with respect to the world landmark position. This procedure thus refines the world landmark position from the linear triangulation.

$$\{\mathbf{l}^{w*}\} = \underset{\mathbf{l}^w}{\operatorname{argmin}} \sum_i \|\pi_l\left(\mathbf{l}_i^w; \mathbf{T}_{c_l w}, \mathbf{K}_l\right) - \mathbf{u}_i\|^2 + \|\pi_r\left(\mathbf{l}_i^w; \mathbf{T}_{c_r c_l} \cdot \mathbf{T}_{c_l w}, \mathbf{K}_r\right) - \mathbf{u}_i\|^2 \qquad (7.2.5.1)$$

In practice this was done with GTSAM by deriving a new class from the built-in class `NoiseModelFactor1<T>`. This essentially means that a factor is created where one templated parameter of type T is optimized. For this to be done an error term has to be defined, and a parameter to be optimized has to decided. The reprojection error described in equation 7.2.5.1 was implemented as the error term, which was optimized with respect to each individual landmark position. All landmark-to-feature correspondences in the current left and right image were added to a non-linear factor. The factor graph were then solved using the Levenberg-Marquardt optimizer, yielding the refined transformation. An illustration of a factor graph constructed for full bundle adjustment is displayed in Figure 7.2.2. For structure-only bundle adjustment, only the landmarks are optimized.

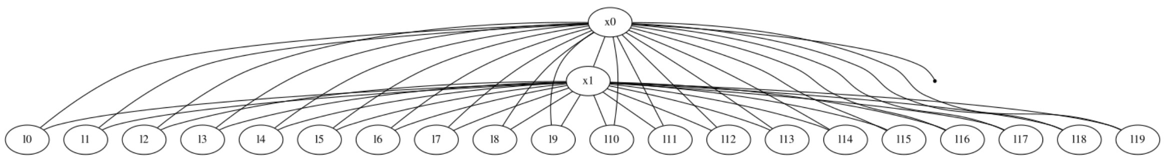


Figure 7.2.2: Example factor graph constructed with GTSAM run-time. The graph depicts a factor graph formulation for full-bundle adjustment of the reprojection error where both poses, **x**, and landmarks, **l**, are optimized. x0 additionally has a prior in the top right of the graph.

7.3 Motion Estimation

7.3.1 Initial Estimate of Relative Transformation

In this section the process of finding initial motion estimates using the `Nistér5Point()` algorithm and `PYR()` is explained. To estimate the essential matrix between two frames Nistérs five-point algorithm is used (Nistér, 2004). This algorithm is used in conjunction with RANSAC (Fischler *et al.*, 1981). More specifically, the algorithm calculates the essential matrix for a number of random five point subsets. The essential matrix that coincides with the largest number of RANSAC inliers is returned. OpenCV provides an implementation of this algorithm in their function `findEssentialMat()`. An advantage of this algorithm is that feature outliers may be removed while simultaneously computing the essential matrix. Features that do not coincide with the general motion of the body is thus ignored. RANSAC thus works, to some extent, as a filter to remove tracked features of moving objects. The relative pose from the essential matrix is only computed up-to scale, as shown by Longuet-Higgins (Longuet-Higgins, 1981). This means that the L2 norm of the recovered translation in each direction adds up to 1. From the essential matrix, $\mathbf{R}$ and $\mathbf{t}$ is decided using Singular Value Decomposition (SVD). The SVD will return four possible solutions for $\mathbf{R}$ and $\mathbf{t}$. To determine what combination gives the correct pose, some keypoints are triangulated to ensure that the Cheirality constraint (Longuet-Higgins, 1981) is satisfied, i.e. all points are in front of both cameras. OpenCV provides an implementation of this procedure in their function `recoverPose()`. Because these algorithms only are capable of determining the relative transformation up-to-scale, the scale has to be provided another way. Smooth motion is assumed because of the vehicle's large moment of inertia. Therefore, the initial motion estimate are scaled according to the scale following from the previous frame. This limits the uncertainty of the motion estimate to be proportional to the acceleration of the vehicle opposed to just being set as a static unknown from the up-to-scale estimate. This scaled motion estimate works as an initial guess for the current relative frame motion. The initial transformation estimate then has to be refined according to the procedure that will be discussed in section 7.3.2. `findEssentialMat()` were ignored whenever the vehicle were moving slowly. This is because it was experienced that this function needs distinct motion to correctly find the exact RANSAC match. In other words, because the translation has unit length 1 there should be a more dominant component to the translation, which is not the case when standing still. Therefore, whenever the vehicle was moving slowly, the translation was set equal to the optimized relative translation from the previous frame. The rotation was replaced by the one produced by the `PYR()` algorithm. It was also tried always using `PYR()`, but generally `findEssentialMat()` yielded better results when there only were a small rotation component. The algorithmic process of this algorithm is described in section 6.3.7.

7.3.2 Motion-only Bundle Adjustment

In this section the algorithmic process for `MotionOnlyBundleAdjustment()` is explained. Using an initial approximation of the relative transformation, the landmarks discovered in the previous frame are reprojected onto the current frame according to equation 2.4.1.7 and compared the corresponding features in both the current left and right image. The resulting error forms the *reprojection error*. This essentially describes one iteration of the motion-only bundle adjustment in equation 7.3.2.1. Using a non-linear solver such as the Levenberg-Marquardt, the re-projection error is iteratively minimized with respect to the relative transformation, $\mathbf{T}$, gradually adjusting the rotation, $\mathbf{T}$, and translation, $\mathtt{t}$, until the overall re-projection error of all landmarks cannot be improved any further. When this is the case the optimization has hit a local minimum. The more information that is available improves the likelihood of the optimization hitting a global minimum, i.e. the correct solution. This procedure thus refines the motion estimate and, because the scale of the landmarks are available from the stereo configuration, the returns the scale of the motion.

$$\{\mathbf{T}^*_{c_l w}\} = \underset{\mathbf{T}_{c_l w}}{\operatorname{argmin}} \sum_i \|\pi_l\left(\mathbf{l}^w_i; \mathbf{T}_{c_l w}, \mathbf{K}_l\right) - \mathbf{u}_i\|^2 + \|\pi_r\left(\mathbf{l}^w_i; \mathbf{T}_{c_r c_l} \cdot \mathbf{T}_{c_l w}, \mathbf{K}_r\right) - \mathbf{u}_i\|^2 \qquad (7.3.2.1)$$

In practice this was done with GTSAM by deriving a new class from the built-in class `NoiseModelFactor1<T>`. This essentially means that a factor are created where one templated parameter of type T is optimized. For this to be done an error term has to be defined, and a parameter to be optimized has to be decided. The reprojection error described in equation 7.3.2.1 was implemented as the error term, which was optimized with respect to the affine 6 DOF relative transformation. All landmark-to-feature correspondences from the previous to current left and right image were added to a non-linear factor. Then, the factor graph were solved using the Levenberg-Marquardt optimizer, yielding the refined transformation. An illustration of a factor graph constructed for full bundle adjustment is displayed in Figure 7.2.2. For motion-only bundle adjustment, only the poses are optimized.

7.4 Loop Closure Detection

The DBoW2 algorithm described in section 6.3.9 was chosen for place recognition in order to detect loop closures. In their survey, B. Williams *et al.* (2009) concludes that image-to-image based matching techniques seemed to scale better than map-to-map or image-to-map methods. The choice were then between DBoW2 and FAB-MAP 2.0. These solutions achieves similar recall performance, but DBoW2 has much lower computational complexity. During testing, DBoW2 were found to be very reliable and no false positives were encountered.

The DBoW2 library is also used with the ORB-SLAM family. Similarly to ORB-SLAM the ORB descriptor were chosen, as the descriptor is more invariant to changes in scale and rotation compared to BRIEF Mur-Artal, Montiel, *et al.*, 2015 and is more computationally efficient Gálvez-López *et al.*, 2012. Furthermore, ORB-SLAM2 has a readily available pre-trained visual vocabulary at their Github repository (Mur-Artal, Montiel, *et al.*, 2017). The vocabulary was downloaded and converted from *text* to *binary* format which reduced the initial load period with a degree of approximately twenty.

To limit the computational complexity, the descriptor $\mathbf{d}_{l,n}$ of the features $\mathbf{u}_{l,n}$ was computed for the left current image $\mathcal{I}_{l,n}$ using OpenCV function `computeDescriptor()` only for keyframes. The descriptor was then added to the DBoW2 database and, by using the DLoopDetector algorithm described in section 6.3.9, loop closure matches were queried. This equals the function `queryDLoopDetector()`. What should be noted is that loop closure matches doesn't necessarily have to be an exact match in pose. There is usually a rotation and translation component between the pose matches that have to found.

If a loop closure was detected, matches between the current image $\mathcal{I}_{l,n}$ and the loop closure frame $\mathcal{I}_{l,i_{loop}}$ was found by applying the circular matching algorithm discussed in section 7.2.3, returning $\mathbf{u}_{l,i_{loop}}$ and $\mathbf{u}_{r,i_{loop}}$. Descriptor based matching were also experimented with, however the matches found by DBoW2 were never returned, thus the descriptor matches would have to be found all over again. In this situation, better results were rather found by the circular matching approach. Using the feature matches the relative transformation, $\mathbf{T}_{i_{loop}/n}$, was computed up-to-scale following the procedure described in section 7.3. This was followed by the motion-only optimization described in section 7.3.2 using the landmarks of the current frame $\{\mathbf{l}_n\}^*$ and the matched loop features, $\mathbf{u}_{l,i_{loop}}$ and $\mathbf{u}_{l,i_{loop}}$, to get an optimized scaled estimate, $\left\{\mathbf{T}_{i_{loop}/n}\right\}^*$

Using the features of the current frame, $\mathbf{u}_{l,n}$ and $\mathbf{u}_{r,n}$, and the features of the loop closure frame, $\mathbf{u}_{l,i_{loop}}$ and $\mathbf{u}_{r,i_{loop}}$, the relative transformation were found following following the procedure in section 7.3.

CHAPTER 8

Implementation of Backend

A modular backend strategy was built such that multiple sensors, hereunder GNSS, IMU and VSLAM frontend measurements, could be embedded into a larger factor graph optimization scheme. The backend has a multi-threaded structure as detailed in section 5.4, where the optimization scheme is divided into a short-term filter and a long-term smoother. The backend spins a rosnode that sequentially searches for new measurements from each sensor. When receiving new measurements they are added as factors together with an initial estimate of the state's value, following individual callback strategies specified in sections 8.1, 8.2 and 8.3. As illustrated in figure 8.0.1, the new factors are first stored in a temporary factor graph container, before they are jointly added to the short-term filter. These factors may include pose estimates from GNSS, IMU preintegration and the VSLAM frontend, as well as projection factors of landmarks. A main backend callback function is also spun which searches for updates to the temporary factor graph container at a rate of 100Hz. Whenever updates are registered, the temporary factor graph container is added to the main filter factor graph, hence, because of the high callback frequency factors are in practice directly added to the filter.

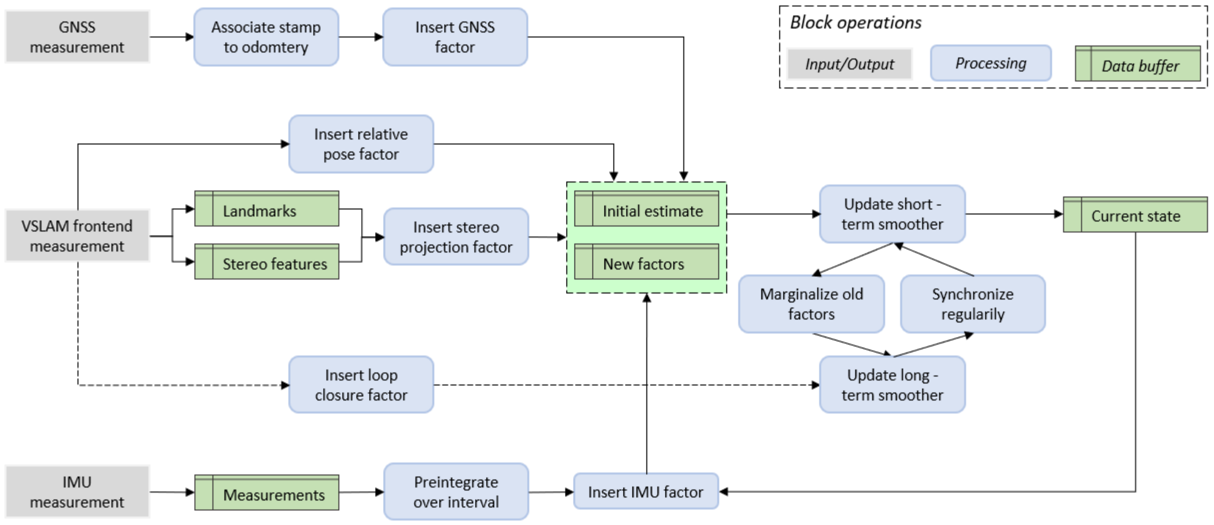
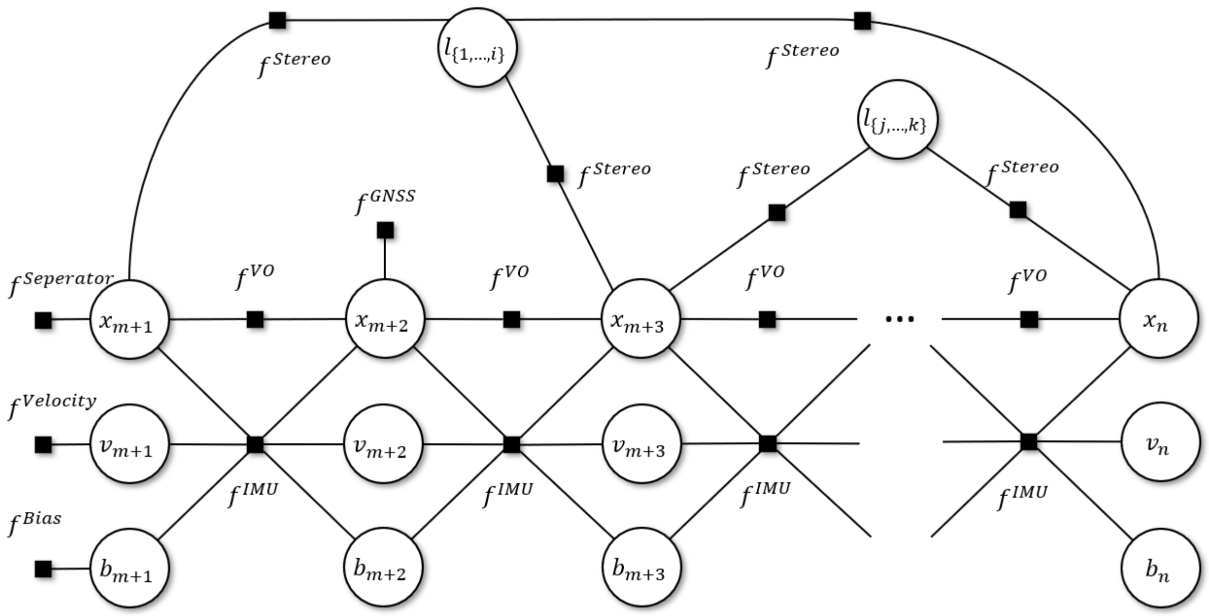
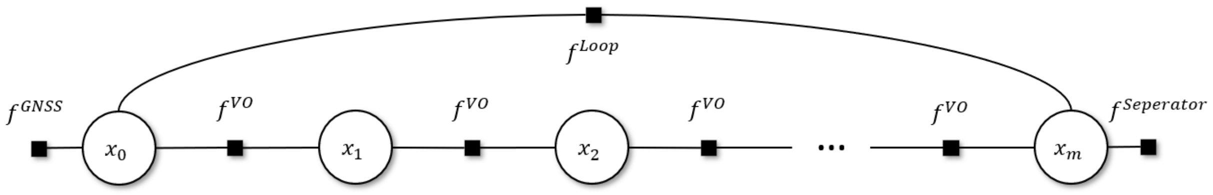


Figure 8.0.1: Data flow diagram for backend. Boxes indicate algorithmic processes and arrows shows the data flow. Red boxes indicate input or output, while blue boxes describe algorithmic processes. Green boxes indicate data buffers of different types.

Whenever the filter is updated three operations are performed. Firstly, new factors are added to the filter as they become available. Next, the complexity of the filter has to be kept constant. Therefore, inactive factors in the filter - i.e. intermediate states that no longer is referred to by future factors - are identified. Key factors have to be chosen amongst the inactive factors. The relative motion factors, `BetweenFactor<Pose3>`, was chosen as key factors for this purpose. The second operation that is performed during the filter update is then to remove the factors that is not considered being key factors. Lastly, the key factors are moved to the long-term smoother. The latter operation is only performed when the filter and smoother is synchronized following the approach described in section 5.4. An example of the constructed filter factor graph, with all sensors enabled, is illustrated in fig. 8.0.2.



(a) Multisensor factor graph example for the short-term filter. There is only a seperator on the leftmost pose, while velocities and biases rather have a prior factor.



(b) Loop closing factor graph example for the long-term smoother. There is only a seperator on the rightmost pose.

Figure 8.0.2: The a practical example of the concurrent filter and smoother as solved in the developed system. x, v, b, and l describe variable values for pose, velocity, bias and landmarks respectively. Filled black squares related by f denotes factors. The superscript of f describes the factor/measurement type.

The long-term smoother is updated every time the filter and smoother is synchronized. The identified key factors in the filter are integrated into the smoother using variable re-ordering where a new separator is chosen amongst the key factors. Loop closures can be added to smoother when both loop pose indices exist in the graph. Therefore, whenever a loop is detected all existing `BetweenFactor<Pose3>`s in the filter is chosen as key factors, while the remaining factors are marginalized out. The loop factor is added according to the procedure that will be described in section 8.1.

Every time the filter and smoother is updated the multiple iterations of an optimization is repeated until the estimate hits a minimum. The current value of the system state are extracted and stored after every filter update before the pose is published over a ros topic.

It was first experimented with only using the iSAM2 algorithm. This works well until a loop closure was detected. In this case the runtime increases drastically for the optimization because the factor graph now performs a full bundle adjustment with both landmarks and poses. The optimization run time took a while to stabilize after this because a large amount of new factors are added for the next iteration. Hence, the next optimization takes a while to. It was therefore experimented with marginalizing all landmarks whenever a loop was detected, however the optimization together with the marginalization still took a while, thus allowing multiple new factors to be added. This resulted in a stable, but inefficient solution. An alternative was using pose graph optimization, but this sacrificed a lot of precision. The aforementioned concurrent filtering and smoothing approach was therefore chosen. The concurrent solution yielded much faster and stable optimization run times, even during loop closures.

Algorithm 10: Main body of VSLAM backend. The algorithm serves as a summary of the full description in this section.

Input: New factors, $\boldsymbol{f}'$, and new values, $\mathcal{X}'$. Potentially, a loop factor, $\boldsymbol{f}^{loop}$
Output: Optimized world pose $\{\mathbf{x}^w\}^*$. Refined landmarks position in world frame $\{\mathbf{l}^w\}^*$.
Nomenclature: $\boldsymbol{f}^t$ denotes filter factors and $\boldsymbol{f}^R$ denotes smoother factors.
Nomenclature: $\mathcal{X}^t$ denotes filter values and $\mathcal{X}^R$ denotes smoother values.
Nomenclature: $\mathcal{X}^S$ separator of filter and smoother.

```
1  while $\boldsymbol{f}^t, \mathcal{X}^t \leftarrow$ UpdateFilter($\boldsymbol{f}'$, $\mathcal{X}'$) do
2      $\boldsymbol{f}^K, \mathcal{X}^K \leftarrow$ IdentifyKeyFactors($\boldsymbol{f}^t$, $\mathcal{X}^t$)
3      MarginalizeNonActiveFactors($\boldsymbol{f}^t$, $\mathcal{X}^t$)
4      if $\Delta t >$ 1second then
5          $\boldsymbol{f}^R, \mathcal{X}^R \leftarrow$ UpdateSmoother($\boldsymbol{f}^K, \mathcal{X}^K$)
6          if Loop is detected then
7              MarginalizeAllNonKeyFactors($\boldsymbol{f}^t$, $\mathcal{X}^t$)
8              UpdateSmoother($\boldsymbol{f}^{loop}$, _)
9          $\mathcal{X}^S \leftarrow$ Synchronize($\boldsymbol{f}^t, \mathcal{X}^t, \boldsymbol{f}^R, \mathcal{X}^R$)
10     $\{\mathbf{x}^w\}^*, \{\mathbf{l}^w\}^* \leftarrow$ getCurrentEstimate($\mathcal{X}^t$)
       return: {Output}
```

8.1 VSLAM Factor Processing

The callback function subscribes to relative pose estimates between sequential image pairs from the frontend. These relative pose esimates are inserted into the pending factor graph container as a `BetweenFactor<Pose3>` before they are added to the filter. This equals $\boldsymbol{f}^{VO}$ in fig. 8.0.2. The factor is connected to the previous visual odometry pose in the factor graph and uses the relative motion to create an estimate of the of the current pose. The noise of the `BetweenFactor<Pose3>` was approximated based on the RMSE for the pose relatives compared to the associated relative of the ground truth, followed by some additional tuning.

Landmarks are inserted into the factor graph for every keyframe. This balance is chosen to ensure that a computationally sustainable amount of features are added, while still relating a sufficient number of landmarks across keyframes. Before landmarks are added, they first have to be transformed into the world frame using the current estimate of the pose, yielding $\mathbf{l}^{\mathrm{w}}$. The uncertainty of the pose grows over time, therefore the pose in which the landmarks was first detected should provide the most accurate estimate of the position for the landmark. This assumption is not necessarily always true, for example if the distance to the landmark is very large and the baseline is small, but the assumption should however hold based on the triangulation procedure described in section 7.2.4. The world position of the landmark is then stored in a temporary container together with the feature position in both the left and right image. Features that only reside in one keyframe may fortify the pose estimate of that frame without necessarily improving it. On the other hand, features that reappear in multiple keyframes provide a lot more information that connects the motion over multiple frames. Therefore, before landmarks are added to the factor graph they are first compared to the previous keyframe to ensure that they appear in the previous frame. If this is the first time the landmark appear, an initial estimate of the landmark position is first added to the values. Then, using the `GenericStereoFactor<Pose3, Point3>` the landmark, $\mathbf{l}^w$, is reprojected with the extrinsic poses of both stereo cameras, yielding the factor operation expressed in eq. (8.1.0.1). This factor minimizes the reprojection error of $\mathbf{l}^{\mathrm{w}}$ compared to the associated feature position, $\mathbf{u}$ in both left and right images.

$$f^{\text{stereo}}\left(\mathbf{T}_{c_l w}, \mathbf{l}^w\right) = \left\|\pi\left(\mathbf{l}^w; \mathbf{T}_{c_l w}, \mathbf{K}_l\right) - \mathbf{u}_l\right\|^2_{\Sigma_{left}} + \left\|\pi\left(\mathbf{l}^w; \mathbf{T}_{c_r c_l} \cdot \mathbf{T}_{c_l w}, \mathbf{K}_r\right) - \mathbf{u}_r\right\|^2_{\Sigma_{right}} \quad (8.1.0.1)$$

This is the stereo specific factor equivalent of equation 5.2.1.4 inserted into the factor graph. Opposed to the optimizations problems described in sections 7.2.5 and 7.3.2 this factor takes two arguments that are optimized, i.e. both the landmark position and the pose. This optimization strategy is called *full bundle adjustment.*

As an alternative to stereo factor, the *smart factor* projection equivalent were also experimented with. The smart factor is a concept introduced by Carlone *et al.* (2014), developed to provide a general framework for variable elimination in factor graphs. The idea requires the factor graph to be divided into variables that are of explicit interest and variables that are only necessary for inference. Thus providing an efficient structure in addition to being robust to unstable factors as they are simply ignored. However, having a different formatting than regular factors, smart factors ended up not being used as it was never worked out how the smart factors were removed from the concurrent filter. Because smart factors and loop closures do not comply, it was therefore chosen to rather use `GenericStereoFactor<Pose3, Point3>` and handle unstable landmarks by removing them. The noise of the `GenericStereoFactor<Pose3, Point3>` was approximated to the reprojection error for every 3D point followed by some fine tuning.

A loop closure constraint is also added to the factor graph whenever this is detected with the frontend. The loop closure constraint connects a prior pose match to the current pose using the `BetweenFactor<Pose3>`. The relative transformation between the loop matches is computed in the frontend and inserted with the factor. The factor is then inserted into the long-term smoother.

8.2 GNSS Factor Processing

The GNSS measurement function is given by equation 2.7.0.1. Since only one state variable is involved in the measurement function, the equation below defines a unary factor

$$f^{gnss}\left(\mathbf{x}_t\right) \triangleq \left\|\mathbf{z}_t^{gnss} - h^{gnss}\left(\mathbf{x}_t\right)\right\|^2_{\Sigma_{gnss}} \tag{8.2.0.1}$$

which is the factor equivalent of equation 5.2.1.4 inserted into the factor graph. GTSAM has a prebuilt `GPSFactor<Point3>`, however, this only supports translation and not rotation. Because the RTK-GNSS system contains accurate information on both the translation and rotation, the `PriorFactor<Pose3>` is therefore preferred.

The GNSS is assumed to be online for the factor graph to be initialized, thus the GNSS measurement corresponding to the first image pair is inserted to the backend as a PriorFactor. This defines the world frame, $\mathcal{F}_W$. After the initial measurement is inserted, if the GNSS is online, each GNSS measurements are associated to the current world pose in the factor graph with the closest matching timestamp. The noise added to the PriorFactor will be a very small multivariate Gaussian noise as the RTK-GNSS is accurate down to a few centimeters.

8.3 IMU Factor Processing

GTSAM has implemented the IMU preintegration procedure described in section 2.6.1 under the `PreintegratedCombinedMeasurements` class. To initialize this class the covariance of the accelerometer and gyroscope, as well as the covariance of their biases have to be defined. An initial estimate of the bias values also have to be placed. The noise was initially based on sensor data, but had to be increased by tuning. Lastly, the transformation from the IMU to the body frame, often referred to as the IMU lever arm, has to be provided. The body frame was set to the left camera of the stereo setup for the lever arm.

The preintegration procedure integrates linear acceleration and angular velocity measurements from the IMU by embedding the measurements into the kinematic model in equation 2.6.1.1. An approximation of the relative rotation, translation and velocity is thus obtained which can be inserted into the factor graph as the prebuilt `CombinedImuFactor`. The mathematics that facilitates the preintegration operation is described in section 2.6.1. However, to embed the preintegrated measurement model in equation 2.6.1.10 into the factor graph, the residuals of equation 5.2.1.4 have to be defined. Since measurement noise is assumed zero-mean and Gaussian up to first order $\left[\delta\boldsymbol{\phi}_{ij}^\top, \delta\mathbf{v}_{ij}^\top, \delta\mathbf{p}_{ij}^\top\right]^\top \sim \mathcal{N}\left(\mathbf{0}_{9\times1}, \boldsymbol{\Sigma}_{ij}\right)$, the residual errors can be defined as

$$
\begin{aligned}
\mathbf{e}_{\Delta \mathrm{R}_{ij}} &\doteq \log\left(\left(\Delta\tilde{\mathrm{R}}_{ij}\left(\bar{\mathbf{b}}_i^g\right)\operatorname{Exp}\left(\frac{\partial\Delta\bar{\mathrm{R}}_{ij}}{\partial\mathbf{b}^g}\delta\mathbf{b}^g\right)\right)^\top \mathrm{R}_i^\top \mathrm{R}_j\right)\\
\mathbf{e}_{\Delta \mathbf{v}_{ij}} &\doteq \mathrm{R}_i^\top\left(\mathbf{v}_j-\mathbf{v}_i-\mathbf{g}\Delta t_{ij}\right)\\
&\quad-\left[\Delta\tilde{\mathbf{v}}_{ij}\left(\bar{\mathbf{b}}_i^g,\bar{\mathbf{b}}_i^a\right)+\frac{\partial\Delta\bar{\mathbf{v}}_{ij}}{\partial\mathbf{b}^g}\delta\mathbf{b}^g+\frac{\partial\Delta\bar{\mathbf{v}}_{ij}}{\partial\mathbf{b}^a}\delta\mathbf{b}^a\right]\\
\mathbf{e}_{\Delta \mathbf{p}_{ij}} &\doteq \mathrm{R}_i^\top\left(\mathbf{p}_j-\mathbf{p}_i-\mathbf{v}_i\Delta t_{ij}-\frac{1}{2}\mathbf{g}\Delta t_{ij}^2\right)\\
&\quad-\left[\Delta\tilde{\mathbf{p}}_{ij}\left(\bar{\mathbf{b}}_i^g,\bar{\mathbf{b}}_i^a\right)+\frac{\partial\Delta\bar{\mathbf{p}}_{ij}}{\partial\mathbf{b}^g}\delta\mathbf{b}^g+\frac{\partial\Delta\bar{\mathbf{p}}_{ij}}{\partial\mathbf{b}_a}\delta\mathbf{b}^a\right]
\end{aligned}
\tag{8.3.0.1}
$$

in which the bias updates of equation 2.6.1.11 was also included. The bias can be readily included in the factor graph, as an additive residual to equation 5.2.1.4 for all consecutive keyframes as

$$
\left\|\mathbf{e}_{\mathbf{b}_{ij}}\right\|_\Sigma^2 \doteq \left\|\mathbf{b}_j^g-\mathbf{b}_i^g\right\|_{\Sigma^g}^2+\left\|\mathbf{b}_j^a-\mathbf{b}_i^a\right\|_{\mathbf{\Sigma}^a}^2 . \tag{8.3.0.2}
$$

These estimates are then refined with the iSAM2 optimization scheme and the optimized states will work as priors for the next iteration. For the first iteration the prior values of the velocity and bias does not exist. The velocity is approximated as the relative pose of the preintegrated interval divided by the elapsed time, i.e. $velocity = \frac{dx}{dt}$. The result is then rotated into the world frame and inserted as a `PriorFactor<Vector3>`. The initial bias' does however have to be tuned and inserted as a `PriorFactor<imuBias::ConstantBias>`. A constant bias model is assumed, however the magnitude of the bias is gradually adjusted with the factor graph optimization. A reasonable initial guess of the bias may therefore be determined by running the entire system and setting the bias as the stabilized result.

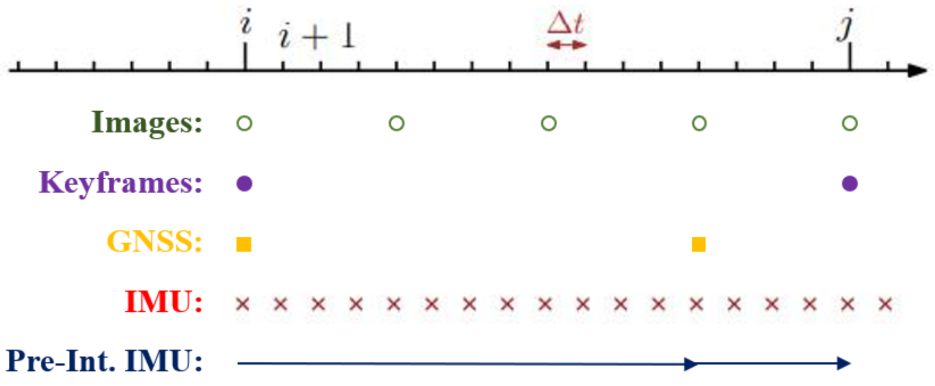


Figure 8.3.1: Different rates for visual odometry, GNSS and IMU. These are not the exact rates used in this system, however it represents how keyframes, GNSS measurements and preintegration is used in the system.

There is a small processing delay with slam frontend solutions. The newest IMU measurements will therefore most likely not correspond to the relative motion estimated by the odometry. The IMU measurements are rather stored. Whenever the factor graph is updated with a measurement from another sensor, for example GNSS or VSLAM, the stamps of the previous pose and the recently added pose are then compared to the stored IMU measurements. All measurements that exist within this time interval is then preintegrated and inserted into the factor graph.

CHAPTER 9

Data Logging

Data was logged in the area of *Brattøra* in Trondheim illustrated in figure 9.0.1. This is a harbor for small boats in the outlet of *Nidelva*. The area is surrounded by docked boats with a train line on one side of the harbor and buildings on the other side. milliAmpere2 is intended to traverse the passage of region 1 in figure 9.0.1 between *Brattørskaia* and *Ravnkloa*. Sequences recorded within this area was directed at testing the intended traversed path. Several more sequences was also logged in the surrounding area of this passage (region 2) to analyze motion over larger regions. Lastly, region three was also included as this basin spans a larger and more open scenery.

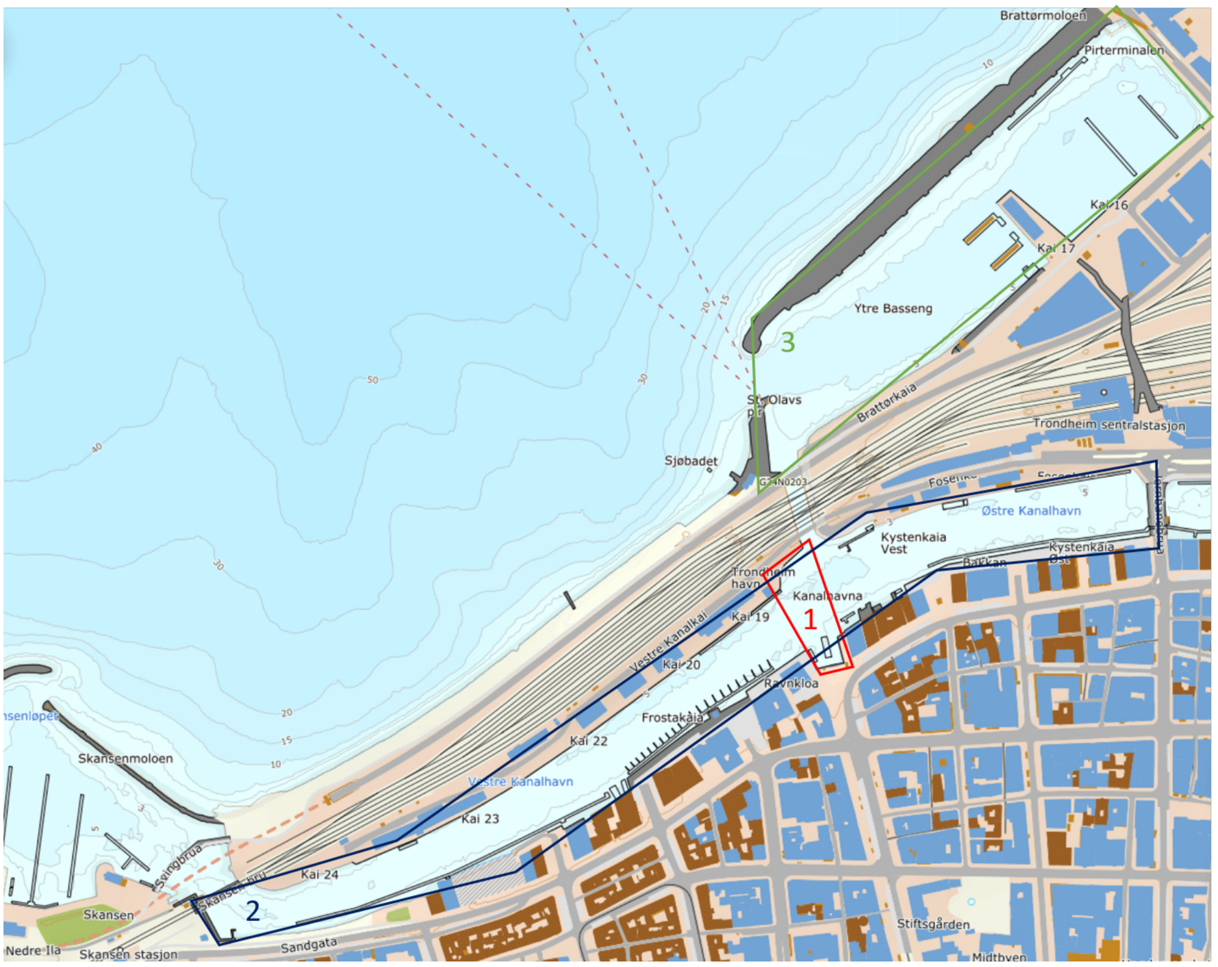


Figure 9.0.1: The image shows an overview of the area surrounding Brattøra in Trondheim. Three regions are marked in map retrieved from *Kartverket.no* to illustrate where the data was recorded. Region *one*, marked in red, depicts the area of the intended operational region for milliAmpere2. This area is marked with more details in fig. 6.1.1. Region *two* covers *Kanalhavna*, which is the area with the most closely related typology to the operational region. Region *three* is *Ytre Basseng*, which is a more spacious basin.

Data was recorded throughout a whole week in April. Preparations and testing of sensors and other equipment was the focus for Monday-Wednesday. On Thursday eight sequences was recorded that was directed at SLAM scenarios. Consequently, approximately static environments was a priority for these scenarios - no large moving objects in the foreground of the scene. On Friday data was recorded for two other thesis' (Gerhardsen, 2021) and (Auestad, 2021), where the focus was directed at fiducial markers and a detection of a GNSS-tagged vessel with known position. Figure 9.0.2 captures the scene that was recorded, together with the setup that was used. Figure 1.1.1 illustrates the scene facing the other side of the harbor.

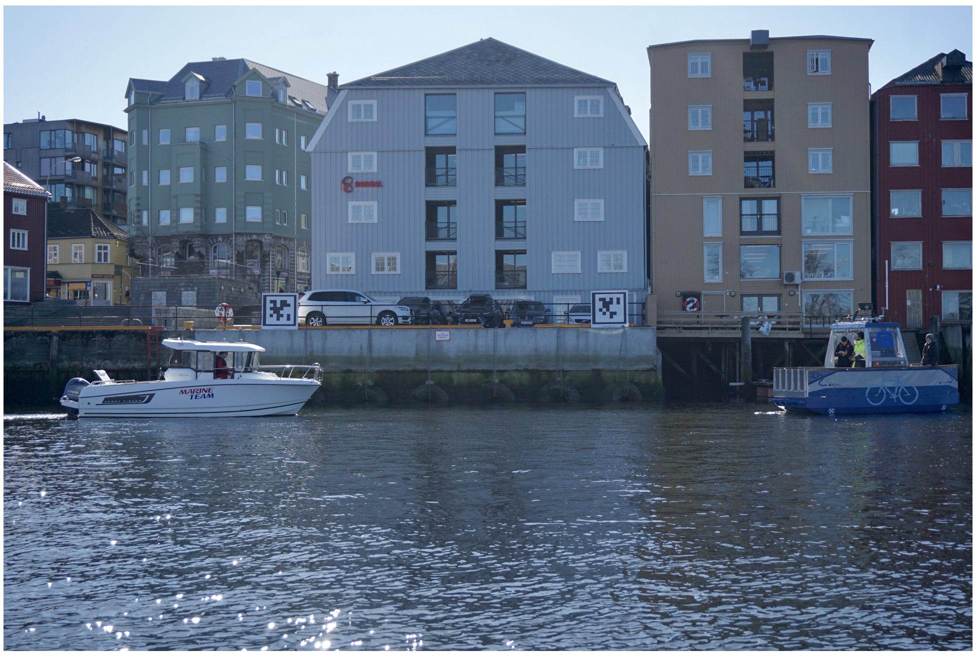


Figure 9.0.2: Situational image that captures milliAmpere in the harbor environment together with the setup that was used during the data collection for the other thesis'. Image captured by the author.

9.1 Camera Calibration

Camera calibration is the process of estimating the intrinsic, extrinsic, and lens-distortion parameters of a camera. Extrinsic parameters have to be determined if multiple cameras are available and facing the same direction. Using the intrinsic and extrinsic parameters, lens distortion can be correct for. There exist different calibration techniques, where the two main ones are:

- *Photogrammetric calibration*: This method uses a calibration object whose geometry in 3D space is known. The object usually consists of two or three orthogonal planes. Alternatively, a plane undergoing a precisely known translation can sometimes also be used.
- *Self-calibration*: No calibration object is used for this procedure. The camera is rather moved in a static scene where the rigidity of the scene constraints the camera parameters. Assuming fixed internal parameters, correspondences between three images are used to recover the intrinsic and extrinsic parameters so that 3D structure can be reconstructed up to a similarity.

Zhangs method is a calibration technique developed by Z. Zhang (2000) at Microsoft Research. It combines ideas from both of the two main calibration strategies described above, where only 2D metric information is used rather than 3D or purely implicit one. The technique only requires the camera to observe a planar pattern, typically a checkerboard, at a minimum of two different orientations. For both images features are detected. Because the plane $z = 0$ is given by the pattern itself, only 2D metric information is needed. Linear transformation between planes are computed so that an initial estimate of the intrinsic parameters can be computed using Singular Value Decomposition. Onwards, the parameters are refined by minimizing the algebraic error using the Levenberg-Marquardt algorithm. The same procedure is applied to find an estimate of the extrinsic parameters. For the lens distortion the tangential distortion is ignored. An estimate of the radial distortion parameters is calculated by comparing the real pixel values and the ideal ones given by the pinhole model. The reprojection error often used as a measure of the quality of the calibration.

9.1.1 Camera Calibration for the milliAmpere Stereo Rig

The stereo camera rig was calibrated both Thursday and Friday prior to the data recording using the ROS stereo camera calibration tool (Wise, 2009). However, when the image data was later analysed it was found that the calibration was insufficient. The intrinsic and distortion parameters varied a lot between the cameras, and a large reprojection error of > 2 were experienced when the calibration was retried with the original data on a later occasion. In comparison, when working with the KITTI data the reprojection error is < 0.02. Thus, this rendered the milliAmpere stereo calibration from the harbor utterly useless. It was therefore decided that a new calibration should be performed at NTNU approximately a week after the data logging. This calibration was performed successfully, giving a reprojection error of $0,04$. It should however be noted that the stereo rig had been exposed to several transportation stages in between the data logging and the second calibration. The potential consequence of this will be discussed in more detail in section 10.1. The extrinsic and intrinsic parameters resulting from the final calibration can be seen in table 9.1 and table 9.2 followed by the uncertainty of each parameter.

Extrinsic Parameters					
Translation [mm]			Rotation [deg]		
X	Y	Z	X	Y	Z
-1740.235 ± 0.114	-9.591 ± 0.028	87.338 ± 0.597	0.688 ± 0.001	5.712 ± 0.001	0.733 ± 0.001

Table 9.1: Extrinsic calibration parameters for milliAmpere stereo rig in camera coordinates.

Intrinsic Parameters						
	Left Camera			Right Camera		
f_u	1236.1239 ± 0.2641			1236.7399 ± 0.2676		
f_v	1235.4177 ± 0.2636			1236.9865 ± 0.2662		
u_0	620.1205 ± 0.6002			642.7828 ± 0.5872		
v_0	534.8395 ± 0.1968			530.3827 ± 0.1881		
Rad.dist.	-0.398	0.234	-0.113	-0.400	0.243	-0.131
Tan.dist.	-0.0002		-0.0005	-0.0003		0.0005

Table 9.2: Intrinsic calibration parameters for milliAmpere stereo rig.

CHAPTER 10

Experimental Results and Discussion

In this chapter, the results from the application of the milliAmpere and KITTI data are discussed. First the developed system applied to milliAmpere in the harbor environment is detailed. This is followed by an analysis of the developed system tested on the KITTI dataset for different urban environments. The analysis using the KITTI dataset involves discussion of the system with different sensor configurations, succeeded by a comparison of the results obtained from other SLAM algorithms. The results are presented for several scenarios, where one representative sequence is discussed in detail for a direct visual comparison. For the remaining results, tendencies from metric results are compared and discussed. All analysis of the developed algorithm were generated on a laptop with an octa-core AMD Ryzen 7 4800H processor running at 2,9 GHz with 16 GB of memory. The laptop is also equipped with a NVIDIA GeForce RTX 2070 GPU. The results was generated from running experiments once on every sequence, with only one application running on the computer. All plots were generated using the trajectory evaluation toolbox (Z. Zhang and Scaramuzza, 2018).

10.1 Developed System Evaluated on the milliAmpere Dataset

In this section, discussions are provided on the developed VSLAM algorithm, described in sections 7 and 8, applied to the logged milliAmpere data. The sequences are recorded in the harbor environment as described in chapter 9, where all recorded sequences starts and ends in approximately the same position. The ground truth of these data sequences are generated by 5Hz RTK-GNSS measurements. IMU measurements are extrapolated from the previous GNSS measurement following the alpha beta-filter procedure described by Bar-Shalom *et al.* (2004). Ground truth data is thus provided at a frequency of 100Hz.

The resulting trajectory from one of the sequences is depicted in fig. 10.1.1. The result show promising tendencies of correct motion, but the trajectory is however of too poor quality to be useful in its current state. It is seen that the algorithm particularly struggles with estimating rotation. The translation estimate from frame to frame is not perfect, but at the same time not the main weakness of the final estimate. However, the erroneous rotation estimates propagate throughout the entire trajectory, causing an overall drift, thus giving a final error of 55 meters. Drift is common for visual odometry, however not to the extent experienced with this scenario, where the end result is insufficient. When analysing the developed VSLAM system applied specifically to the milliAmpere setting, several reasons for the insufficient trajectory result were discovered. However, the main issue is rooted in an incorrect stereo camera calibration. This, and other error sources along with attempts to fix symptoms of the errors will be discussed next.

It is argued that the main issue experienced with the milliAmpere sequences is caused by the stereo calibration. In section 9.1.1 the original stereo calibration at the day of recording was

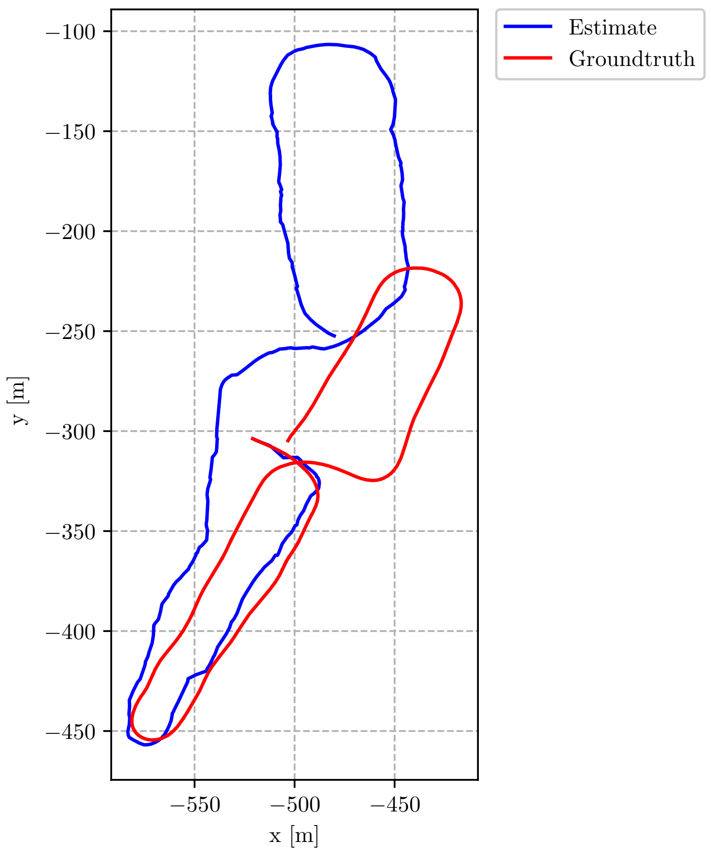


Figure 10.1.1: Results from a representative sequence from the milliAmpere data. The estimated trajectory by the VSLAM and the ground truth RTK-GNSS date are aligned at the beginning. The latitude and longitude from the GNSS are transformed to meters, setting the world frame of the motion.

rendered completely useless, thus necessitating the second calibration performed at NTNU. Even though this second calibration achieves a reprojection error of $0,04$ during calibration, this was not the case when the second calibration result was applied to the recorded slam sequences. In reality the rectified images had a reprojection error of $\langle 0.4, 1.0 \rangle$ for all triangulated stereo matches. An interesting observation was that the reprojection error was by far most dominant vertically in the image. For the left image the reprojection showed a positive vertical displacement, while for the right image the displacement was the same. but negative. Because the reprojection error furthermore varied between feature correspondences it is most likely caused by a vertical offset in the rotation of the calibration, perhaps in combination with the vertical translation error between the stereo cameras. To pinpoint the exact reason for this discrepancy more time would be required than available. However, a direct complication is that the stereo cameras were exposed to several transportation stages in between the data logging and the second calibration. The stereo rig was removed from the ferry every evening and remounted every morning in addition to being transported to and from NTNU. Stereo calibration is an extremely sensitive matter, thus (small) mechanical shocks and vibrations from car transportations may impact the calibration.

A bad calibration affects several of the submodules composing the VSLAM motion estimate. Aside from the triangulation, the bundle adjustment procedures are greatly affected. Because the discrepancies in all calibration matrix, projection matrices and relative transformation are affecting the bundle adjustment, the basis for these optimizations are wrong. Both the multi-frame bundle adjustment and IMU fusion was tried in the backend, but GTSAM quickly returned `IndeterminantLinearSystemExceptions` for unstable factors. The reason being inaccurate landmarks that were compared over multiple frame and inaccurate initial motion estimates. For the frontend, the optimization sometimes completed successfully so that a simple frame-to-frame estimate of the trajectory could be calculated. When the optimization were unsuccessful, the previous relative motion was used, argued by smoothness in section 7.3.1. This simplification did however not hold when the motion estimate failed over multiple sequential frames as can be seen during the turn at the position $[-575, -455]$ in fig. 10.1.1. This was also the case for at the position $[-545, -355]$, where milliAmpere was angled to the left in this frame. Even though the heading was corrected in the next frame, the motion estimate failed, thus never correcting the heading estimate. Because optimization problems generally are very prone to wrong calibrations, a potential improvement that should be researched is including the camera calibration in the factor graph optimization of the backend. GTSAM provides predfined factors for this purpose. Doing online self calibration, the system should improve its overall robustness to changes in calibration caused by mechanical shocks, etc. That being said, it would probably not solve the initial calibration discrepancies of this magnitude.

The estimated trajectory also appears jagged. The reason for this is that the frame rate originally was recorded at 20Hz. Because milliAmpere is moving very slowly at $\approx 3m/s$ the initial feature based motion estimate, calculated using the `findEssentialMat()`, is not easily distinguishable when the relative transformation shows such small displacements. Therefore it was tried to drop the frame rate to 5Hz which stabilized this issue to some extent, however resulting in more jagged motion caused by larger steps for each relative. A better alternative would be to skip the `findEssentialMat()` procedure all together and rather replace it with a motion prediction model. This will be listed as a potential improvement to the system. Because the motion prediction only serve as an initial estimate to be refined with the bundle adjustment it could serve as a more stable prior, especially with low velocities. What also could be even more interesting would be to refine this model based motion prior using *JET*, described in section 6.3.8. JET achieved state-of-the-art rotation estimates at the time of release. These two improvements should in theory together both improve the initial motion estimate and the feature tracking. It should be noted that the initial intention of this thesis was to solve the feature tracking and initial motion estimate this way, but due to compatibility issues caused by outdated software versions of JET, this plan was never fulfilled.

Another complication that had to be accounted for was non-static content in the harbor environment. During the data logging it was ensured that no moving objects was situated in the foreground of the scene. Water, and more specifically waves, was however an issue. Most feature that were incorrectly tracked from the previous to the current image in the water were generally filtered out using the `findEssentialMat()` function, where RANSAC caught outliers. However, potentially as a result of the insufficient calibration, some of the incorrectly tracked features in the water remained. It was observed that such scenarios destabilized the bundle adjustment. The first and most obvious fix would be a recalibration so that RANSAC would have the potential to only remove moving objects. Furthermore, segmentation could be applied to mask out the water of the image. Segmentation is however prone to illumination changes, but alternative segmentation techniques such as U-Net (Ronneberger *et al.*, 2015) using deep learning have achieves great segmentation results. This is however a more computationally extensive task. A completely different approach could be to use keypoints from LiDAR scans to obtain the 3D landmarks and then reproject these onto the images in a similar matter as V-LOAM (J. Zhang *et al.*, 2015). This is proposed because LiDAR scans requires hard surfaces to reflect the laser scans. Consequently, water and waves would not be reflected. This was however not tested because it would require a complete restructure of the system, but will rather be included as a very interesting modification to be explored.

Lastly, in addition to the recalibration there are two additional adjustments that should be considered the next time data logging is performed. Firstly, the loop closure in fig. 10.1.1 was never discovered. The reason for this was simply how the data was recorded. The vehicle has to be facing approximately the same direction over some sequential frames. Because neither were the case, the loop closure was never discovered. To facilitate appearance based place recognition, the recorded trajectory should be reappearing over approximately the same area, with approximately the same heading over some sequential frames. An alternative adjustment to the algorithm would be to search for loop closures using the full 360 °camera rig mounted on milliAmpere. The second adjustment that should be considered is the baseline of the stereo rig. The baseline was set very large (see table 9.1) to facilitate a higher accuracy of features detected at long ranges (Pinggera *et al.*, 2014). However, this comes at the expense of feature matching for objects closer in the scene. With a large baseline it is much harder to find these feature correspondences. A smaller baseline between 0.5m and 1m is therefore proposed. To compensate for the reduced resolution for feature displacements a more accurate stereoscopic method for distance estimation of landmarks should be included, for example as proposed by Pinggera (2018). Either way, it is argued that a more robust depth estimation procedure should be included, to ensure that the basis for scaled motion estimates are correct for both close and distant 3D points.

Mainly as a result of bad calibration the full multi-frame visual-inertial SLAM system was reduced to a simple frame-to-frame visual odometry module in the milliAmpere setting. It would be very interesting to evaluate the performance of the developed system with a proper stereo calibration, but because the simplified module did not achieve very satisfying results with the current calibration, additional evaluation of the other recorded sequences will not be included in the discussion. The described improvements are still relevant and will therefore be included under future work section 11.2.

10.2 Developed System Evaluated on the KITTI Dataset

Evaluation was also conducted on the KITTI dataset to enlighten the actual potential of the developed algorithm. Because of the similarity in sensor configuration between milliAmpere and the KITTI dataset, and the fact that the data comes pre-calibrated, rectified, and synchronized, the KITTI dataset was considered as a good substitution to the milliAmpere dataset as an alternative benchmark for testing. The similarity in sensor configuration should prove valuable when other sensors such as LiDARs are added to the multi-sensor SLAM system in future work. All modifications discussed in section 10.1 are ignored and the system is included in its entirety as detailed in sections 7 and 8. The ground truth of the dataset is provided by the on board INS-system at a rate of 10Hz.

First, the result from different sensor configurations are included to demonstrate and discuss their effect on the overall trajectory. The VSLAM solution will be active for all scenarios, hence the frontend thread described in chapter 7 will be active for all sensor configurations. The general backend threads of the filter and smoother will also be active for all scenarios, where the availability of each sensor will vary. First only the VSLAM is enabled, then both VSLAM and IMU are online, before the GNSS is finally added to the joint sensor configuration. The sensor configuration will be detailed in each of the following subsections. Lastly, the developed system is compared to two other visual odometry/SLAM systems for comparability, LIBVISO2 and ORB-SLAM2. It was decided to only visualize one representative sequence so that the different sensor configurations and SLAM/VO-systems could be easily compared visually. Metric results are also available for all sequences to evaluate additional general tendencies.

10.2.1 VSLAM

In this section only the VSLAM factors detailed in section 8.1 will be embedded in the factor graph. The traversed trajectory is depicted in fig. 10.2.1, where the error of the translation and rotation are individually depicted in figures 10.2.2a and 10.2.2b. All generated trajectories are transformed with the SE3 transformation that minimizes the overall ATE, before the error is calculated.

From the figures it is observed that a quite accurate estimate of the motion is achieved, particularly considering the extent of the traversed distance. The scale and the translation part of the transformation are generally estimated very well. The position error in the plane, i.e. for x and y, depicted in fig. 10.2.2a is a bit jagged where the trajectory follows a general curve with additive perturbations. One of the reasons this happens is the uncertainties of the landmarks and features when their reprojection error is minimized at every keyframe. It is also observed some inaccuracies in the calculated alignment used by the trajectory evaluation software, causing some of the spikes in fig. 10.2.2b.

From fig. 10.2.2b it can be seen that erroneous rotation is triggered at a few frames. The erroneous rotation estimate then propagates to the rotation and position over the next frames until a loop closure is found. A very visual example of the propagating error is seen over the last stretch to the left in fig. 10.2.1 from coordinate $[-20, 450]$ to $[0, 20]$. In fig. 10.2.2b it is seen that a wrong roll and pitch estimate causes the vehicle to be traversing upwards, thus affecting the scale of the motion in the plane. The scenarios that trigger the rotation errors is isolated to

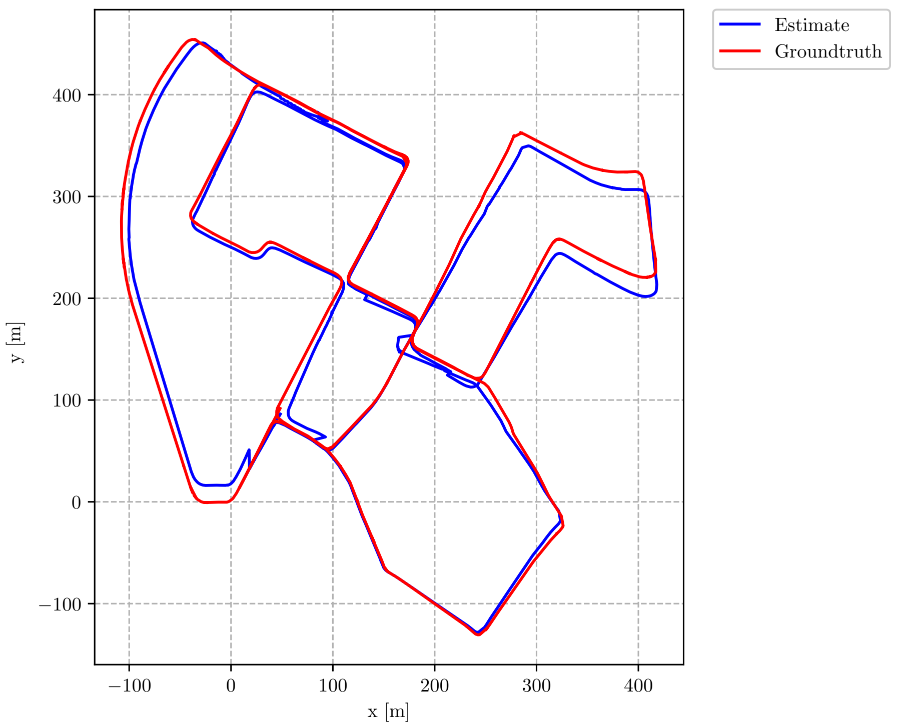
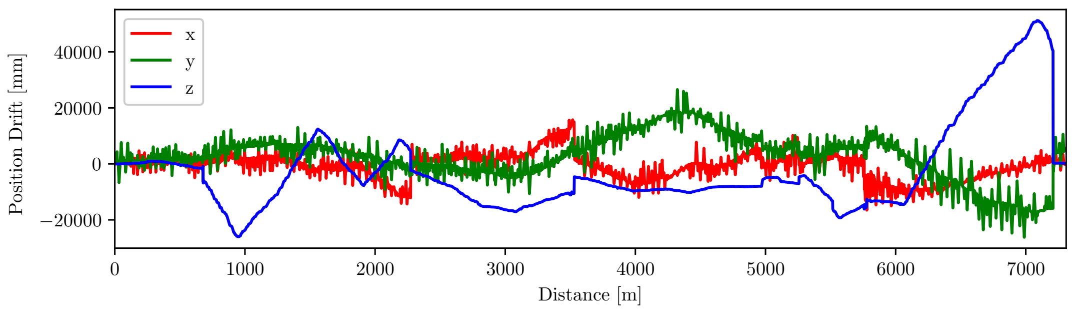


Figure 10.2.1: VSLAM

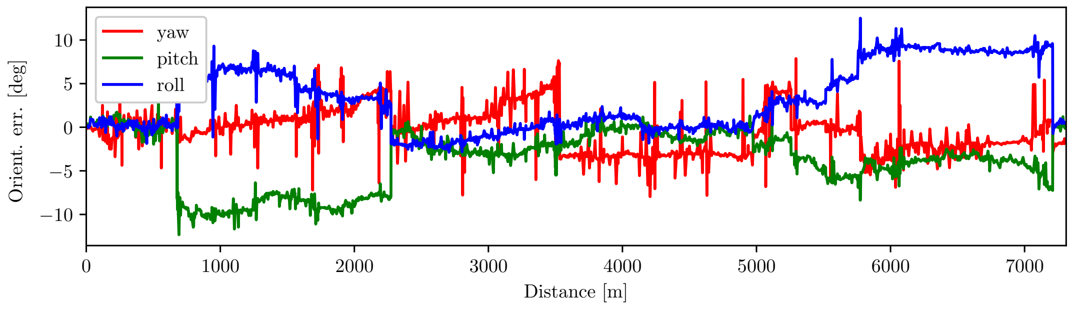
the following:

- As stated in section 7.3.1, motion estimates by the Nistér-5-point algorithm was replaced by the previous transformation in combination with new rotation estimates by PYR, when the vehicle was moving very slowly. However it was still observed that RANSAC sometimes filtered out very many features when the vehicle was moving slowly, but was still active. This happens at approximately 700m in fig. 10.2.2b, equaling $[180, 350]$ in fig. 10.2.1, and triggers the rotation error as a result of a poor optimization with fewer features. This issue was discussed in the previous section, and an exact threshold on the velocity for when RANSAC should be disabled was never precisely determined. This issue could however be avoided by rather using a model based motion prediction. The model based pose prediction could for example be used to find feature correspondences in a similar matter as ORB-SLAM (Mur-Artal, Montiel, *et al.*, 2015). ORB-SLAM uses the predicted motion to initiate search regions for descriptor based matches individually. The predicted motion could then be optimized using motion-only bundle adjustment. The alternative previously mentioned would be using JET to find correspondence matches jointly under an epipolar constraint and thus refine the predicted motion estimate simultaneously.

- While RANSAC generally excluded features from moving objects, it was observed that slowly moving objects in the distance not always were ignored. For example, after approximately 2700m in fig. 10.2.2b, equaling $[300, -50]$ in fig. 10.2.1, two slow moving cyclists are traversing the scene from left to right; consequently provoking the a left shift compared to the true yaw estimate. In the trajectory this equals the bottom right corner of fig. 10.2.1. This also happens just before the top left corner of the trajectory, with yet another bicycle. The bicycle is moving towards the vehicle, thus causing the shift in roll and pitch



(a) Translation error in millimeters. x, y and z denotes motion in the body frame with respect to the world frame as described in section 2.1.



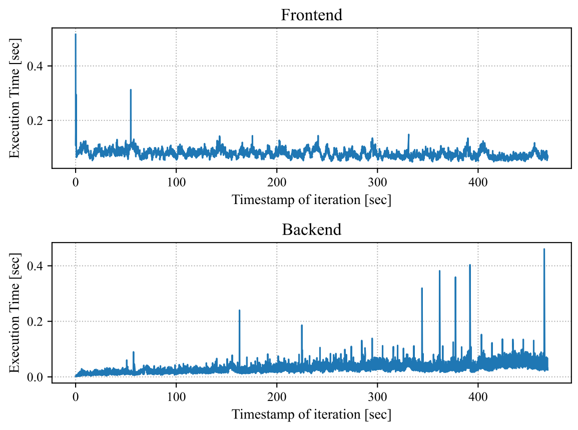
(b) Rotational error in degrees for yaw, pitch and roll.

Figure 10.2.2: Propagating errors throughout the 00 sequence of KITTI using only VSLAM.

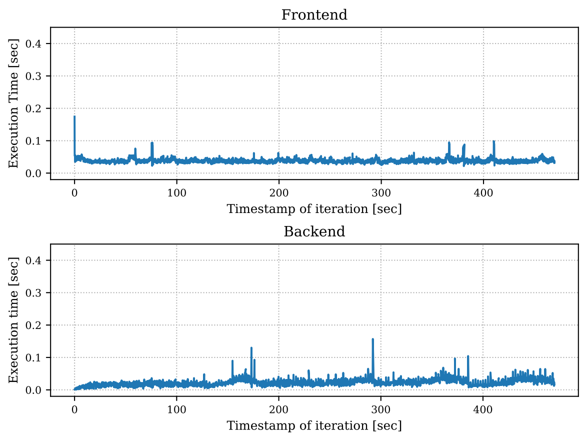
for $> 5500m$ in fig. 10.2.2b. This equals the stretch from $[-20, 450]$ to $[0, 20]$ to the left in fig. 10.2.1. Alternatively to RANSAC, more careful feature detection could be applied, so that moving objects rarely or never are detected. This could for example be done by checking the feature detection against the phase shift of the optical flow at different sub regions of the frame. The computational complexity of such a modification should however be considered. Another alternative would be to use object detection, for example using deep learning. Deep learning algorithms achieves very accurate results for object detection, but to operate with minimal delay they should be combined with an object tracker (Zhao *et al.*, 2019). A proposed solution to this is included in future work (section 11.2). It should also be noted that inclusion of other sensors might reduce the error of the wrong estimate, but the cause of the issue then still remains. Object detection is therefore considered one of the key improvements that should be performed.

- The last issue that was observed was small erroneous shifts in pitch when the car hit speed bumps. This could have been avoided by identifying and ignoring frames which contains speed bumps, for example by using PYR to identify abrupt changes in pitch. This could also be relevant in the ferry scenario to avoid waves causing discrepancies in pitch.

Loop closures are successfully identified with no false positives. They are furthermore successfully integrated in the long-term smoother and correctly closing loops in the separate smoother thread. It is seen that all accumulated drift is removed and the pose is set equal to the prior. The overall execution time of the backend depicted in fig. 10.2.3a is observed as approximately constant with an average execution time of 0.0226s. The execution time do show some spikes occurring at loop closures. This is simply because the filter and smoother is synchronized one second after a loop closure. Thus, if the smoother execution exceeds one second the filter is delayed until the smoother has completed the optimization. The filter and smoother was synchronized regularly with one second delay to quickly update the separator of the factor graph in case of loop closures. The delay from the small waiting period did not affect subsequent filter updates either, but the spikes of the execution time could be removed by synchronizing at a lower frequency. The execution time of the frontend, also depicted in fig. 10.2.3a, remains fairly constant with an average processing of 0.0608s. Some spiking are observed whenever large amounts of new features have to be detected, for example with the first frame. With GPU enabled the spiking of the frontend was largely reduced and the average processing time was down to 0.0429s. A reduction of spikes for the backend was also observed, but this may be caused by minor discrepancies between the two runs or because the GPU takes some of the general load from the CPU. Regardless, it is seen that the system overall has a very efficient structure with an average execution time of 0.0655s with GPU enabled.



(a) Without GPU enabled.



(b) With GPU enabled.

Figure 10.2.3: Execution time for frontend and backend of the VSLAM system.

10.2.2 VSLAM and IMU

In this section the both the VSLAM factors detailed in section 8.1 and the IMU preintegration detailed in section 8.3 was embedded in the factor graph. The traversed trajectory is depicted in fig. 10.2.4, where the error of the translation and rotation are individually depicted in figures 10.2.5a and 10.2.5b.

With the IMU extending the factor graph configuration discussed in the previous section, much more consistent motion estimates were observed. The estimated trajectory now follows the ground truth with minimal errors throughout almost the entire trajectory. In fig. 10.2.5a fewer spikes in the position estimates are observed, since the additional sensor information from the IMU reduces the uncertainty of the joint estimate. Furthermore, the IMU preintegrates angular rates so that additional information is provided for the rotation. Consequently, the rotation error observed in fig. 10.2.5b is somewhat more centered around zero compared to when only the VSLAM was active.

Rotation estimation, especially at low velocities has been one of the main challenges for the VO backbone of the developed system. The inclusion of additional information on rotation from the IMU is therefore very welcomed. Additionally it is seen that the erroneous impact from speed bumps and low motion estimates are largely ironed out. Still, the impact from the slow moving objects discussed in section 10.2.1 remain. Even with the IMU activated, the two cyclist scenarios still affect the estimated trajectory in fig. 10.2.4. The reoccurring error of moving objects is therefore considered one of the key improvements that have to be made. In table 10.1 and table 10.2 the ATE and standard deviation of the trajectory in fig. 10.2.5a is listed. It is seen that the score is lower than expected, considering the demonstrated accuracy throughout the major part of the trajectory in fig. 10.2.5a. This clearly demonstrates the propagating influence caused by disturbances from moving objects.

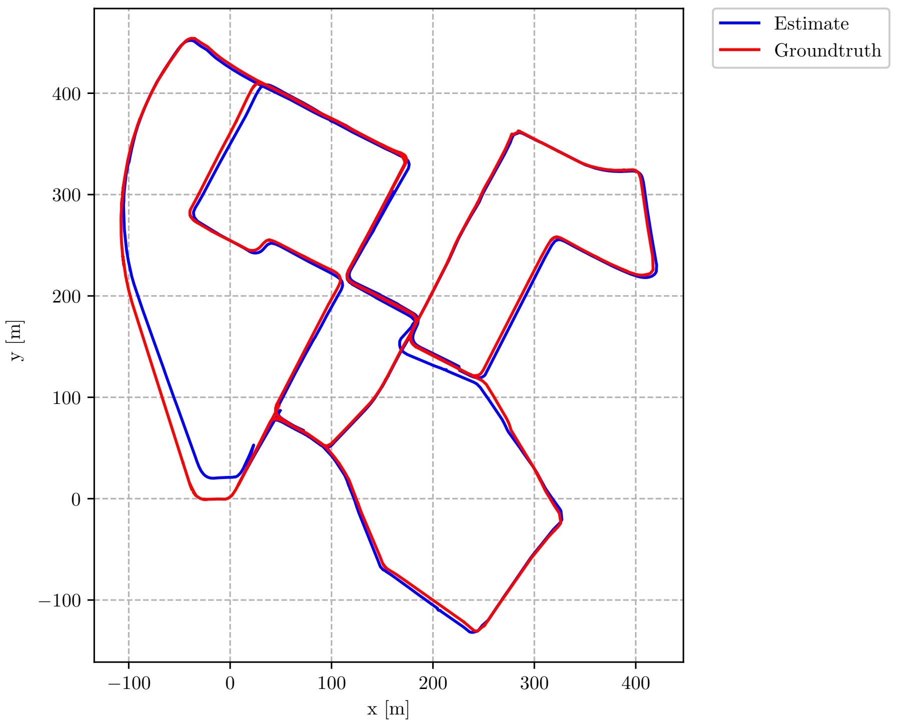
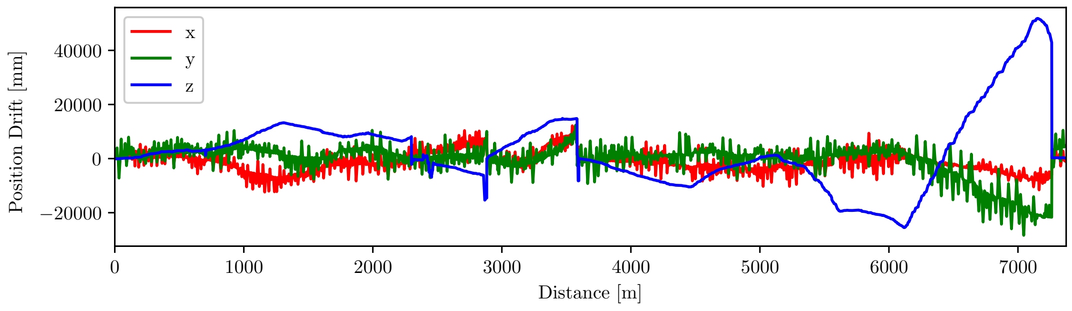
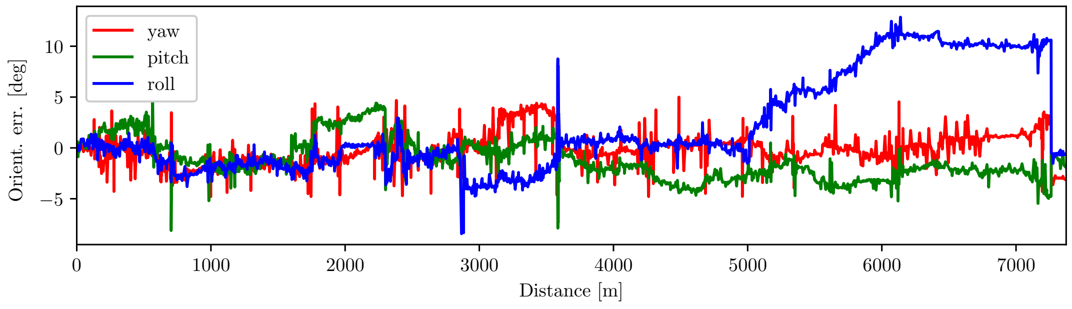


Figure 10.2.4: Estimated trajectory when VSLAM and IMU i activated.



(a) Translation error in millimeters. x, y and z denotes motion in the body frame with respect to the world frame as described in section 2.1.



(b) Rotational error in degrees for yaw, pitch and roll.

Figure 10.2.5: Propagating errors throughout the 00 sequence of KITTI using both VSLAM and IMU .

10.2.3 VSLAM, IMU and GNSS

In this section the all of the factors detailed in sections 8.1, 8.3 and 8.2 was embedded in the factor graph. The traversed trajectory is depicted in fig. 10.2.6. The trajectory is perfectly estimated, which makes sense as GNSS measurements are frequently added at 1 Hz. Consequently, the global measurements are frequently correcting local drift, while the visual-inertial SLAM module provides accurate motion estimates in between. This enables the vehicle to report accurate trajectory estimates relative to a global reference frame. No further analysis will be conducted on this sensor configuration, but the illustration were rather included to demonstrate that GNSS factors are successfully added to the sensor configuration. One improvement that however will be advertised is embedding a motion model that are used to connect factors if the IMU is not available. Currently, if the IMU is offline, odometry and GNSS measurements are rather associated to the closest odometry measurement.

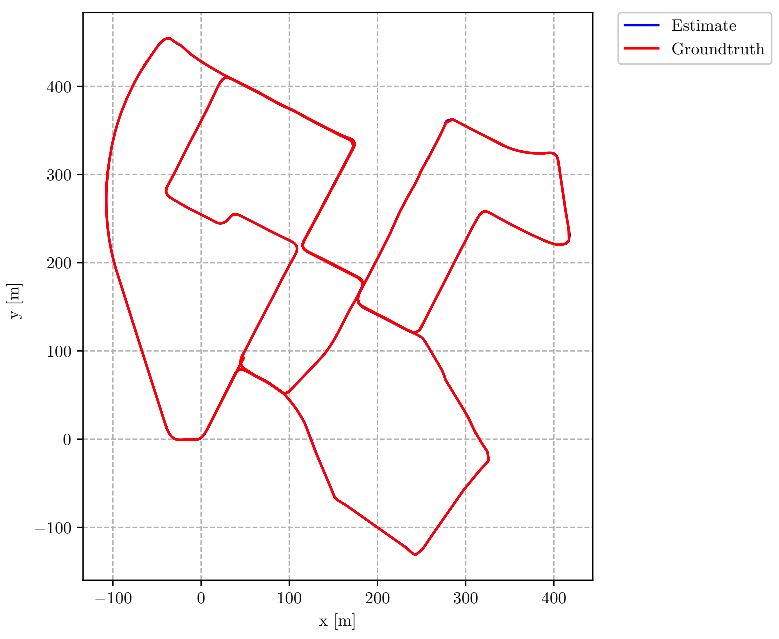


Figure 10.2.6: Estimated trajectory when VSLAM, IMU and GNSS i activated.

Figure 10.2.7 depicts the point cloud of all the individual landmarks that were detected throughout the traversed trajectory. Note that landmarks only shortly exist within the filter window, meaning that all the landmarks depicted in fig. 10.2.7 are stored in a separate point cloud container when they are removed from the filter. Landmarks that were removed from the filter are only stored for visualization purposes, and displayed in a separate thread.

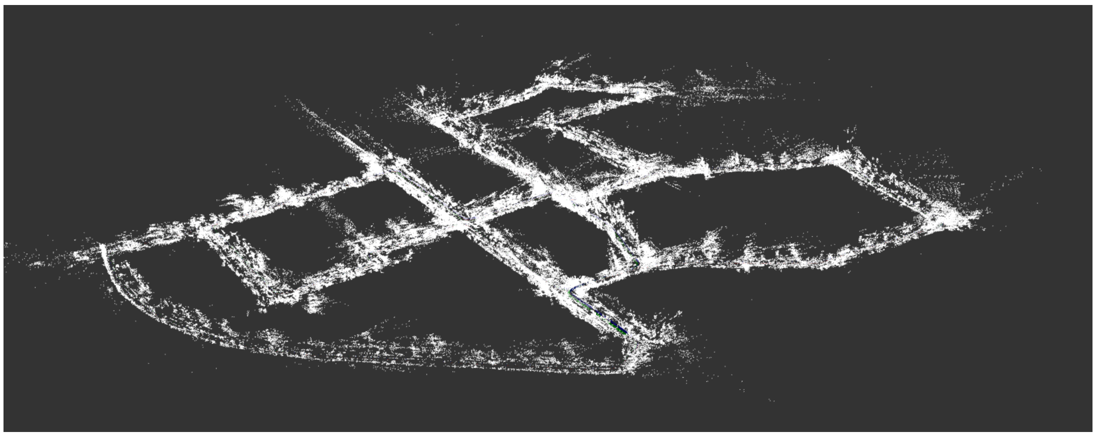


Figure 10.2.7: Point cloud of all detected landmarks throughout the traversed trajectory.

10.2.4 Developed System Compared to LIBVISO2 and ORB-SLAM2

To evaluate the performance of the Visual-Intertial sensor combination of the developed SLAM system, it was compared with the well known frame-to-frame VO method, LIBVISO2 (Geiger, Ziegler, *et al.*, 2011). LIBVISO2 was chosen because of the resembling approach for the visual odometry part of the system. The ATE for the position estimates by ORB-SLAM2 is also included to demonstrate the performance of one of the the current state-of-the-art VSLAM algorithms available, and thus put the performance of the developed system in perspective. Tables 10.1 and 10.2 compares the Absolute Trajectory Error (ATE)/RMSE of aligned trajectories and standard deviation computed using the trajectory evaluation tool by Z. Zhang and Scaramuzza (2018). The results were generated for the 10 training sequences from the KITTI dataset (Geiger, Lenz, Stiller, and Urtasun, 2013b), that had public ground-truth available. Sequence 03 did not have available IMU data and was therefore not included in the evaluation. The results were generated for the developed system, and for LIBVISO2. The ATEs of ORB-SLAM2 were pulled from the ORB-SLAM2 publication (Mur-Artal, Montiel, *et al.*, 2017).

10.2.4.1 LIBVISO2

LIBVISO2 (Geiger, Ziegler, et al.*, 2011)*: is a fast feature-based VO library for both monocular and stereo cameras. It extract features by filtering the images with a corner and blob mask followed by non-maximum and non-minimum suppression on the filtered images. Features are then matched on subsequent frames using the circular matching procedure described in section 7.2.3. Outliers are removed using RANSAC and the egomotion is then estimated by minimizing the reprojection error from frame to frame using Gauss-Newton on the remaining matches.

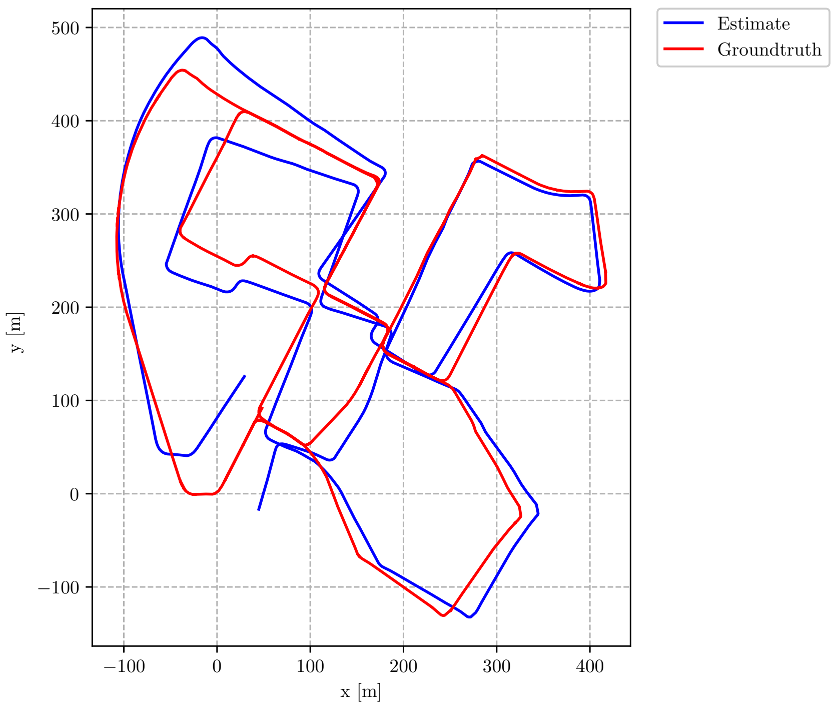
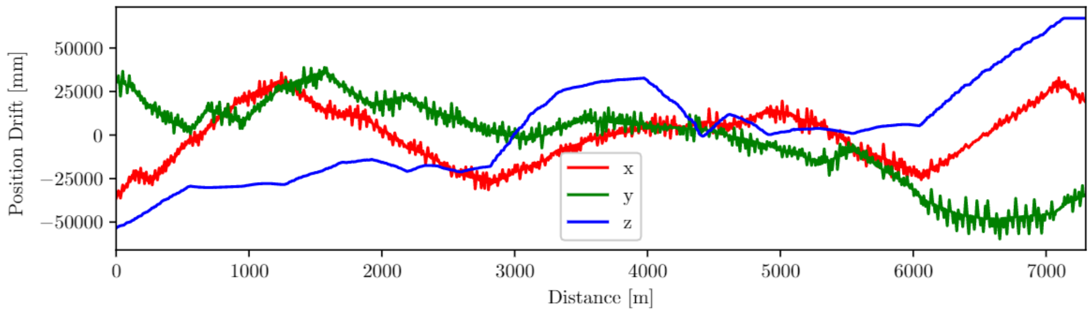
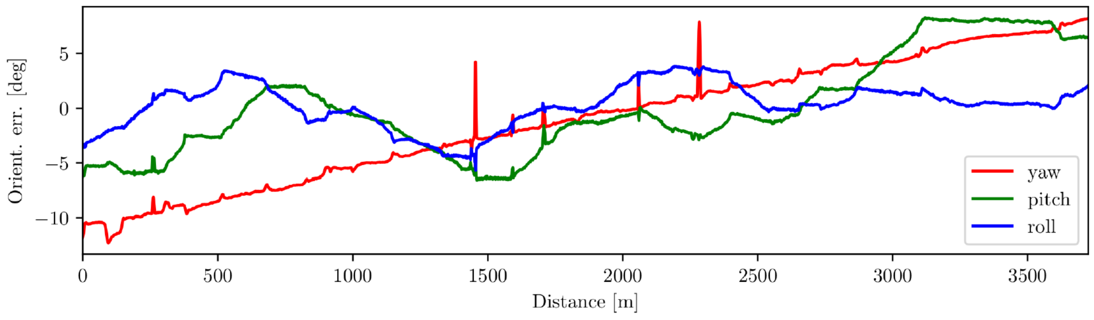


Figure 10.2.8: Estimated trajectory by LIBVISO2. The trajectory is aligned with the ground truth using the SE3 transformation that minimizes the overall ATE.



(a) Translation error in millimeters. x, y and z denotes motion in the body frame with respect to the world frame as described in section 2.1.



(b) Rotational error in degrees for yaw, pitch and roll.

Figure 10.2.9: Propagating errors of LIBVISO2 throughout the 00 sequence of KITTI.

Figure 10.2.8 depicts the estimated trajectory by LIBVISO2 for the 00 sequence. When compared to the trajectory computed by the developed VSLAM in fig. 10.2.1, it is seen that the developed system is far better at estimating the rotation compared to LIBVISO2. One of the key reasons is that the developed system includes multiframe bundle adjustment in the backend. The addition of IMU to the developed system in fig. 10.2.4 also improves the result substantially. There are seen some spikes in both of the compared estimates. It is assumed that they are rooted in alignment inaccuracies with the ground truth because the spike stabilize to the prior error after the spike. Being a frame-to-frame algorithm there is no reason why LIBVISO2 should first get a large error and then return to the prior error afterwards. During testing it was also experienced an error in the ground truth of the 00 sequence, with repeated measurements from the INS system on the vehicle. It is therefore not unrealistic that this may happen on more than one occasion. From fig. 10.2.9 it is furthermore seen that LIBVISO2 has a constant bias in the yaw estimates adding to in the overall displacement.

From figures 10.1 and 10.2 it is seen that the developed visual-inertial SLAM algorithm outperforms the ATE of LIBVISO2 for almost all sequences, both containing loop closures and not. For most sequences the developed system has a far lower positional ATE and a slightly smaller ATE on the rotation. There are however two main exception where LIBVISO2 wins by a large margin: sequence 01 and 04. These sequences have two similar characteristics in that they are following a more sparsely textured, approximately straight highway. There is additionally a lot traffic along these highways. On sequence 01, the developed system and LIBVISO2 accumulate a large error, but the developed system shows much larger errors. This is mainly rooted in the fact that when cars pass by a lot of features gets detected on those objects. Consequently, the algorithm includes passing objects in the motion estimate. Because the scene is so sparsely textured, features attached to vehicles are not always filtered out using RANSAC. This leads to several events where the developed VO system under-estimates the translation, or calculates a slightly wrong rotation. For the developed system the overall impact increases because the erroneous stereo features are included over multiple frames. This is a related issue to the one discussed in section 10.2.1, now happening at a larger scale. The same thing happens with the 04 sequence, but for this sequence the camera is stuck behind a large truck, obstructing much of the scene.

While generally outperforming LIBVISO2 on all other marks, it was seen that the standard deviation of the rotation in table 10.2 generally is lower for LIBVISO2 than the developed system. One of the reasons being that the rotation estimates for LIBVISO2 "oscillates" less. The reason for these perturbations for the developed system were discussed in section section 10.2.1. An additional reason for the larger standard deviation is that the developed system is very prone to moving objects, which happens in all sequences to a varying degree. Loop closures is a factor that reduces the overall ATE, but does not affect the standard deviation to the same extent. Thus, when loop closures then corrects the accumulated drift of the developed system the standard deviation still remains. This should therefore be considered when observing that the rotation ATE was lower for the developed system, but that the standard deviation generally was lower for LIBVISO2.

		Developed System		LIBVISO2		ORB-SLAM2
Sequence	Length [m]	ATE $_{\text{pos}}$ [m]	σ[m]	ATE $_{\text{pos}}$ [m]	σ[m]	ATE $_{\text{pos}}$ [m]
00*	3724	**8.38**	**3.95**	30.88	15.16	1.3
01	2454	56.98	23.57	**38.37**	**7.96**	10.4
02*	5066	**31.33**	17.64	39.23	**16.42**	5.7
03	560	–	–	–	–	–
04	392	5.235	2.49	**0.94**	**0.39**	0.2
05*	2204	**4.54**	**2.96**	12.10	6.99	0.8
06*	1232	**3.16**	**1.88**	4.63	1.93	0.8
07*	694	**2.34**	**1.53**	5.54	3.51	0.5
08	3222	**10.46**	**7.69**	21.32	8.76	3.6
09	1704	**11.26**	**4.58**	16.91	10.52	3.2
10	918	**3.78**	2.29	4.07	**1.79**	1.0

Table 10.1: ATE and standard deviation of the position estimates for 10 of the training sequences from the KITTI dataset (Geiger, Lenz, Stiller, and Urtasun, 2013b). The results were generated for the developed system and LIBVISO2. Sequences followed by a star marks trajectories that contain loop closures. The developed system is compared to LIBVISO2 where the better result is marked as bold. For ORB-SLAM2 the ATE were retrieved from Mur-Artal and Tardós (2017) and reprinted by permission from ©[2017] IEEE.

		Developed System		LIBVISO2	
Sequence	Length [deg]	ATE $_{\text{rot}}$ [deg]	σ[deg]	ATE $_{\text{rot}}$ [deg]	σ[deg]
00*	3724	**4.93**	**2.15**	7.53	3.15
01	2454	9.86	**3.34**	**8.62**	4.01
02*	5066	**6.19**	3.83	8.88	**3.47**
03	560	–	–	–	–
04	392	4.92	2.28	**1.50**	**0.10**
05*	2204	**5.01**	3.77	6.46	**2.28**
06*	1232	**3.39**	2.19	3.66	**1.76**
07*	694	**3.56**	2.74	4.78	**1.75**
08	3222	**6.46**	4.74	7.67	**2.75**
09	1704	**7.44**	4.55	8.01	**2.81**
10	918	4.63	2.91	**3.98**	**1.50**

Table 10.2: ATE and standard deviation of the rotation estimates for 10 of the training sequences from the KITTI dataset (Geiger, Lenz, Stiller, and Urtasun, 2013b). The results were generated for the developed system and LIBVISO2. For ORB-SLAM2 they were retrieved from Mur-Artal and Tardós (2017). Sequences followed by a star marks trajectories that contain loop closures. The developed system is compared to LIBVISO2 where the better result is marked as bold.

10.2.4.2 ORB-SLAM2

ORB-SLAM2 (Mur-Artal and Tardós, 2017): is another feature-based VSLAM library for both monocular and stereo cameras, capable of real-time operations. The library has one thread that localizes the camera at every frame and also decides when to add new keyframes. This is done by an initial feature matching between two frames using FAST features and ORB descriptors, followed by motion-only bundle adjustment. A second thread performs local mapping, which processes new keyframes and performs local bundle adjustment over covisible points. New correspondences are updated in a covisibility graph, while older keypoints with few matches are marginalized out. The last thread searches for, and performs, loop closures following the procedure described in section 6.3.9. Whenever a loop is detected the pose matches and landmarks are connected by a similarity transform, before a pose graph optimization is performed. This thread launches a fourth thread to perform full BA after the initial pose graph optimization.

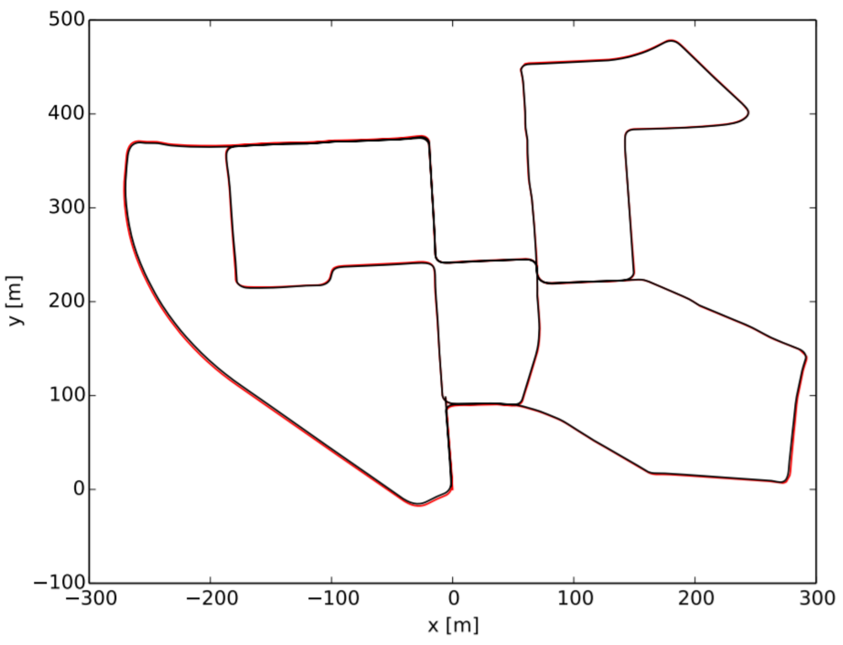


Figure 10.2.10: Estimated trajectory by ORB-SLAM2 retrieved from Mur-Artal and Tardós (2017) and reprinted by permission from ©[2017] IEEE.

ORB-SLAM2 was included in the discussion mainly to demonstrate the performance of one of the best VSLAM algorithms available, and thus put the performance of the developed system in perspective. The results are gathered from Mur-Artal and Tardós (2017). From fig. 10.2.10 it is seen that ORB-SLAM2 estimates the trajectory very well. This coincides with their calculated translation ATE in table 10.1. It is seen that ORB-SLAM2 performs better by a good margin on the position ATE for all sequences. It should however be noted that Krombach *et al.* (2018) performed a comparison their own algorithm to ORB-SLAM2. In their analysis ORB-SLAM2 achieved slightly worse results that presented in the original paper. Regardless, ORB-SLAM2 is one of the best algorithms available and the better precision is therefore expected.

The second reason why ORB-SLAM2 is included is to emphasize two weaknesses of the developed system. ORB-SLAM2 separates detected 3D points based on their depth. They use far points ($40 \times baseline$) only for information on rotation while closer points are used for both rotation and translation. This could stabilize the uncertainty of 3D points greatly. The other inspiration would be to not marginalize out inactive landmarks that are no longer tracked. Similarly as ORB-SLAM2, inactive key points that has a lot of associations could be stored in a map and potential correspondences could be search for in this map. In doing so the system could be able to associate key points over larger traversed distances. From table 10.1 it is seen that also ORB-SLAM2 struggle with sequence 01 compared to the other sequences. This is also rooted in many moving objects in the sparsely textured scene. However, erroneous estimates is perhaps stabilized to a larger extent by these two approaches.

CHAPTER 11

Conclusions

11.1 Concluding Remarks

In this thesis, a GPU-accelerated feature-based stereo VSLAM frontend was developed utilizing both structure-only and motion-only bundle adjustment for initial frame-to-frame motion relatives. Both stereo measurements and initial motion estimates are embedded in a short-term smoother using iSAM2 as the underlying optimization paradigm, thus incrementally performing full multi-frame bundle adjustment over a confined window. Measurements from IMU preintegration and GNSS are furthermore fused with the VSLAM data in the factor graph optimization. Concurrently, loop closures are detected and included in a long-term smoother. The full system is embedded in ROS, thus fitting nicely with the milliAmpere sensor and software interface.

The developed system was validated on the renowned KITTI dataset (Geiger, Lenz, Stiller, and Urtasun, 2013b). Analysis showed that the VIO part of the system outperforms the popular stereo VO system LIBVISO2 (Geiger, Ziegler, *et al.*, 2011) on most tested sequences. When loop closures are extended to the VIO module, the performance was improved further, and in most cases greatly exceeded the performance of LIBVISO2. The trajectory estimates were additionally compared to one of the current state-of-the-art solutions, ORB-SLAM2 (Mur-Artal and Tardós, 2017), to emphasize weaknesses demonstrated by some edge cases. While achieving good results, it was seen that developed system struggles more during lower motion and was sometimes unable to ignore features from slow moving objects.

A new dataset was recorded using an extended sensor setup on milliAmpere, descibed in section section 6.1, where an additional stereo camera rig was included. Unfortunately, the developed system was unsuccessfully tested in its proper state mainly due to problems with the stereo calibration. It was experienced that the inaccuracy of the calibration propagated through many subsequent submodules of the frontend. Consequently, the initial motion estimate and 3D point positions were too inaccurate, thus destabilizing inclusion of additional sensors in the factor graph optimization of the backend. Some stable initial motion estimates were produced by the frontend, but these were to few for a precise estimate of the motion of the vehicle. To the extent of the authors knowledge, the analysis presented in this thesis is however one of the first surveys of visual SLAM applied in a maritime harbor environment.

11.2 Future Work

Based on the results and discussions from chapter 10, several improvements, extensions and profoundly interesting complementary future research topics to this thesis can be identified. A list containing summaries of the most prominent research topics for the developed system, both directed specifically to the milliAmpere setting and general remarks, that should be the subject of future work is included.

- Stereo calibration: In order to utilize stereo cameras for navigation in the milliAmpere setting, correct stereo calibration is a necessity. This can be obtain by recalibrating the cameras in the harbor environment if new data are collected. A larger chessboard should the printed as trouble with range specific part of the calibration was experienced. Alternatively, an online calibration of the camera parameters be could embedded in the factor graph optimization as proposed by Elisha *et al.* (2017). This would be a very relevant and interesting improvement regardless of the recalibration. The latter alternative enables reuse of the collected data. If new data should be recorded it is furthermore suggested that a smaller baseline is used to simplify feature matching of stereo matches closer to the vehicle.
- The stereo projection factors could be replaced by the smart factor equivalent (Carlone *et al.*, 2014). Smart factors provides a more robust alternative, where unstable 3D points are ignored. While overall sacrificing some precision, the exclusion of unstable 3D points might stabilize jagged motion in the overall trajectory estimate.
- Currently only poses are transferred to the long-term smoother. It is rather proposed that not all 3D points are culled, but that essential 3D points that are observed in many keyframes are also transferred to the long-term smoother. This matches the approach taken by ORB-SLAM (Mur-Artal, Montiel, *et al.*, 2015). In this case the execution time of loop closures performed by the long-term smoother would increase. The rate of the synchronizations would in this case have to be reduced so that the the short-term smoother wouldn't have to wait for the loop closure to finish.
- False loop closure detections were never experienced throughout this thesis. However, a false loop closure could potentially have fatal results for the overall trajectory estimate. Sünderhauf *et al.* (2012) proposed switchable factor constraints that are able to recognize and reject outliers during the optimization, both in cases of general data association errors and for false positive loop closure detections. The rejection is achieved by making the topology of the underlying factor graph representation subject to the optimization rather than keeping it fixed. The software is avaliable for both GTSAM v2.0 and g^2o, but should be updated to fit the newest version of GTSAM.
- Stereo features could be triangulated in a more robust fashion. There are multiple ways of doing so. One, is by using the disparity to identify far 3D points. The 3D points should then be categorized based on depth. Far points provide good information changes in orientation, but provide weak information for translation and scale. As an example, ORB-SLAM2 (Mur-Artal and Tardós, 2017) separates far points from closer points by depth higher than 40 times the stereo baseline. This separation could largely improve the stability of the optimization where far points may provide false information on the translation. This is particularly important in the milliAmpere situation where the view stretches over larger areas. In addition the fundamental depth estimation should be improved, for example by taking inspiration from Pinggera (2018).
- The estimation of the noise models should be improved. The measurement noise for both the IMU and GNSS is available through data sheets, but for the VSLAM, the noise model,

or covariance matrices, for both relative poses and 3D points were approximated. The noise was set to the reprojection error for every 3D point, while the noise model of the relative transformations in the frontend was approximated based on the RMSE for the pose relatives compared to the associated relative of the ground truth. The noise models could be better estimated by only specifying the pixel uncertainty, and thus let that uncertainty propagate to the relative pose constraints estimated by a windowed BA procedure. An interesting alternative would be to use MLPnP (Urban *et al.*, 2016), currently used for motion-only bundle adjustment in ORB-SLAM3 (Campos *et al.*, 2020). MLPnP has the additional advantage of estimating the uncertainty of the motion.

- Both the frontend and backend would greatly benefit from having a motion model for pose estimation. For the frontend it should prove valuable to have a motion model that predict the relative transformation between frames, rather than assuming and approximately smooth motion which is refined. In the backend, measurements are currently associated either based on interconnecting factors using IMU preintegration or by the finding the timestamp matches. To account for scenarios where the time stamps does not match, and the IMU is not online, a much better solution would be to use a motion model that estimates an approximate motion relative between VSLAM poses and GNSS poses. If a constant velocity model is used these measurements could be embedded using the GTSAM `ConstantVelocityFactor`. This would also be even more relevant if other measurements such as LiDAR odometry are added to the factor graph.
- At the time of release PMO by Fanani *et al.* (2017) was ranked as the best monocular method on the KITTI odometry benchmark (Geiger, Lenz, Stiller, and Urtasun, 2012). This was achieved without loop closing mechanism, without RANSAC and also without multiframe bundle adjustment. The results were however achieved using a combination of model based pose prediction and sparse direct bundle adjustment through JET (Bradler *et al.*, 2017), discussed in section 6.3.8. The JET software is available, but require updates to fit newer versions of softwares like OpenCV. Because rotation estimates is considered one of the main weaknesses of the system, JET would therefore serves as a significant improvement to the VSLAM system.
- Modify the code by Skjellaug (2020) to fit with the developed backend. The frontend of this system can be extracted with some small modifications, but the backend of this system have to be completely restructured to fit the concurrent structure.
- Non-static objects in the scene should be removed from the image. In the milliAmpere situation this includes segmentation of water and clouds. Classical color based segmentation techniques could be applied, but deep learning based segmentation approaches like U-Net (Ronneberger *et al.*, 2015) have demonstrated better results and more robustness towards illumination changes. Other non-static objects should also be detected and ignored. This includes cars, cyclists, trains, humans and, in the milliAmpere setting, boats. In the maritime setting this could for example be solved similar to Schöller *et al.* (2019) or Hermann *et al.* (2015). Deep learning based object detectors are currently considered the state-of-the-art approach for images (Zhao *et al.*, 2019). Detections should be confirmed by radar or LiDAR detection to avoid false positives in a joint target tracker as demonstrated by Helgesen (2019) or Wolf *et al.* (2010). To provide real time capabilities, the approved

detections should be tracked separately in the images, for example using Lucas Kanade. To account for the processing delay of the deep learning, all current object tracks should be back annotated to the newest processed deep learning frame to verify if there are any new detection. New detections should then be forwarded to the newest frame containing the newest tracks. This should provide an efficient detection and tracking solution that potentially could operate concurrently with the SLAM system.

- The DBOW2 module inherently has the problem that the database structure grows over the explored area. If this is not handled, it can eventually cause problems with memory overflow and execution time.
- Relocalization in case of lost motion track could quite easily be included using the place recognition module. Whenever a location was recognized, rather than performing loop closure, a new track could simply be initiated.

In addition to the aforementioned improvements some interesting complementary research topics are suggested.

- V-LOAM (J. Zhang *et al.*, 2015), which currently achieves the second best results on the KITTI odometry benchmark, is a SLAM system that estimates the motion of the vehicle by projecting LiDAR points onto monocular images. To avoid complications by triangulation a similar approach could be taken. Detection of LiDAR keypoints could be done similarly as Skjellaug (2020). The 3D keypoints could then be reprojected onto two consequtive frames where the initial point position in the new frame is initiated by a motion model. The estimate could then be refined using JET. Using MLPnP the scale and uncertainty of the motion is obtained. This estimate could then be refined over a windowed full bundle adjustement where additional sensor measurements could also be fused in a factor graph optimization scheme. Loop closures could be detected either as performed by Skjellaug (2020) using 3D-3D matching, or using appearance based approaches as done in this thesis. Loop closures should then be performed concurrently as done in this thesis. A 3D point culling procedure similar to the previously discussed improvement should be taken to maintain a limited amount of features in a map over the traversed area. An additional interesting extention of this approach could be to project the 360°LiDAR scans onto images from the 360°camera rig mounted on milliAmpere. This procedure was initially considered for this thesis, but ignored due to the complications with the JET software discussed in section 6.3.8.
- A milliAmpere-specific research topic is a module that is targeted only at searching for place recognition within the enclosed harbor. It separates from other loop closing solutions in that potential loop matches should only be searched against a predefined/pre-recorded database of GNSS tagged images within the enclosed area. A small extension of the DBoW2 database would suffice for this purpose. To improve the robustness of potential loop candidates a majority vote can be applied using the same approach applied to all five cameras of the milliAmpere (described in section 6.1) covering a 360°field of view. The transformation from the loop detection to the GNSS tagged loop match would then be computed for all five cameras and the transformation is then added as a GNSS-like factor to the factor graph.

Bibliography

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016, ISSN: 1552-3098, 1941-0468. DOI: 10.1109/TRO.2016.2624754. [Online]. Available: http://ieeexplore.ieee.org/document/7747236/ (visited on 10/07/2020).

[2] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual-inertial and multi-map slam," *arXiv preprint arXiv:2007.11898*, 2020.

[3] E. Skjellaug, "Feature-Based Lidar SLAM for Autonomous Surface Vehicles Operating in Urban Environments," M.S. thesis, Norwegian University of Science and Technology, Jun. 2020.

[4] K. Bimbraw, "Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology," in *2015 12th international conference on informatics in control, automation and robotics (ICINCO)*, IEEE, vol. 1, 2015, pp. 191–198.

[5] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," Tech. Rep., 2015.

[6] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," en, *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, Sep. 2013, ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364913491297. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0278364913491297 (visited on 11/29/2020).

[7] J. E. Manley, "Unmanned surface vehicles, 15 years of development," in *OCEANS 2008*, IEEE, 2008, pp. 1–4.

[8] Z. Liu, Y. Zhang, X. Yu, and C. Yuan, "Unmanned surface vehicles: An overview of developments and challenges," *Annual Reviews in Control*, vol. 41, pp. 71–93, 2016.

[9] A. S. Golden and R. E. Weisbrod, "Trends, causal analysis, and recommendations from 14 years of ferry accidents," *Journal of Public Transportation*, vol. 19, no. 1, p. 2, 2016.

[10] E. Jokioinen, J. Poikonen, R. Jalonen, and J. Saarni, "Remote and autonomous ships-the next steps," *AAWA Position Paper, Rolls Royce plc, London*, 2016.

[11] K. L. Nilsen. (2017). "Førerløs fremtid for fløttmann-båten," [Online]. Available: https://trondheim24.no/nyheter/forerlos-framtid-flotmann-baten/. 10.12.20.

[12] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3d reconstruction in real-time," in *2011 IEEE intelligent vehicles symposium (IV)*, Ieee, 2011, pp. 963–968.

[13] I. Cvišić, J. Ćesić, I. Marković, and I. Petrović, "Soft-slam: Computationally efficient stereo visual simultaneous localization and mapping for autonomous unmanned aerial vehicles," *Journal of field robotics*, vol. 35, no. 4, pp. 578–595, 2018.

[14] D. Gálvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[15] S. Williams, V. Indelman, M. Kaess, R. Roberts, J. J. Leonard, and F. Dellaert, "Concurrent filtering and smoothing: A parallel architecture for real-time navigation and full smoothing," *The International Journal of Robotics Research*, vol. 33, no. 12, pp. 1544–1568, 2014.

[16] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree," en, *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 216–235, Feb. 2012, ISSN: 0278-3649, 1741-3176. DOI: `10.1177/0278364911430419`. [Online]. Available: `http://journals.sagepub.com/doi/10.1177/0278364911430419` (visited on 09/09/2020).

[17] H. P. Moravec, "Obstacle avoidance and navigation in the real world by a seeing robot rover.," Stanford Univ CA Dept of Computer Science, Tech. Rep., 1980.

[18] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, Ieee, vol. 1, 2004, pp. I–I.

[19] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[20] H. Strasdat, J. Montiel, and A. J. Davison, "Real-time monocular slam: Why filter?" In *2010 IEEE International Conference on Robotics and Automation*, IEEE, 2010, pp. 2657–2664.

[21] F. Dellaert, "Factor graphs and gtsam: A hands-on introduction," Georgia Institute of Technology, Tech. Rep., 2012.

[22] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G^{2}o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*, Shanghai, China: IEEE, May 2011, pp. 3607–3613, ISBN: 978-1-61284-386-5. DOI: `10.1109/ICRA.2011.5979949`. [Online]. Available: `http://ieeexplore.ieee.org/document/5979949/` (visited on 09/09/2020).

[23] S. Agarwal, K. Mierle, *et al.*, *Ceres solver*, `http://ceres-solver.org`.

[24] J. Dong and Z. Lv, "Minisam: A flexible factor graph non-linear least squares optimization framework," *arXiv preprint arXiv:1909.00903*, 2019.

[25] M. Doaa, A. Mohammed, M. Salem, H. Ramadan, and M. I. Roushdy, "Comparison of optimization techniques for 3d graph-based slam," *Recent Advances in Information Science*, 2013.

[26] F. Youyang, W. Qing, and Y. Gaochao, "Incremental 3-d pose graph optimization for slam algorithm without marginalization," *International Journal of Advanced Robotic Systems*, vol. 17, no. 3, p. 1 729 881 420 925 304, 2020.

[27] G. Grisetti, T. Guadagnino, I. Aloise, M. Colosi, B. Della Corte, and D. Schlegel, "Least squares optimization: From theory to practice," *arXiv preprint arXiv:2002.11051*, 2020.

[28] F. Dellaert and M. Kaess, "Factor Graphs for Robot Perception," en, *Foundations and Trends in Robotics*, vol. 6, no. 1-2, pp. 1–139, 2017, ISSN: 1935-8253, 1935-8261. DOI: `10.1561/2300000043`. [Online]. Available: `http://www.nowpublishers.com/article/Details/ROB-043` (visited on 09/09/2020).

[29] H.-P. Chiu, S. Williams, F. Dellaert, S. Samarasekera, and R. Kumar, "Robust vision-aided navigation using Sliding-Window Factor graphs," in *2013 IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany: IEEE, May 2013, pp. 46–53, ISBN: 978-1-4673-5643-5. DOI: `10.1109/ICRA.2013.6630555`. [Online]. Available: `http://ieeexplore.ieee.org/document/6630555/` (visited on 11/04/2020).

[30] L. Carlone, Z. Kira, C. Beall, V. Indelman, and F. Dellaert, "Eliminating conditionally independent sets in factor graphs: A unifying perspective based on smart factors," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 4290–4297.

[31] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd, "Real time localization and 3d reconstruction," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, IEEE, vol. 1, 2006, pp. 363–370.

[32] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, IEEE, 2007, pp. 225–234.

[33] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015, ISSN: 1552-3098, 1941-0468. DOI: `10.1109/TRO.2015.2463671`. [Online]. Available: `https://ieeexplore.ieee.org/document/7219438/` (visited on 09/09/2020).

[34] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[35] M. S. Ødven, "Lidar-Based SLAM for Autonomous Ferry," M.S. thesis, Norwegian University of Science and Technology, Jan. 2019.

[36] N. Dalhaug, "Lidar-Based Localization for Autonomous Ferry," M.S. thesis, Norwegian University of Science and Technology, Jun. 2019.

[37] M. E. Gerhardsen, "Fiducial slam for autonomous ferry," M.S. thesis, Norwegian University of Science and Technology, Jun. 2021.

[38] T. S. Hellum, *Visual odometry for autonomous ferry*, Dec. 2020.

[39] T. I. Fossen, *Handbook of marine craft hydrodynamics and motion control.* John Wiley & Sons, 2011.

[40] R. W. Beard and T. W. McLain, *Small unmanned aircraft: Theory and practice.* Princeton university press, 2012.

[41] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[42] J. Sola, J. Deray, and D. Atchuthan, "A micro lie theory for state estimation in robotics," *arXiv preprint arXiv:1812.01537*, 2018.

[43] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision.* Cambridge university press, 2003.

[44] T. V. Haavardsholm, *A handbook in visual slam. compendium in ttk21 introduction to visual simultaneous localization and mapping - vslam*, Nov. 2020.

[45] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[46] D. Nistér, "An efficient solution to the five-point relative pose problem," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 6, pp. 756–770, 2004.

[47] D. P. Curtis. (2011). "Understanding the baseline," [Online]. Available: `http://www.shortcourses.com/stereo/stereo3-14.html`. 16.05.21.

[48] P. Pinggera, D. Pfeiffer, U. Franke, and R. Mester, "Know your limits: Accuracy of long range stereoscopic object measurements in practice," in *European conference on computer vision*, Springer, 2014, pp. 96–111.

[49] T. Lupton and S. Sukkarieh, "Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 61–76, 2011.

[50] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation," Georgia Institute of Technology, 2015.

[51] R. W. Beard and T. W. McLain, *Small unmanned aircraft: Theory and practice.* Princeton university press, 2012.

[52] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*, Springer, 2006, pp. 430–443.

[53] T. Pire, T. Fischer, G. Castro, P. De Cristóforis, J. Civera, and J. J. Berlles, "S-ptam: Stereo parallel tracking and mapping," *Robotics and Autonomous Systems*, vol. 93, pp. 27–42, 2017.

[54] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*, Ieee, 2011, pp. 2564–2571.

[55] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *European conference on computer vision*, Springer, 2010, pp. 778–792.

[56] J.-Y. Bouguet *et al.*, “Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm,” Tech. Rep. 1-10, 2001, p. 4.

[57] E. Brekke, *Fundamentals of Sensor Fusion.* 2020.

[58] P. C. Mahalanobis, “On the generalized distance in statistics,” National Institute of Science of India, 1936.

[59] P. J. Huber, “Robust estimation of a location parameter,” in *Breakthroughs in statistics*, Springer, 1992, pp. 492–518.

[60] J. Nocedal and S. Wright, *Numerical optimization.* Springer Science & Business Media, 2006.

[61] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, “Svo: Semidirect visual odometry for monocular and multicamera systems,” *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 249–265, 2016.

[62] J. Engel, T. Schöps, and D. Cremers, “Lsd-slam: Large-scale direct monocular slam,” in *European conference on computer vision*, Springer, 2014, pp. 834–849.

[63] J. Engel, V. Koltun, and D. Cremers, “Direct sparse odometry,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017.

[64] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, “Factor graphs and the sum-product algorithm,” *IEEE Transactions on information theory*, vol. 47, no. 2, pp. 498–519, 2001.

[65] L. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition,” *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, Feb. 1989, ISSN: 00189219. DOI: `10.1109/5.18626`. [Online]. Available: `http://ieeexplore.ieee.org/document/18626/` (visited on 09/09/2020).

[66] P. Heggernes and P. Matstoms, “Finding Good Column Orderings for Sparse QR Factorization,” 1996.

[67] M. Kaess, V. Ila, R. Roberts, and F. Dellaert, “The Bayes Tree: An Algorithmic Foundation for Probabilistic Robot Mapping,” in *Algorithmic Foundations of Robotics IX*, B. Siciliano, O. Khatib, F. Groen, D. Hsu, V. Isler, J.-C. Latombe, and M. C. Lin, Eds., vol. 68, Series Title: Springer Tracts in Advanced Robotics, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 157–173, ISBN: 978-3-642-17451-3. DOI: `10.1007/978-3-642-17452-0_10`. [Online]. Available: `http://link.springer.com/10.1007/978-3-642-17452-0_10` (visited on 09/30/2020).

[68] R. E. Tarjan and M. Yannakakis, “Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs,” en, *SIAM Journal on Computing*, vol. 13, no. 3, pp. 566–579, Aug. 1984, ISSN: 0097-5397, 1095-7111. DOI: `10.1137/0213035`. [Online]. Available: `http://epubs.siam.org/doi/10.1137/0213035` (visited on 09/30/2020).

[69] W. M. Gentleman, “Least squares computations by givens transformations without square roots,” *IMA Journal of Applied Mathematics*, vol. 12, no. 3, pp. 329–336, 1973.

[70] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, “A column approximate minimum degree ordering algorithm,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 30, no. 3, pp. 353–376, 2004.

[71] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering," in *2011 IEEE International Conference on Robotics and Automation*, Shanghai, China: IEEE, May 2011, pp. 3281–3288, ISBN: 978-1-61284-386-5. DOI: `10.1109/ICRA.2011.5979641`. [Online]. Available: `http://ieeexplore.ieee.org/document/5979641/` (visited on 09/09/2020).

[72] L. C. K. Theimann and T. Ø. Olsen, "Stereo vision for autonomous ferry," M.S. thesis, Norwegian University of Science and Technology, Jun. 2020.

[73] M. Quigley, K. Conley, B. 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*, Kobe, Japan, vol. 3, 2009, p. 5.

[74] A. Geiger, P. Lenz, C. Stiller, and U. Raquel. (2013). "The kitti vision benchmark suite," [Online]. Available: `http://www.cvlibs.net/datasets/kitti/`. 18.11.20.

[75] 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.

[76] M. Barnada, C. Conrad, H. Bradler, M. Ochs, and R. Mester, "Estimation of automotive pitch, yaw, and roll using enhanced phase correlation on multiple far-field windows," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2015, pp. 481–486.

[77] H. Bradler, M. Ochs, N. Fanani, and R. Mester, "Joint epipolar tracking (jet): Simultaneous optimization of epipolar geometry and feature correspondences," in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, IEEE, 2017, pp. 445–453.

[78] D. Gálvez-López and J. D. Tardós. (2014). "Dbow2 open-source code," [Online]. Available: `https://github.com/dorian3d/DBoW2`. 25.05.21.

[79] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," Stanford, Tech. Rep., 2006.

[80] D. Gálvez-López and J. D. Tardós. (2014). "Dloopdetector open-source code," [Online]. Available: `https://github.com/dorian3d/DLoopDetector`. 25.05.21.

[81] T. Lindeberg, "Scale invariant feature transform," 2012.

[82] G. Golub and W. Kahan, "Calculating the singular values and pseudo-inverse of a matrix," *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, vol. 2, no. 2, pp. 205–224, 1965.

[83] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[84] H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Nature*, vol. 293, no. 5828, pp. 133–135, 1981.

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

[86] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. (2017). "Orb-slam2 open-source code," [Online]. Available: `https://github.com/raulmur/ORB_SLAM2`. 25.05.21.

[87] K. Auestad, "Stereo vision for autonomous ferry," M.S. thesis, Norwegian University of Science and Technology, Jun. 2021.

[88] M. Wise. (2009). "How to calibrate a stereo camera," [Online]. Available: `http://wiki.ros.org/camera_calibration/Tutorials/StereoCalibration`. 16.06.21.

[89] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with applications to tracking and navigation: theory algorithms and software.* John Wiley & Sons, 2004.

[90] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, Springer, 2015, pp. 234–241.

[91] J. Zhang and S. Singh, "Visual-lidar odometry and mapping: Low-drift, robust, and fast," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 2174–2181.

[92] P. Pinggera, "Stereoscopic methods for high-performance object detection and distance estimation: Extending visual environment perception for intelligent vehicles," Ph.D. dissertation, Universitätsbibliothek Johann Christian Senckenberg, 2018.

[93] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[94] N. Krombach, D. Droeschel, S. Houben, and S. Behnke, "Feature-based visual odometry prior for real-time semi-dense stereo slam," *Robotics and Autonomous Systems*, vol. 109, pp. 38–58, 2018.

[95] Y. B. Elisha and V. Indelman, "Active online visual-inertial navigation and sensor calibration via belief space planning and factor graph based incremental smoothing," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 2616–2622.

[96] N. Sünderhauf and P. Protzel, "Switchable constraints for robust pose graph slam," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 1879–1884.

[97] S. Urban, J. Leitloff, and S. Hinz, "Mlpnp-a real-time maximum likelihood solution to the perspective-n-point problem," *arXiv preprint arXiv:1607.08112*, 2016.

[98] N. Fanani, A. Stürck, M. Ochs, H. Bradler, and R. Mester, "Predictive monocular odometry (pmo): What is possible without ransac and multiframe bundle adjustment?" *Image and Vision Computing*, vol. 68, pp. 3–13, 2017.

[99] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. (2012). "Visual odometry / slam evaluation 2012," [Online]. Available: `http://www.cvlibs.net/datasets/kitti/eval_odometry.php`. 20.06.21.

[100] F. E. Schöller, M. K. Plenge-Feidenhans, J. D. Stets, and M. Blanke, "Assessing deep-learning methods for object detection at sea from lwir images," *IFAC-PapersOnLine*, vol. 52, no. 21, pp. 64–71, 2019.

[101] D. Hermann, R. Galeazzi, J. C. Andersen, and M. Blanke, "Smart sensor based obstacle detection for high-speed unmanned surface vehicle," *IFAC-PapersOnLine*, vol. 48, no. 16, pp. 190–197, 2015.

[102] Ø. K. Helgesen, "Sensor fusion for detection and tracking of maritime vessels," M.S. thesis, Norwegian University of Science and Technology, Jan. 2019.

[103] M. T. Wolf, C. Assad, Y. Kuwata, A. Howard, H. Aghazarian, D. Zhu, T. Lu, A. Trebi-Ollennu, and T. Huntsberger, "360-degree visual detection and target tracking on an autonomous surface vehicle," *Journal of Field Robotics*, vol. 27, no. 6, pp. 819–833, 2010.

Appendices

Abbreviations

Several abbreviations are used throughout this thesis, all of which are summarized here.

ASV	Autonomous Surface Vehicle
ATE	Absolute Trajectory Error
BA	Bundle Adjustment
BoW	Bag of Words
BRIEF	Binary Robust Independent Elementary Features
DBoW2	Dynamic Bags of Words 2
DOF	Degrees of Freedom
FAST	Features from Accelerated Segment Test
FOV	Field of view
GNSS	Global Navigation Satellite System
GTSAM	Georgia Tech Smoothing and Mapping library
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
iSAM2	Incremental Smoothing and Mapping 2
KITTI	Karlsruhe Institute of Technology and Toyota Technological Institute
LiDAR	Light Detection and Ranging
mA	milliAmpere
MAP	Maximum a Posteriori
NED	North-East-Down
NTNU	Norwegian University of Science and Technology
OpenCV	Open Source Computer Vision Library
ORB	Oriented FAST and Rotated BRIEF
RANSAC	Random Sample Consensus
RE	Relative/Odometry Error
RMSE	Root-Mean-Square Error
ROS	Robot Operating System

RTK-GNSS	Real-Time Kinematic-Global Navigation Satellite System
SLAM	Simultanous Localization And Mapping
VIO	Visual-Inertial Odometry
VO	Visual Odometry
VSLAM	Visual Simultanous Localization And Mapping

Mathematical Notations

Most of the relevant mathematical notations and symbols used throughout this thesis are summarized here. The remaining symbols are explained where they appear.

Physical constants

g	Gravitational constant	$9.81\,m/s^2$

Entity formats

s	Scalars (lowercase letters in italic)
$\mathbf{b}$	Vectors (lowercase letters in boldface)
$\mathbf{A}$	Linear matrices (uppercase letters in boldface)
$g(\cdot)$	Functions (lowercase letters)
$\{\cdot\}^*$	Optimized variable, vector or matrix
Δ	Linearized state vector. Sometimes followed by a descriptive symbol to specify the linearized variable. Δt is the exception, describing time between the current and previous time step.

Entry set content

$\mathbb{R}^{m\times n}$	Real Numbers of m times n entries
$\emptyset$	Empty set
$\mathbf{I}$	Identity matrix

Mathematical operations

$\mathbf{A}^T$	Matrix and vector transpose
$\mathbf{A}^{-1}$	Matrix inverse
$[\mathbf{v}]^\times$	Skew symmetric form of vector
$\lVert\mathbf{v}\rVert$	Vector norm
$\mathbf{A}\oplus\mathbf{B}$	Lie plus operator
$\mathbf{A}\ominus\mathbf{B}$	Lie minus operator
$\cup$	Union
$\cap$	Intersection
$\mathrm{Det}(\cdot)$	Determinant of matrix
$\exp(\cdot)$	Exponential of variable

$\arg\min$	The input argument that gives the function's minimum
$\arg\max$	The input argument that gives the function's maximum

Frame and transformation representations

$\mathcal{F}_a$	Frame a is the coordinate frame of which an object's pose or motion is described
$\mathbf{v}^a$	Vector expressed in $\mathcal{F}_a$
$\mathbf{R}_{ab}$	Rotation from frame $\mathcal{F}_b$ to frame $\mathcal{F}_a$
$\mathbf{t}^a_{ab}$	Translation from frame $\mathcal{F}_b$ to frame $\mathcal{F}_a$ as expressed by the subscript. Superscript denotes the frame which the motion is expressed in.
$\mathbf{T}_{ab}$	Transformation from frame $\mathcal{F}_b$ to frame $\mathcal{F}_a$

Symbol representations specific to this thesis

ξ	Random variable
$\mathbf{p}$	Position, consisting of $\{x, y, z\}$
$\mathbf{v}$	Velocity
$\mathbf{a}$	Acceleration
$\mathbf{\Theta}$	Orientation, consisting of $\{\phi, \theta, \psi\}$ representing roll, pitch and yaw respectively
ω	Angular rate
$\mathbf{x}$	Pose, consisting of both position and orientation
$\mathbf{R}$	Rotation
$\mathbf{t}$	Translation
$\mathbf{T}$	Transformation
$\mathcal{I}$	Image
$\mathbf{l}$	3D point - also referred to as landmark
$\mathbf{u}$	Image point, consisting of the horizontal and vertical coordinate $\{u, v\}$
$\mathbf{K}$	Calibration matrix of pinhole model
$\mathbf{P}$	Projection matrix
$\mathbf{z}$	Measurement
$\mathbf{b}$	Bias
η	Noise
μ	Mean
σ	Standard deviation
$\mathbf{\Sigma}$	Covariance matrix
$\mathbf{\Lambda}$	Information matrix

$\mathbf{R}$	Square root information matrix
$\mathcal{X}$	Set of variables in factor graph. A single entry is denoted by indent.
f	Set of factors in factor graph. Single entry denoted by indent.

Operations specific to this thesis

$\pi_p(\mathbf{l})$	Projection of point in 3D space $\mathbf{l}$ onto image plane
$h(\xi)$	Measurement function
$\mathcal{I}(\mathbf{u})$	Image intensity at pixel
$\mathcal{N}(\xi;\mu,\Sigma)$	Normal distribution of random variable ξ with mean μ and covariance matrix Σ
$f(\mathcal{X})$	Factor described by set variables

Thomas Schiøler Hellum

NTNU
Norwegian University of
Science and Technology