Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

Eirik Flemsæter Falck

Implementation and evaluation of 16-bit thermal-inertial odometry using non-linear factor graphs

Masteroppgave i Kybernetikk og robotikk
Veileder: Kostas Alexis
Medveileder: Nikhil Vijay Khedekar
Juni 2022

NTNU
Kunnskap for en bedre verden

Eirik Flemsæter Falck

Implementation and evaluation of 16-bit thermal-inertial odometry using non-linear factor graphs

Masteroppgave i Kybernetikk og robotikk
Veileder: Kostas Alexis
Medveileder: Nikhil Vijay Khedekar
Juni 2022

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

TTK4900 — Engineering Cybernetics, Master's Thesis

Implementation and evaluation of 16-bit thermal-inertial odometry using non-linear factor graphs

Author:
Eirik Flemsæter Falck

Supervisor:
Kostas Alexis

Co-supervisors:
Nikhil Vijay Khedekar

Master's thesis
Department of Engineering Cybernetics
Norwegian University of Science and Technology
June 6, 2022

Preface

This report concludes my master's thesis in cybernetics and robotics at the Norwegian University of Science and Technology (NTNU). All the work was done in the spring semester of 2022 and is not based on a specialization project from the previous semester.

In this work, I have based some of the code for the proposed method on the open-source repository for Semi-direct Visual Odometry (SVO)[1], created by researchers at University of Zürich and ETH Zürich. My work also relies on the factor graph modeling and solving framework Georgia Tech Smoothing and Mapping (GTSAM)[2] from Georgia Tech University. This work extended the SVO implementation to work with thermal images, and GTSAM's IMU preintegration, projection factors, and Incremental Smoothing and Mapping (iSAM2) optimizer implementations were used to create a back-end system.

All the presented work was done by me, with guidance from my supervisor, Dr. Kostas Alexis, and my co-supervisor, Ph.D. student Nikhil Vijay Khedekar. Furthermore, all figures used in this report were created by me except for the figures licensed under the Creative Commons license.

Lastly, I want to thank my supervisor, Dr. Kostas Alexis, and my co-supervisor, Ph.D. student Nikhil Vijay Khedekar, for their valuable input and discussions throughout the semester and for being available for questions and guidance when needed. Furthermore, I want to thank the team at the Autonomous Robotics Lab (ARL) for providing datasets with thermal image data.

[1]`https://github.com/uzh-rpg/rpg_svo_pro_open`
[2]`https://gtsam.org/`

Abstract

This thesis presents the adaptation and evaluation of the Semi-direct Visual Odometry (SVO) algorithm to use 16-bit thermal images and the implementation of a factor graph-based back-end. Thermal cameras can passively sense the environment, even in complete darkness and in the presence of obscurants like dust or smoke. This makes it an attractive alternative modality to visible-light cameras for visual-inertial odometry applications. However, state-of-the-art VIO algorithms are designed to work with 8-bit images and cannot directly take advantage of the higher bit-depths that radiometric thermal images provide. Therefore, SVO was modified to support 16-bit matching, feature detection on histogram equalized thermal images, and 16-bit feature detection. The method is dubbed Semi-direct Thermal Inertial Odometry (STIO) and was evaluated on three datasets recorded in different environments. The factor graph-based back-end was implemented using the GTSAM library. STIO was found to be heavily dependent on the parameters, especially on the grid cell size and the FAST threshold, because these directly affected the number of tracked features. Feature detection on histogram equalized images gave features that were more locally distinct and could therefore be tracked with greater reliability. Interestingly, the VIO-extension of SVO run solely on 8-bit histogram equalized images performed surprisingly well, and often outperformed the 16-bit STIO. This was because the 8-bit version detected more edgelet features in addition to all the same corner features. The GTSAM back-end quickly diverged and did not end up in working conditions because of missing information due to programming errors. The author was unable to identify the specific bugs. A simplified initialization system for finding the IMU biases was successfully implemented, although the acceleration biases were found to not converge to the ground truth. Future work includes implementing and evaluating adaptive histogram equalization, comparison against semi-dense methods, and fixing the GTSAM back-end.

Sammendrag

Denne oppgaven presenterer adapsjonen og evalueringen av algoritmen Semi-direct Visual Odometry (SVO) til å bruke 16-biters termiske bilder, samt implementasjonen av en faktorgraf-basert backend. Varmekamera kan passivt observere omgivelsene, selv i helt mørke forhold og i tilstedeværelsen av støv og røyk. Dette gjør dem til attraktive alternativer til vanlige kamera til bruk i visuell-treghets-odometri (VIO). De fremste VIO-algoritmene er derimot designet for bruk med 8-bitersbilder og kan ikke utnytte radiometriske termiske bilders større datadybde. Av den grunn var SVO endret til å støtte 16-bitersbilder til bruk i matching, deteksjon av kjennemerker på histogram-normaliserte bilder og deteksjon direkte på 16-bitersbilder. Det modifiserte metoden er kalt Semi-direct Thermal Inertial Odometry (STIO), og ble evaluert på tre datasett fra ulike omgivelser. Den faktorgraf-baserte backenden er basert på GTSAM-biblioteket. Det ble funnet at STIO var særdeles avhengig av hvilke parametere som ble brukt. Spesielt hadde størrelsen på rutene i deteksjonsrutenettet og FAST-grensen stor betydning fordi disse var tett knyttet til antallet kjennemerker som ble brukt. Deteksjon på histogram-normaliserte bilder ga kjennemerker som lokalt sto mer ut og kunne derfor bli matchet med større treffsikkerhet. Overraskende nok gjorde VIO-utvidelsen av SVO som kun brukte 8-biters histogram-normaliserte bilder det vel så godt, om ikke bedre, enn 16-bitersversjonene. Dette antas å skylles at 8-bitersversjonen fant flere kant-kjennemerker i tillegg til de samme hjørnene. GTSAM-backenden divergerte fort og kom aldri i en fungerende tilstand på grunn av manglende informasjon som resultat av programmeringsfeil. Forfatteren fant derimot ikke årsaken til feilen til tross for stor feilsøkingsinnsats. Et forenklet oppstartssystem for å finne biasene i treghetssensoren var vellykket gjennomført, selv om akselerometerbiasene ikke konvergerte til den sanne verdien. Fremtidig arbeid inkluderer å teste ut adaptiv histogram-normalisering, sammenlikning med semi-dense algoritmer som DSO, og å fikse GTSAM-backenden.

Table of Contents

List of Tables

List of Figures

Abbreviations

APE Absolute Pose Error. 37, 38, 68, 74, 94

CLAHE Contrast Limited Adaptive Histogram Equalization. 93

COLAMD Column Approximate Minimum Degree. 33

DAG directed acyclic graph. 26

DSO Direct Sparse Odometry. 90, 93

EuRoC European Robotics Challenge. 52, 55, 57, 86

FAST Features from Accelerated Segment Test. 3, 11, 45, 48, 56, 57, 59, 60, 63, 65, 68, 70, 77, 81, 90, 91, 93, 95

FFC Flat Field Correction. 5, 47, 52, 70, 89, 91, 95

GPS Global Positioning System. 1, 91

GTSAM Georgia Tech Smoothing and Mapping. 2, 30, 39, 49–52, 57, 59, 86, 87, 89, 92–96

IFL Incremental Fixed Lag. 94

IMU Inertial Measurement Unit. 1, 2, 33–35, 37, 42, 49–52, 55–57, 59, 66, 68, 70, 74, 77, 81, 86, 87, 90–93, 95, 96

iSAM Incremental Smoothing and Mapping. 31, 49, 86, 92, 94

KTIO Keyframe-based Direct Thermal-Inertial Odometry. 1, 2, 52

LiDAR Light Detection and Ranging. 52, 55, 92, 93

LM Levenberg-Marquardt. 92

LOAM LiDAR Odometry and Mapping. 52, 55, 57

LWIR Long-wave Infrared. 5, 52

MAP Maximum A-Posteriori. 23–25, 27, 29, 30, 33, 44

MAV Micro-Areal Vehicle. 1, 6, 52, 55, 56, 65, 68, 81, 86, 92

NLS Non-linear Least Squares. 25, 29, 30, 48, 49

OKVIS Open Keyframe-based Visual-Inertial SLAM. 48, 94

RMSE Root Mean Square Error. 37, 38, 84, 86

ROTIO Robust Thermal-Inertial Odometry. 2, 48, 57, 68, 74, 77, 81, 90

ROVIO Robust Visual-Inertial Odometry. 57, 90

ROVTIO Robust Visual-Thermal-Inertial Odometry. 2

SAM Smoothing and Mapping. 30, 31

SLAM Simultaneous Localization and Mapping. 1, 10, 23, 26–28, 30, 49, 92, 93

STIO Semi-direct Thermal-Inertial Odometry. 39, 47, 57, 59, 68, 70, 74, 77, 81, 89–93, 95

SVO Semi-direct Visual Odometry. 2, 39, 41–44, 46–49, 57, 59, 65, 89, 91, 92, 95

TIO Thermal-Inertial Odometry. 2, 57

VIO Visual-Inertial Odometry. 1, 2, 33, 44, 48, 51, 57, 90–93, 95, 96

VO Visual Odometry. 2, 23, 91, 95

Chapter 1

Introduction

1.1 Motivation

State estimation is an essential component of any autonomous robotic system. For the robot to make an informed decision, it needs to know how it is positioned and oriented relative to its surroundings. This is especially challenging in Global Positioning System (GPS) denied environments such as in indoor or underground settings. Therefore, there is much research into fusing multiple sensor modalities to accurately and robustly estimate the ego-motion of a robot moving in such environments.

Visual-Inertial Odometry (VIO) is one class of problems trying to fuse visual information for cameras and inertial information from an Inertial Measurement Unit (IMU). VIO is a subset of visual Simultaneous Localization and Mapping (SLAM). Several state-of-the-art algorithms exist (e.g. SVO [23], DSO [15], LSD-SLAM [16], OKVIS [35], ORB-SLAM [9], VINS-Mono [42]) that work well for visual-light images. However, the illumination conditions of the environment can greatly affect the effectiveness of the algorithms as darkness eliminates much of the information available from camera images. This is particularly challenging for Micro-Areal Vehicles (MAVs) that have limited power and cannot externally illuminate the scene using flashlights. Obscurants like dust and smoke also reduce visual-light cameras' visibility and degrade the data.

Therefore, research is being done into the use of sensors that can passively observe a scene in adverse conditions. One such sensor is a thermal camera that can ”see” thermal radiation. Therefore, a thermal camera can observe the scene, even in complete darkness, since everything with a temperature above 0 Kelvin emits thermal radiation. Thermal radiation also has a longer wavelength and can better penetrate obscurants like dust and smoke. This makes it a good candidate as an alternative to or complementary sensor to visual-light cameras.

However, thermal images can typically provide a larger 14- or 16-bit bit depth than the usual 8-bits of visual-light cameras and can not be used directly with the methods mentioned above. This motivates the development of algorithms that can take advantage of the full data. Some examples include Keyframe-based Direct Thermal-Inertial Odometry (KTIO) [33], Robust Thermal-Inertial Odometry (ROTIO) [32], and Robust Visual-Thermal-Inertial Odometry (ROVTIO) [18]. These methods leverage direct matching, meaning they operate directly on the pixels' intensity values. This is opposed to indirect methods that employ other means to find correspondences between images. The authors of [33, 32, 18] all concluded that direct methods are the better choice when using the full 16-bit thermal data.

This motivates the exploration into using other state-of-the-art direct VIO methods for Thermal-Inertial Odometry (TIO). For this reason, this thesis will convert the Semi-direct Visual Odometry (SVO) front-end algorithm to work with thermal images. Visual methods for state estimation distinguish between the front-end and back-end. ”The front-end abstracts sensor data into models that are amenable for estimation, while the back-end performs inference on the abstracted data produced by the front-end.” [8] SVO is originally a Visual Odometry (VO) method only, but extensions exists that include VIO capabilities [35] using the Ceres optimizer library [1]. However, it is of interest to develop a VIO back-end using a custom-built factor graph library like Georgia Tech Smoothing and Mapping (GTSAM) as well. This can allow for simplified development and inclusion of other sensor modalities and loop closures in the future.

1.2 Objectives

The main objectives for this master's thesis are, therefore, to convert the state-of-the-art VIO algorithm SVO to work on 16-bit radiometric thermal images and implement a factor graph-based optimization back-end for VIO using the GTSAM library. These will then be evaluated using real-life datasets. The converted SVO algorithm's accuracy and reliability will be evaluated by testing it on several thermal datasets from various environments. This will give insight into the strength and weaknesses of the proposed method. To focus on the core functionality of the GTSAM back-end and ease the development process, the back-end will be assisted with external priors as an initialization routine.

Therefore, the research questions for this thesis can be summarized as follows:

1. What is a good way of extending a state-of-the-art VIO method to work with 16-bit radiometric thermal images?
2. How does the proposed method compare against other TIO methods in constructed indoor and real-life outdoor environments?
3. How can a factor graph-based optimization back-end for VIO be implemented in GTSAM?
4. Can IMU preintegration and external priors be used to ease development by helping initialize the system?

1.3 Contributions

This thesis contributes with an extensive evaluation of how the modified SVO front-end algorithm performs on multiple datasets that cover scenarios where thermal cameras have an advantage over visible-light cameras. It also contributes with a thorough comparison of how the FAST feature detector performs when it is used on histogram equalized images versus directly on the 16-bit image.

1.4 Outline

This thesis is structured as follows: Chapter 2 will introduce relevant background theory necessary for the development and understanding of the proposed method; Chapter 3 will go into details surrounding the implementation of the proposed method, and also outline the experiments that will be used to evaluate the method; Chapter 4 will present the results from the experiments; Chapter 5 will discuss the results and propose future work; and lastly, Chapter 6 will conclude the thesis.

Chapter 2

Theory

2.1 Thermal imaging

Thermal imaging is the process of creating images based on heat. Regular cameras work by focusing visible light onto an image sensor which registers the intensity of different wavelengths, thus creating an image. The process is the same for thermal cameras, except it uses infrared radiation instead of visible light.

All objects with an absolute temperature above 0 Kelvin emits thermal radiation dictated by Stefan-Boltzmann law. [56, ch. 17, p. 569] Using this law, the subdivision of the infrared spectrum called Long-wave Infrared (LWIR) is defined to be the wavelengths between 8 and 15 μm and corresponds to temperatures between -80 and 89 °C. [7, p. 22]

LWIR is, therefore, the called the "thermal imaging" range as many objects typically found in our environments are in this temperature range, enabling the creation of a picture of the world without external light or thermal source. [7] This enables thermal cameras to "see" in the dark. The emitted thermal radiation is also a longer wavelength than visible light, making it possible to penetrate obscurants such as smoke and dust. These facts make thermal cameras a viable substitute or complementary sensor in visual state estimation where visible light cameras struggle. [33]

Thermal cameras accumulate sensor noise during their operation and need to reduce this noise periodically. This is done through Flat Field Correction (FFC). During FFC, the camera pauses for up to 500 milliseconds, and a uniform temperature (i.e., a flat field) is presented to the image sensor. This allows for the estimation of noise correction parameters. [19] However, this process poses two challenges. Firstly, the FFC process can lead to a significant change in illumination. This can make both direct and indirect methods of matching fail due to too large intensity differences. Secondly, the image data stream gap can cause drift in the odometry estimate. This can be especially pronounced in filter-based algorithms where the estimate is iteratively propagated but is less pronounced in optimization-based algorithms that optimize over a window of keyframes. [33]

Other challenges with thermal imaging include absorption in the environment and the lens, thermal reflection in the scene, and vignetting from non-uniform camera temperatures. In the same way that glass is transparent for visible light, germanium is transparent for thermal radiation. For this reason, lenses in thermal cameras are typically made of germanium. [50] Also, similarly to visible light, thermal radiation will reflect off objects in the scene and cause objects to look "washed out." [11] Lastly, thermal cameras work best when all components are at the same temperature. However, this can be hard to achieve, especially on an autonomous MAV. Therefore, the non-uniform temperature will cause some vignetting because the outer parts of the camera are cooler than the inner parts. [49] Vignetting can be challenging for computer vision tasks.

Thermal cameras come in radiometric and non-radiometric variants. Thermal cameras can be used to measure the temperature of each pixel. However, radiometric measurements can be corrected to account for effects due to emissivity and distance to the object, thus finding a more accurate temperature measurement. This is not true for non-radiometric measurements. [55]

Radiometric measurements pose a challenge in visual state estimation, though. Radiometric measurements are typically more than 8-bit (e.g., 14-or 16-bit) and need to be represented using 16-bit integers. However, virtually all state-of-the-art visual state estimation algorithms are based on 8-bit, gray-scale visible light images. [23, 3, 9, 15] To use thermal images directly, the algorithms must be changed to support 16-bit images natively, or the thermal images must be rescaled to 8-bit.

2.1.1 Re-scaling techniques

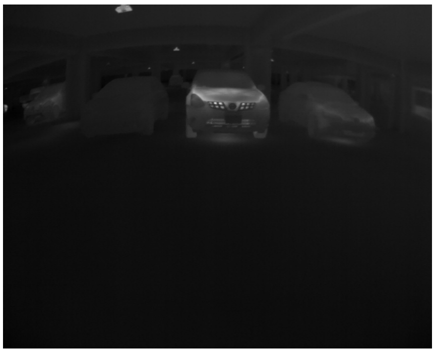
There are several ways to rescale a 16-bit image to 8-bit. One way is to divide all pixel values by the same factor. Alternatively, a predefined interval can be selected, or lastly, a histogram equalization can be applied.

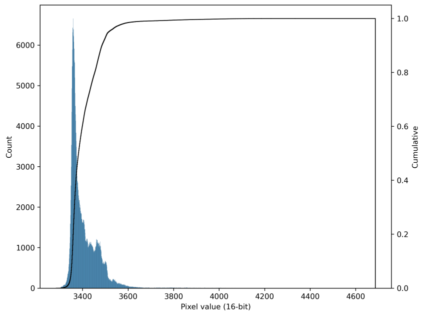
Dividing all pixel values by the same factor is a simple but naive method. For example, a 16-bit image has values in the range 0 to 65 535, so integer division by a factor of 256 will rescale all pixel values to the 8-bit range 0 to 255. However, as Fig. 2.1a shows, this results in loss of information in terms of low contrast, which is poorly suited for feature detection. The images get a low contrast because the variance in temperature in a scene is typically low. This cause many values to be truncated to the same value.

Instead, the fact that there is low variance in temperatures can be exploited, and the 16-bit values can be truncated by selecting an acceptable range of values to be rescaled to 8-bit. This will increase the contrast in the rescaled image. However, this requires manually selecting the range for each environment. It is also unsuitable for long-term usage because thermal cameras accumulate sensor noise over time, shifting the acceptable range of values. [33]

A better way to increase global contrast is to use histogram equalization. Figures 2.1b and 2.1d shows the histogram before and after equalization and the resulting equalized image. The histogram equalization is done by making the cumulative distribution of the image linear. [25, sec. 3.3.1, p. 144] This spreads out the values that are close together. As Fig. 2.1c shows, the contrast is significantly better than Fig. 2.1a. However, histogram equalization by itself does not convert the image to 8-bit. This is instead done afterward by rescaling the image.

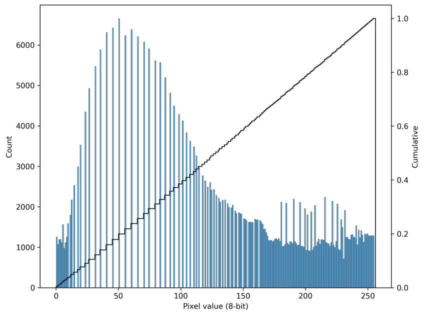
However, the histogram equalization is dependent on the temperature of the objects currently in the camera's field of view. The image contrast will change significantly if a


(a) 16-bit thermal image scaled by a constant factor.



(b) Histogram and cumulative distribution of the full 16-bit image in Fig. 2.1a.

(c) Histogram equalized 16-bit thermal image.



(d) Histogram and cumulative distribution of the 8-bit histogram equalized image in Fig. 2.1c.

Figure 2.1: Applying histogram equalization on 16-bit images give better contrast in the 8-bit rescaled image by making the cumulative distribution have a linear trend. Made using a thermal image from the Urban Parking Lot dataset presented in Section 3.4.2.

cold or hot object enters the view. [33] This can be problematic when the thermal camera is moving in robotic state estimation.

Rescaling is also important for visualization. High contrast is desirable to see the objects in the scene easily. For this reason, in the remainder of this thesis, the histogram equalization technique will be used to illustrate thermal images unless otherwise noted.

2.2 Camera geometry

Section 2.2.1 is copied more or less verbatim from the author's specialization project [17], with only minor modifications. Section 2.2.2 is simplified from [17] as the specific details surrounding distortion are less important in this thesis.

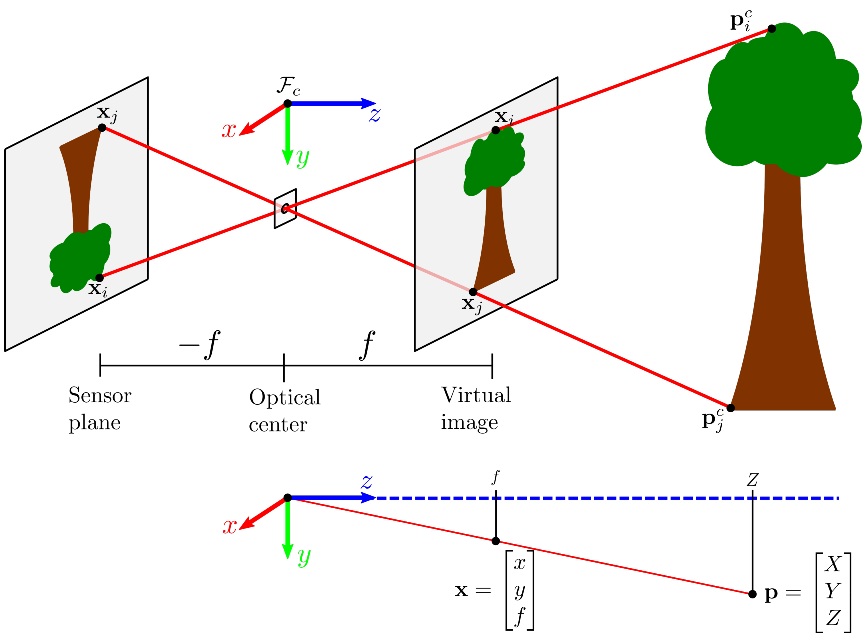


Figure 2.2: An illustration showing the pinhole camera model. The camera frame is fixed in the optical center, i.e. where the pinhole is. The blue z-axis is called the optical axis.

2.2.1 Pinhole camera

This section will briefly introduce the pinhole camera and equidistant distortion models. The theory behind the pinhole model is compiled from Hartley and Zisserman [27] and Ma et al. [38].

The pinhole camera model is a geometric camera model describing the relationship between 3D world points in the camera frame $\mathbf{p}^c$ with the 2D pixel coordinates in the image $\mathbf{u}$. Mathematically, it is written as $\mathbf{u} = \pi(\mathbf{p}^c)$ where $\pi\colon \mathbb{R}^3 \to \Omega$ is the projection function mapping 3D points to the image plane $\Omega \subset \mathbb{R}^2$. To express the projective geometry as matrix multiplications, homogeneous coordinates are used. Thus, the world point and pixel coordinates are expanded as

$$\mathbf{p}^c = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^\top \tag{2.1}$$

$$\mathbf{u} = \begin{bmatrix} u & v & 1 \end{bmatrix}^\top \tag{2.2}$$

Figure 2.2 shows the setup for the camera model. Light rays from an object in the world go through the pinhole and onto the image sensor. The origin of the camera frame, also known as the optical center, is fixed to the pinhole. The z-axis is called the optical axis and is perpendicular to the sensor plane. The focal length f specifies the distance from the pinhole to the sensor. The image on the sensor will make objects appear upside down. Therefore, a virtual image is placed in front of the pinhole when modeling to

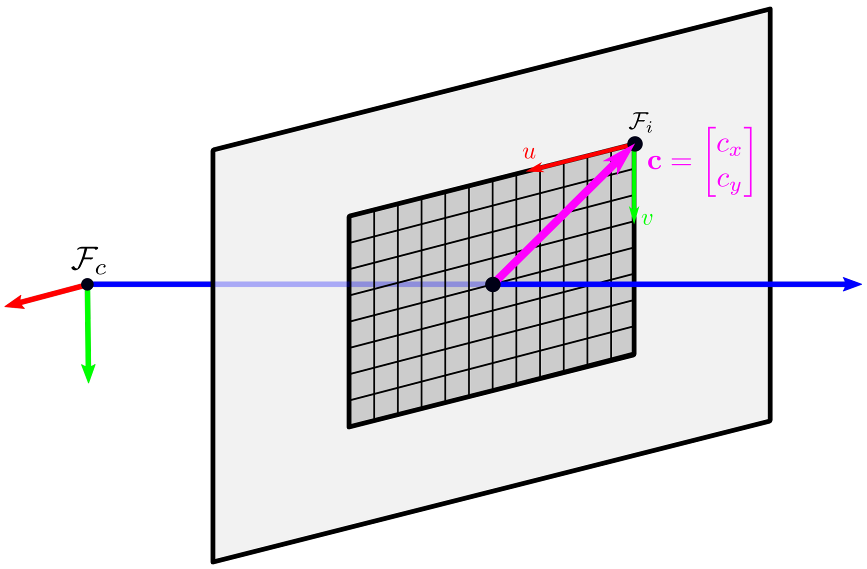


Figure 2.3: Affine transformation from camera frame $\mathcal{F}_c$ to image frame $\mathcal{F}_i$.

make the derivations easier. The coordinates in the virtual image are f units along the optical axis in front of the optical center and are often expressed in the the camera frame using $\mathbf{x} = \begin{bmatrix} x_f & y_f & f \end{bmatrix}^\top = \begin{bmatrix} x & y & 1 \end{bmatrix}^\top$. The last equality is valid as homogeneous coordinates are unique up to a scale factor. When the last component of $\mathbf{x}$ is 1, the point is said to be in the *normalized image plane*. It is preferred to work with points in the normalized image plane to make the calculations easier. The coordinates in $\mathbf{x}$ are different from the pixel coordinates $\mathbf{u}$ by an affine transformation, shown later.

Similar triangles are used to find the pixel coordinate of a point from the scene. First, the coordinates in the normalized image plane can be expressed as

$$\frac{x}{f} = \frac{X}{Z} \qquad \frac{y}{f} = \frac{Y}{Z} \tag{2.3}$$

$$x = \frac{f}{Z}X \qquad y = \frac{f}{Z}Y \tag{2.4}$$

which can be expressed as the matrix equation

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \frac{1}{Z} \underbrace{\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}_f} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{\Pi}_0} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \frac{1}{Z}\mathbf{K}_f \mathbf{\Pi}_0 \mathbf{p}^c \tag{2.5}$$

An affine transformation is used to find the pixel coordinates $\mathbf{u}$ of $\mathbf{x}$. This transformation is given by

$$\mathbf{u} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} s_x & 0 & c_x \\ 0 & s_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}_i} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{K}_i \mathbf{x} \tag{2.6}$$

where s_x and s_y are the size of the sensor pixels in metric units in the horizontal and vertical direction, and c_x and c_y are the coordinates of the principal point. The principle point is where the optical axis intersects the sensor plane. An additional skew parameter s_θ can be included in $\mathbf{K}_i$, but this is often negligible small in modern cameras and is therefore ignored. Putting Eq. (2.5) and Eq. (2.6) together, the resulting model becomes

$$\mathbf{u} = \frac{1}{Z}\mathbf{K}_i \mathbf{K}_f \mathbf{\Pi}_0 \mathbf{p}^c = \frac{1}{Z}\mathbf{K}\mathbf{\Pi}_0 \mathbf{p}^c = \pi(\mathbf{p}^c) \tag{2.7}$$

The matrix $\mathbf{K} = \mathbf{K}_i \mathbf{K}_f$ is called the intrinsic matrix and is often expressed as

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

where $f_* = f s_*$. The parameters of the pinhole model can therefore be summarizes as the vector $\mathbf{k} = \begin{bmatrix} f_x & f_y & c_x & c_y \end{bmatrix}$.

2.2.2 Distortion

Equation (2.7) is valid in the ideal case, but in reality, the pixel location is distorted when the light rays pass through the lens. This is modeled with a distortion model. Figure 2.4 shows how a light ray becomes distorted and ends up in a different location $\mathbf{x}_d$ than the projective model would suggest. This shift in location is modeled with a distortion model. Many models exist, and which model works best for a particular camera will depend on the camera's lens geometry and other factors. To illustrate how distortion work, the simple radial distortion model will be presented.

Radial distortion is built on the idea that points further away from the projective center will be more affected. The distance of a point $\mathbf{x}$ from the projective center $r = \sqrt{x^2 + y^2}$ is calculated in the normalized image plane. The relationship between the distorted point location $\mathbf{x}_d$ and the undistorted location $\mathbf{x}$ use a low-order polynomial in r, e.g.

$$\mathbf{x} = \mathbf{x}_d(1 + k_1 r^2 + k_2 r^4) \tag{2.9}$$

where $\mathbf{d} = \begin{bmatrix} k_1 & k_2 \end{bmatrix}$ are the parameters of the distortion model. [48] This polynomial is used for the simple radial distortion, but other polynomials or functions can also be used. The distortion parameters $\mathbf{d}$ are found together with the intrinsic parameters $\mathbf{k}$ when doing intrinsic camera calibration.

In conclusion, the pinhole camera model $\mathbf{u} = \pi(\mathbf{p}^c)$ includes both the pinhole projection Eq. (2.7) and the distortion Eq. (2.9). A common scenario in SLAM is that the landmark is given in the world frame, $\mathbf{p}^w$, and the camera's pose is also given in the world frame, $\mathbf{T}_{wc}$. It is therefore also common to see π take the pose of the camera as a parameter, i.e.

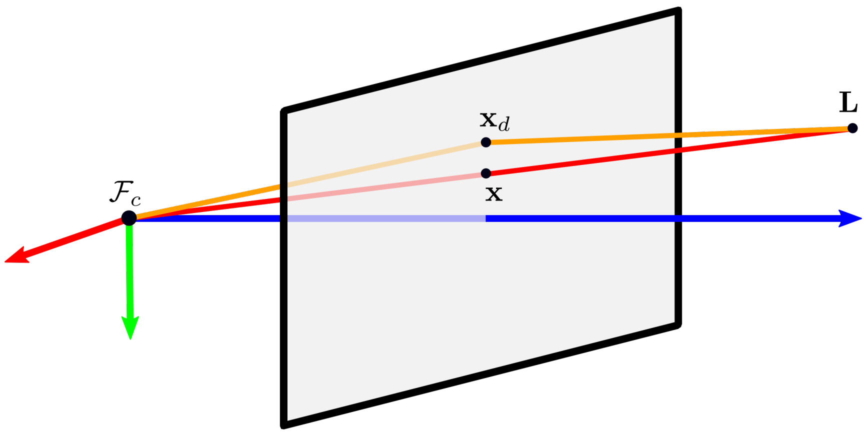


Figure 2.4: Illustration of how distortion affects the projected point.

$$\mathbf{u} = \pi(\mathbf{p}^w;\ \mathbf{T}_{wc}) = \pi(\mathbf{T}_{wc}^{-1}\mathbf{p}^w) \tag{2.10}$$

2.3 FAST corner detector

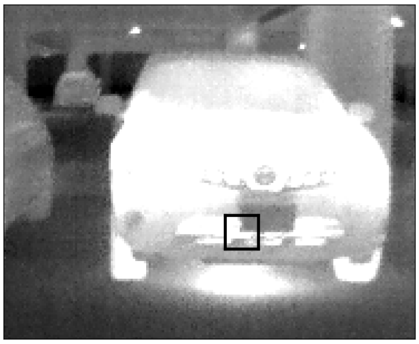
FAST is a corner detector proposed in [44]. It is a feature detector that gives repeatable corner features at a low computation cost. FAST operates directly on pixel values. This section will briefly summarize each part of the algorithm. For more details, please refer to the original paper [44].

Corners are found by comparing the 16 pixels in a circle around a point p, see Fig. 2.5b, with intensity I_p. An appropriate threshold t is also chosen. An initial high-speed test is performed by examining the pixels at 1, 9, 5, and 13 (i.e. the cardinal directions, see Fig. 2.5b). If less than three pixels are brighter than $I_p + t$ or darker than $I_p - t$, p cannot be a corner. However, if at least three out of the four pixels pass the threshold test, a full segment test checks all the pixels in the circle. If at least n contiguous pixels are brighter than $I_p + t$ or darker than $I_p - t$, the pixel p is selected as a candidate. In the original paper, n was chosen to be 12.

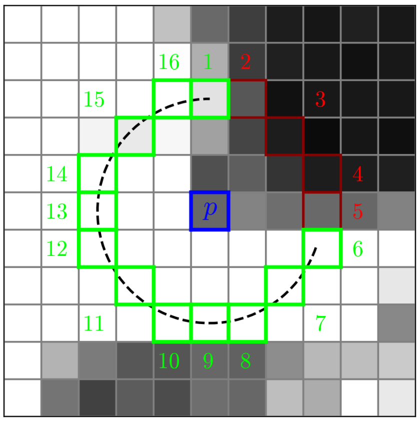
This segment test can give many points close together. To remove adjacent corners, non-maximal suppression is used. A score V is calculated for each point p as the sum of the absolute difference between I_p and the intensities of the pixels in the contiguous arc. The points with higher values V are kept, and the others are discarded.

In the paper by Rosten and Drummond [44], a machine learning approach is proposed as an alternative to the segment test. However, both OpenCV [4] and the library that will be used in this thesis[1] does not incorporate this machine learning approach. Instead, they use the segment test and non-maximal suppression. Hence, the machine learning approach is not described here, and its details are left to the original paper [44].

[1]Modified version of `fast_neon`: `https://github.com/uzh-rpg/fast_neon`



(a) The image to detect on.



(b) Zoomed in view of the black square in Fig. 2.5a.

Figure 2.5: Illustration of how the pixels around a pixel of interest are sampled. The green squares are all pixels brighter than $I_p + t$, while the red are darker. The dashed arc indicate that there are $n = 12$ contiguous pixels brighter than the reference, thus passing the segment test. This figure is inspired by Fig. 1 from the original paper [44], but uses a histogram equalized thermal image from the Urban Parking Lot dataset presented in Section 3.4.2.

2.4 Lie theory for state estimation

Lie theory gives a convenient and robust representation of poses lying on a non-Euclidean manifold. It provides a way to represent probability densities and do optimization with non-Euclidean elements. Lie theory is a comprehensive branch of mathematics, so this section will only present the most basic and relevant parts for state estimation. This section is based on the paper by Solà et al. [46]. Concrete examples will be given where relevant when presenting the theory, and the examples will only use manifolds relevant to state estimation. This section will not serve as a rigorous study of Lie theory but rather a pragmatic introduction to how to apply the theory to one specific case.

2.4.1 Lie groups

A Lie group is a smooth manifold whose elements satisfy the group axioms. A smooth, or differentiable, manifold is a topological space that resembles linear space locally. In other words, a smooth manifold is a curved, smooth surface with no edges or spikes embedded in a higher-dimensional space. For example, the one-dimensional perimeter of the unit circle is a smooth manifold embedded in two-dimensional space.

A group $(\mathcal{G}, \circ)$ is a set $\mathcal{G}$ with a compositional operator $\circ$ for which the elements $\mathcal{X}, \mathcal{Y}, \mathcal{Z} \in \mathcal{G}$ satisfy the group axioms

$$\text{Closure under } \circ \;:\; (\mathcal{X} \circ \mathcal{Y}) \in \mathcal{G} \tag{2.11a}$$
$$\text{Identity } \mathcal{E} \in \mathcal{G} \;:\; \mathcal{E} \circ \mathcal{X} = \mathcal{X} \circ \mathcal{E} = \mathcal{X} \tag{2.11b}$$
$$\text{Inverse } \mathcal{X}^{-1} \;:\; \mathcal{X}^{-1} \circ \mathcal{X} = \mathcal{X} \circ \mathcal{X}^{-1} = \mathcal{E} \tag{2.11c}$$
$$\text{Associativity} \;:\; (\mathcal{X} \circ \mathcal{Y}) \circ \mathcal{Z} = \mathcal{X} \circ (\mathcal{Y} \circ \mathcal{Z}) \tag{2.11d}$$

Note that the compositional operator is not commutative in general.

The group structure of a Lie group ensures that the composition of elements on the manifold stays on the manifold due to the closure property Eq. (2.11a), and that each element on the manifold has an inverse, Eq. (2.11c). These two properties are vital for representing a robot's state. Another property of the Lie group is that the manifold looks the same at every point. For example, every point on the unit circle will locally look the same.

In summary, Lie groups combines the local properties of smooth manifolds with the global properties of groups, enabling the use of calculus and non-linear composition of distant objects.

Lie groups also have a group action operator "$\cdot$" that can transform elements of other sets. Given a Lie group $\mathcal{M}$ and a set $\mathcal{V}$, the Lie element $\mathcal{X} \in \mathcal{M}$ *acts* on the element $v \in \mathcal{V}$ through $\cdot$, i.e. $\mathcal{X} \cdot v \in \mathcal{V}$. This is used extensively in robotics to represent rotation, transformation, and scaling. For example, a rotation matrix $\mathcal{X} = \mathbf{R}_{ab} \in \mathcal{M} = SO(3)$ can transform a vector $v = \mathbf{v}^b \in \mathcal{V} = \mathbb{R}^3$ from frame $\mathcal{F}_b$ to frame $\mathcal{F}_a$ through the group action $\mathbf{v}^a = \mathbf{R}_{ab} \cdot \mathbf{v}^b \triangleq \mathbf{R}_{ab}\mathbf{v}^b$. For $SO(3)$ and $\mathbb{R}^3$, the group action reduces to matrix multiplication. On the other hand, a transformation $\mathbf{T} \triangleq [\mathbf{R} \mid \mathbf{t}] \in SE(3)$ acts on a homogeneous vector $\tilde{\mathbf{x}} = \begin{bmatrix}\mathbf{x} & 1\end{bmatrix}^\top \in \mathbb{R}^4$ as $\mathbf{T} \cdot \tilde{\mathbf{x}} \triangleq \mathbf{R}\mathbf{x} + \mathbf{t}$. In later sections and other works, the operator "$\cdot$" is typically dropped for brevity.

2.4.2 Tangent spaces

The velocity $\dot{\mathcal{X}}(t) = \partial\mathcal{X}/\partial t$ of a point $\mathcal{X}(t)$ moving on a manifold $\mathcal{M}$ exist in the tangent space at that point, denoted $T_\mathcal{X}\mathcal{M}$. Due to the smoothness of the manifolds, there exists a unique tangent space at every point, and the structure of the tangent space is the same everywhere. Tangent spaces are essential as velocities and perturbations are defined in the tangent space. Through a special operation called the exponential map, the elements of the tangent space can transform the elements of the manifold.

The tangent space at the identity is called the *Lie algebra* of $\mathcal{M}$ and is denoted $\mathfrak{m} \triangleq T_\mathcal{E}\mathcal{M}$. Every Lie group has an associated Lie algebra. Three main properties relate them to each other. These properties are

- The Lie algebra $\mathfrak{m}$ is a vector space. This means there exists a linear transformation between elements of $\mathfrak{m}$ and the vectors of $\mathbb{R}^m$. The dimension m is the number of degrees of freedom of the manifold $\mathcal{M}$. For $SO(3)$, $m = 3$ as there can be rotated around each of the three axes.
- The *exponential map* $\exp : \mathfrak{m} \to \mathcal{M}$ exactly transforms an element in the Lie algebra to an element on the manifold. The inverse operation is the *logarithmic map* $\log : \mathcal{M} \to \mathfrak{m}$. These operations will be introduces in Section 2.4.3.

- Elements of the tangent space at $\mathcal{X}$ can be transformed into elements of the Lie algebra $\mathfrak{m}$, and vice versa, through a linear transform called the *adjoint*. This will be introduced in Section 2.4.5. Together with the fact that all tangent spaces of a manifold have the same structure, the properties of the Lie algebra are the same as any other tangent space.

Elements of a tangent space are denoted with a "hat" symbol, for example $\boldsymbol{\tau}^\wedge$. An optional left superscript can be added to indicate the exact tangent space the element is in, for example ${}^{\mathcal{X}}\boldsymbol{\tau}^\wedge \in T_{\mathcal{X}}\mathcal{M}$ and ${}^{\mathcal{E}}\boldsymbol{\tau}^\wedge \in T_{\mathcal{E}}\mathcal{M} = \mathfrak{m}$.

The structure of the Lie algebra can be found by time differentiating the group constraint Eq. (2.11c). Multiplicative groups are groups where the composition $\circ$ can be thought of as multiplication, e.g., rotation matrices in $SO(3)$ where the composition becomes matrix multiplication. For multiplicative groups, the constraint for the tangent space becomes

$$\begin{aligned} \frac{\partial}{\partial t}(\mathcal{X}^{-1} \circ \mathcal{X}) &= \frac{\partial \mathcal{E}}{\partial t} \\ \dot{\mathcal{X}^{-1}} \circ \mathcal{X} + \mathcal{X}^{-1} \circ \dot{\mathcal{X}} &= 0 \end{aligned} \tag{2.12}$$

where the product rule is used on the left side, the derivative of the constant identity $\mathcal{E}$ is zero, and $\dot{\mathcal{X}^{-1}}$ denotes the time derivative of the inverse. Elements of the tangent space are, therefore, on the form

$$\mathbf{v}^\wedge = \mathcal{X}^{-1} \circ \dot{\mathcal{X}} = -\dot{\mathcal{X}^{-1}} \circ \mathcal{X} \tag{2.13}$$

where $\mathbf{v}^\wedge$ can be thought of as the velocity at $\mathcal{X}$. For $SO(3)$, the group constraint is $\mathbf{R}^\top\mathbf{R} = \mathbf{I}$, so the tangent space constraint becomes $\dot{\mathbf{R}}^\top\mathbf{R} = -\mathbf{R}^\top\dot{\mathbf{R}}$. This means that elements of the tangent space of $SO(3)$ are skew-symmetric 3×3 matrices. A derivation of this is given in Example 3 of [46].

Additive groups are groups where the composition $\circ$ can be thought of as addition, e.g., vectors in $\mathbb{R}^3$. For additive groups, the group constraint becomes $\mathcal{X} - \mathcal{X} = 0$ and the tangent space constraint becomes $\dot{\mathcal{X}} = \dot{\mathcal{X}}$. This means there is no constraint on the tangent space, which means that the tangent space is coincident with the manifold. In other words, there is no difference between the manifold and the tangent space.

Although the tangent space elements have non-trivial structures, one of the main properties is that there exists a linear transform between the Lie algebra and the vectors of $\mathbb{R}^m$. Two operations, called *hat* and *vee*, are used to transform elements from $\mathfrak{m}$ to $\mathbb{R}^m$, and vice versa. These are isomorphisms, i.e., two inverse linear maps between two similar spaces, and are defined as

$$\text{Hat}: \quad \mathbb{R}^m \to \mathfrak{m}; \quad \boldsymbol{\tau} \mapsto \boldsymbol{\tau}^\wedge = \sum_{i=1}^{m} \tau_i E_i \tag{2.14a}$$

$$\text{Vee}: \quad \mathfrak{m} \to \mathbb{R}^m; \quad \boldsymbol{\tau}^\wedge = (\boldsymbol{\tau}^\wedge)^\vee = \boldsymbol{\tau} = \sum_{i=1}^{m} \tau_i \mathbf{e}_i \tag{2.14b}$$

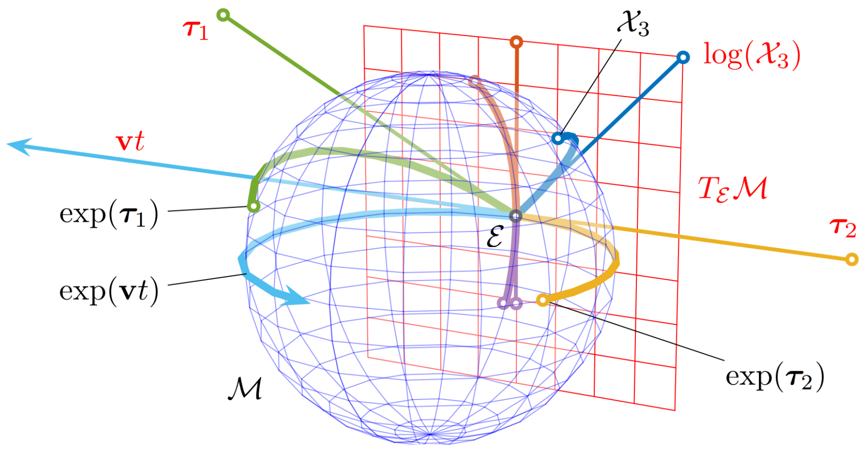


Figure 2.6: Overview illustration showing the relationship between the Lie algebra (tangent space at identity $\mathcal{E}$), manifold elements $\mathcal{X}$, increments, and the exponential map. Image from [46], licensed under CC BY-NC-SA 4.0.

where E_i are the *generators* of $\mathfrak{m}$ and $\mathbf{e}_i$ are the basis vectors of $\mathbb{R}^m$. They are also related through the hat and vee operators as $\mathbf{e}_i^\wedge = E_i$. The generators can be thought of as basis elements for the Lie algebra.

These isomorphic transformations are of interest because the Euclidean vectors $\boldsymbol{\tau} \in \mathbb{R}^m$ are simpler to work with. They can be stacked in large, composite state vectors (Section 2.4.8 will give more details on this), and most importantly, they can be manipulated using linear algebra and matrix operations.

2.4.3 Exponential and logarithmic maps

Velocities and perturbations are given in the Lie algebra, so to evolve a state, it is useful to be able to map a tangent element to the manifold. The exponential map $\exp()$ can exactly transfer a tangent element to an element in the corresponding Lie group. This operation is generally known as *retraction*. As seen in Fig. 2.6, the exponential map $\exp()$ wraps a vector in the Lie algebra around the *geodesic*[2] of the manifold. The inverse mapping is called the logarithmic map $\log()$. Collectively, these maps are known as *charts*. [12]

The exponential map arises by considering the time derivative of a manifold element $\mathcal{X} \in \mathcal{M}$. Left-multiplying Eq. (2.13) with $\mathcal{X}$ gives the ordinary differential equation (ODE)

$$\dot{\mathcal{X}} = \mathcal{X}\mathbf{v}^\wedge \tag{2.15}$$

where the composition operator $\circ$ has been left out for notational clarity. For constant $\mathbf{v}$, the solution of this ODE is

[2]A geodesic is the shortest path between two points on a curved manifold. Intuitively, for a sphere, this is the resulting path from wrapping a string around the sphere. Incidentally, the curved lines showing airplane flight paths on flat maps are geodesics on the spherical globe but look curved when projected to a flat surface.

$$\mathcal{X}(t) = \mathcal{X}(0)\exp(\mathbf{v}^\wedge t) \tag{2.16}$$

Since $\mathcal{X}(t)$ and $\mathcal{X}(0)$ are elements of the group, so must $\exp(\mathbf{v}^\wedge t) = \mathcal{X}(0)^{-1}\mathcal{X}(t)$ also be. This shows that $\exp(\mathbf{v}^\wedge t)$ maps elements of the Lie algebra to elements of the group.

Defining $\boldsymbol{\tau}^\wedge \triangleq \mathbf{v}^\wedge t$ as a perturbation living in the Lie algebra, the exponential and logarithmic maps are defined as

$$\begin{aligned} \exp: \quad & \mathfrak{m} \to \mathcal{M}; \quad \boldsymbol{\tau}^\wedge \mapsto \mathcal{X} = \exp(\boldsymbol{\tau}^\wedge) && \text{(2.17a)} \\ \log: \quad & \mathcal{M} \to \mathfrak{m}; \quad \mathcal{X} \mapsto \boldsymbol{\tau}^\wedge = \log(\mathcal{X}) && \text{(2.17b)} \end{aligned}$$

Key properties of the exponential map include

$$\begin{aligned} \exp((t+s)\boldsymbol{\tau}^\wedge) &= \exp(t\boldsymbol{\tau}^\wedge)\exp(s\boldsymbol{\tau}^\wedge) && \text{(2.18a)} \\ \exp(t\boldsymbol{\tau}^\wedge) &= \exp(\boldsymbol{\tau}^\wedge)^t && \text{(2.18b)} \\ \exp(-\boldsymbol{\tau}^\wedge) &= \exp(\boldsymbol{\tau}^\wedge)^{-1} && \text{(2.18c)} \\ \exp(\mathcal{X}\boldsymbol{\tau}^\wedge\mathcal{X}^{-1}) &= \mathcal{X}\exp(\boldsymbol{\tau}^\wedge)\mathcal{X}^{-1} && \text{(2.18d)} \end{aligned}$$

The last property Eq. (2.18d) is easily proved by simplifying the terms of the Taylor expansion of $\mathcal{X}^{-1}\mathcal{X}$. This property is key in developing the adjoint operator in Section 2.4.5.

The exponential map can be found in closed form for multiplicative groups by expanding and simplifying the absolute convergent Taylor series of the exponential function

$$\exp(\boldsymbol{\tau}^\wedge) = \mathcal{E} + \boldsymbol{\tau}^\wedge + \frac{1}{2}(\boldsymbol{\tau}^\wedge)^2 + \frac{1}{3!}(\boldsymbol{\tau}^\wedge)^3 + \dots \tag{2.19}$$

by taking advantage of the algebraic properties of powers of Lie algebra elements $\boldsymbol{\tau}^\wedge$. The logarithmic map is found by taking the inverse. Example 4 in [46] shows the exponential map for $SO(3)$. For additive groups, the exponential map is simply the identity function because the manifold is coincident with $\mathbb{R}^m$.

The elements of the Lie algebra $\boldsymbol{\tau}^\wedge$ are rarely of interest in state estimation as the equivalent Euclidean vectors $\boldsymbol{\tau}$ are used instead. Hence, convenient shortcuts are defined to directly map Euclidean vectors $\boldsymbol{\tau} \in \mathbb{R}^m$ to elements of the manifold $\mathcal{X} \in \mathcal{M}$, and vice versa. These are distinguished from the standard exponential and logarithmic maps by capitalizing the function, i.e.

$$\begin{aligned} \mathrm{Exp}: \quad & \mathbb{R}^m \to \mathcal{M}; \quad \boldsymbol{\tau} \mapsto \mathcal{X} = \mathrm{Exp}(\boldsymbol{\tau}) \triangleq \exp(\boldsymbol{\tau}^\wedge) && \text{(2.20a)} \\ \mathrm{Log}: \quad & \mathcal{M} \to \mathbb{R}^m; \quad \mathcal{X} \mapsto \boldsymbol{\tau} = \mathrm{Log}(\mathcal{X}) \triangleq \log(\mathcal{X})^\vee && \text{(2.20b)} \end{aligned}$$

Figure 2.7 gives an overview of the exponential and logarithmic maps, the hat and vee operators, and how they transform elements between the manifold, Lie algebra, and Euclidean vector.

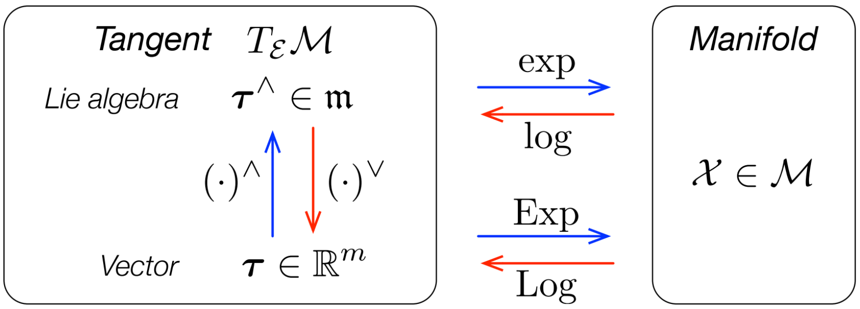

Tangent $T_\mathcal{E}\mathcal{M}$
Lie algebra $\boldsymbol{\tau}^\wedge \in \mathfrak{m}$
$(\cdot)^\wedge$
$(\cdot)^\vee$
Vector $\boldsymbol{\tau} \in \mathbb{R}^m$
exp
log
Exp
Log
Manifold
$\mathcal{X} \in \mathcal{M}$

Figure 2.7: A schematic overview of the relationship between the tangent space at the identity, its isomorphic vector space, and the manifold. Image from [46], licensed under CC BY-NC-SA 4.0.

2.4.4 Plus and minus operators

Plus and minus operators make it possible to introduce increments between elements of a (curved) manifold and express them in its (flat) tangent vector space. They are defined using the Exp/Log operators and a composition $\circ$. Due to the non-commutativity of the composition, a right and a left flavor of the plus and minus operators exist.

The right versions of the operators are defined as

$$\text{right-}\oplus: \quad \mathcal{Y} = \mathcal{X} \oplus {}^{\mathcal{X}}\boldsymbol{\tau} \triangleq \mathcal{X} \circ \operatorname{Exp}\left({}^{\mathcal{X}}\boldsymbol{\tau}\right) \in \mathcal{M} \tag{2.21a}$$

$$\text{right-}\ominus: \quad {}^{\mathcal{X}}\boldsymbol{\tau} = \mathcal{Y} \ominus \mathcal{X} \triangleq \operatorname{Log}\left(\mathcal{X}^{-1} \circ \mathcal{Y}\right) \in T_\mathcal{X}\mathcal{M} \tag{2.21b}$$

because $\operatorname{Exp}\left({}^{\mathcal{X}}\boldsymbol{\tau}\right)$ appears on the right side of the composition in Eq. (2.21a). Hence, ${}^{\mathcal{X}}\boldsymbol{\tau}$ belongs to the tangent space at $\mathcal{X}$. By convention, the increment ${}^{\mathcal{X}}\boldsymbol{\tau} \in T_\mathcal{X}\mathcal{M}$ is said to be in the *local* frame at $\mathcal{X}$.

Likewise, the left versions of the operators are defined as

$$\text{left-}\oplus: \quad \mathcal{Y} = {}^{\mathcal{E}}\boldsymbol{\tau} \oplus \mathcal{X} \triangleq \operatorname{Exp}\left({}^{\mathcal{E}}\boldsymbol{\tau}\right) \circ \mathcal{X} \in \mathcal{M} \tag{2.22a}$$

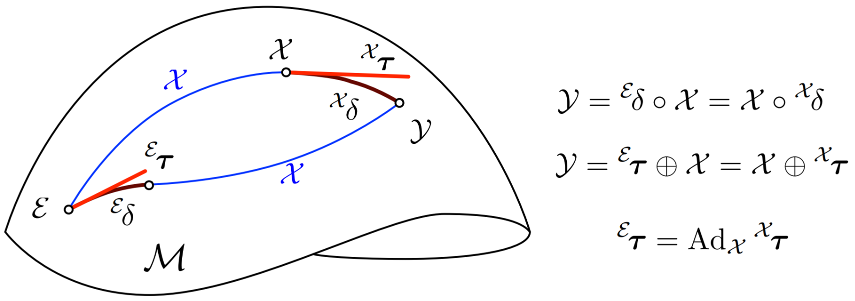
$$\text{left-}\ominus: \quad {}^{\mathcal{E}}\boldsymbol{\tau} = \mathcal{Y} \ominus \mathcal{X} \triangleq \operatorname{Log}\left(\mathcal{Y} \circ \mathcal{X}^{-1}\right) \in T_\mathcal{E}\mathcal{M} \tag{2.22b}$$

These are the left versions because $\operatorname{Exp}\left({}^{\mathcal{E}}\boldsymbol{\tau}\right)$ appear on the left side of the composition in Eq. (2.22a). The increment ${}^{\mathcal{E}}\boldsymbol{\tau} \in T_\mathcal{E}\mathcal{M}$ is now said to be in the *global* frame.

Notice the ambiguity of the minus operator in Eq. (2.21b) and Eq. (2.22b). This thesis will exclusively use the right version of the plus and minus operators to avoid confusion. This implies that all perturbations are expressed locally. This is consistent with the convention chosen by Solà in Solà [45]. Note that other authors use the global convention, notably NASA Jet Propulsion Laboratory (JPL). [45] The choice of which convention is used is not always obvious and can quickly become a source of confusion.

2.4.5 Adjoint

The adjoint operator transform a tangent element between the global and local frame. Setting Eq. (2.21a) equal to Eq. (2.22a) yields the expression $\mathcal{X} \circ \operatorname{Exp}\left({}^{\mathcal{X}}\boldsymbol{\tau}\right) = \operatorname{Exp}\left({}^{\mathcal{E}}\boldsymbol{\tau}\right) \circ$



$\mathcal{X}$
${}^{\mathcal{X}}\boldsymbol{\tau}$
$\mathcal{X}$
${}^{\mathcal{X}}\delta$
$\mathcal{Y}$
${}^{\mathcal{E}}\boldsymbol{\tau}$
$\mathcal{X}$
$\mathcal{E}$
${}^{\mathcal{E}}\delta$
$\mathcal{M}$
$\mathcal{Y} = {}^{\mathcal{E}}\delta \circ \mathcal{X} = \mathcal{X} \circ {}^{\mathcal{X}}\delta$
$\mathcal{Y} = {}^{\mathcal{E}}\boldsymbol{\tau} \oplus \mathcal{X} = \mathcal{X} \oplus {}^{\mathcal{X}}\boldsymbol{\tau}$
${}^{\mathcal{E}}\boldsymbol{\tau} = \mathrm{Ad}_{\mathcal{X}}\, {}^{\mathcal{X}}\boldsymbol{\tau}$

Figure 2.8: Illustration showing the difference between right- and left-$\oplus$, as well as how the adjoint relate the global and local increment element $\boldsymbol{\tau}$. Image from [46], licensed under CC BY-NC-SA 4.0.

$\mathcal{X}$. By manipulating this equation, the relationship between the tangent space elements can be written as ${}^{\mathcal{E}}\boldsymbol{\tau}^{\wedge} = \mathcal{X}{}^{\mathcal{X}}\boldsymbol{\tau}\mathcal{X}^{-1}$. Thus, the adjoint $\mathrm{Ad}_{\mathcal{X}}$ of $\mathcal{M}$ at $\mathcal{X}$ is defined as

$$\mathrm{Ad}_{\mathcal{X}} : \mathfrak{m} \to \mathfrak{m}; \quad \mathrm{Ad}_{\mathcal{X}}(\boldsymbol{\tau}^{\wedge}) \triangleq \mathcal{X}\boldsymbol{\tau}^{\wedge}\mathcal{X}^{-1} \tag{2.23}$$

which means ${}^{\mathcal{E}}\boldsymbol{\tau}^{\wedge} = \mathrm{Ad}_{\mathcal{X}}({}^{\mathcal{X}}\boldsymbol{\tau}^{\wedge})$. The adjoint has two useful properties:

$$\text{Linearity}: \quad \mathrm{Ad}_{\mathcal{X}}(a\boldsymbol{\tau}^{\wedge} + b\boldsymbol{\sigma}^{\wedge}) = a\mathrm{Ad}_{\mathcal{X}}(\boldsymbol{\tau}^{\wedge}) + b\mathrm{Ad}_{\mathcal{X}}(\boldsymbol{\sigma}^{\wedge}) \tag{2.24a}$$

$$\text{Homomorphism}: \quad \mathrm{Ad}_{\mathcal{X}}(\mathrm{Ad}_{\mathcal{Y}}(\boldsymbol{\tau}^{\wedge})) = \mathrm{Ad}_{\mathcal{X}\mathcal{Y}}(\boldsymbol{\tau}^{\wedge}) \tag{2.24b}$$

Since the adjoint is a linear map, an equivalent matrix can be found. [34, p. 94] This is called the adjoint matrix and is defined as

$$\mathbf{Ad}_{\mathcal{X}} : \mathbb{R}^m \to \mathbb{R}^m; \quad {}^{\mathcal{X}}\boldsymbol{\tau} \mapsto {}^{\mathcal{E}}\boldsymbol{\tau} = \mathbf{Ad}_{\mathcal{X}}{}^{\mathcal{X}}\boldsymbol{\tau} \tag{2.25}$$

In the same way the adjoint maps an element in the tangent space at $\mathcal{X}$ to the Lie algebra, the adjoint matrix maps an element in the equivalent Cartesian vector space at $\mathcal{X}$ to the equivalent Cartesian vector space of the Lie algebra.

To find the adjoint matrix, the $^{\vee}$ operator is applied to Eq. (2.23) to give $\mathbf{Ad}_{\mathcal{X}}\boldsymbol{\tau} = (\mathcal{X}\boldsymbol{\tau}^{\wedge}\mathcal{X}^{-1})^{\vee}$. However, the adjoint matrix is dependent on the underlying manifold and needs to be developed for each Lie group. See Example 6 of [46] for a derivation of the adjoint matrix for $SE(3)$.

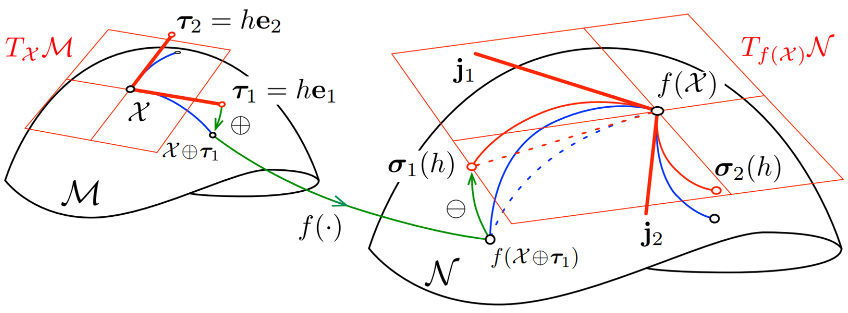
Some useful properties of the adjoint matrix include

$$\mathcal{X} \oplus \boldsymbol{\tau} = (\mathbf{Ad}_{\mathcal{X}}\boldsymbol{\tau}) \oplus \mathcal{X} \tag{2.26a}$$

$$\mathbf{Ad}_{\mathcal{X}^{-1}} = \mathbf{Ad}_{\mathcal{X}}^{-1} \tag{2.26b}$$

$$\mathbf{Ad}_{\mathcal{X}\mathcal{Y}} = \mathbf{Ad}_{\mathcal{X}}\mathbf{Ad}_{\mathcal{Y}} \tag{2.26c}$$

Note that for Eqs. (2.26b) and (2.26c), the left side of the equality is often cheaper to compute than the right side.



$T_{\mathcal{X}}\mathcal{M}$
$\boldsymbol{\tau}_2 = h\mathbf{e}_2$
$\boldsymbol{\tau}_1 = h\mathbf{e}_1$
$\mathcal{X}$
$\oplus$
$\mathcal{X}\oplus\boldsymbol{\tau}_1$
$\mathcal{M}$
$f(\cdot)$
$T_{f(\mathcal{X})}\mathcal{N}$
$\mathbf{j}_1$
$f(\mathcal{X})$
$\boldsymbol{\sigma}_1(h)$
$\boldsymbol{\sigma}_2(h)$
$\ominus$
$\mathbf{j}_2$
$f(\mathcal{X}\oplus\boldsymbol{\tau}_1)$
$\mathcal{N}$

Figure 2.9: Illustration showing how each direction $\boldsymbol{\tau}_i$ on manifold $\mathcal{M}$ relate to the directional derivative on manifold $\mathcal{N}$. Image from [46], licensed under CC BY-NC-SA 4.0.

2.4.6 Derivatives

Derivatives are essential in linearization and optimization, and it is, therefore, essential to define when using states on manifolds in state estimation. This section will present the derivatives in the form of Jacobian matrices mapping two vector tangent spaces. All Jacobians defined in this section fulfill the chain rule, so more complex Jacobians can easily be formed.

Before developing the theory for derivatives on Lie groups, a brief summary of Jacobians on Euclidean vector spaces will be given. For a multivariate function $f : \mathbb{R}^m \to \mathbb{R}^n$, the Jacobian matrix is defined as the $n \times m$ matrix with all the partial derivatives, i.e.

$$\mathbf{J} = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \triangleq \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix} \in \mathbb{R}^{n\times m} \tag{2.27}$$

This matrix can be further partitioned into $\mathbf{J} = [\mathbf{j}_1, \ldots, \mathbf{j}_m]$ where $\mathbf{j}_i = [\frac{\partial f_1}{\partial x_i}, \ldots, \frac{\partial f_n}{\partial x_i}]^\top$ are it's i-th column vector. This column vector is found using

$$\mathbf{j}_i = \frac{\partial f(\mathbf{x})}{\partial x_i} \triangleq \lim_{h\to 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h} \in \mathbb{R}^n \tag{2.28}$$

where $\mathbf{e}_i$ is the i-th standard basis vector[3] of $\mathbb{R}^m$. The columns of $\mathbf{J}$ thus represent the directional derivative along each dimension of $\mathbb{R}^n$. A more compact form of writing the full Jacobian is introduced for convenience. This compact form is

$$\mathbf{J} = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \triangleq \lim_{h\to 0} \frac{f(\mathbf{x} + \mathbf{h}) - f(\mathbf{x})}{\mathbf{h}} \in \mathbb{R}^{n\times m} \tag{2.29}$$

with $\mathbf{h} \in \mathbb{R}^m$. This is the same notation used in Eq. (2.27). This form is just a notational convenience as dividing by a vector $\mathbf{h}$ is undefined and proper computation of the Jacobian requires Eq. (2.28). Finally, for small values of $\mathbf{h}$, a linear approximation is found using

[3]The standard basis vector is the vector of all zeros but with the i-th element set to 1.

$$f(\mathbf{x}+\mathbf{h}) \xrightarrow{\mathbf{h}\to\mathbf{0}} f(\mathbf{x}) = \frac{\partial f(\mathbf{x})}{\partial x}\mathbf{h} = f(\mathbf{x}) + \mathbf{J}\mathbf{h} \tag{2.30}$$

For a function $f : \mathcal{M} \to \mathcal{N}$ acting on manifolds, the definition of the (right) Jacobian Eq. (2.29) is changed to use the right-$\oplus$ and -$\ominus$

$$\mathbf{J}_{\mathcal{X}}^{f(\mathcal{X})} \triangleq \frac{{}^{\mathcal{X}}\partial f(\mathcal{X})}{\partial \mathcal{X}} \triangleq \lim_{\boldsymbol{\tau}\to\mathbf{0}} \frac{f(\mathcal{X}\oplus\boldsymbol{\tau})\ominus f(\mathcal{X})}{\boldsymbol{\tau}} \in \mathbb{R}^{n\times m} \tag{2.31}$$

$$= \lim_{\boldsymbol{\tau}\to\mathbf{0}} \frac{\operatorname{Log}\left(f(\mathcal{X})^{-1}\circ f(\mathcal{X}\circ\operatorname{Exp}(\boldsymbol{\tau}))\right)}{\boldsymbol{\tau}} \tag{2.32}$$

Since the right-$\oplus$ and -$\ominus$ operators are used, this is called the *right* Jacobians.

The notation Eq. (2.31) conveys an intuition of what is going on: the Jacobian is the derivative of $f(\mathcal{X})$ with respect to $\mathcal{X}$, but the infinitesimal variation $\boldsymbol{\tau}$ is expressed in the tangent space at $\mathcal{X}$. In this light, the variations in $\mathcal{X}$ and in $f(\mathcal{X})$ are expressed as vectors in the local tangent spaces at $\mathcal{X} \in \mathcal{M}$ and $f(\mathcal{X}) \in \mathcal{N}$. The resulting Jacobian matrix is thus a linear mapping between variations in the *local* tangent spaces $T_{\mathcal{X}}\mathcal{M} \to T_{f(\mathcal{X})}\mathcal{N}$. This is illustrated in Fig. 2.9. The local property of the right Jacobian is indicated with the superscript $\mathcal{X}$ in Eq. (2.31). Many examples of Jacobians on common manifolds in state estimation are given in the appendices of [46].

Finally, for small values of $\boldsymbol{\tau}$, the following linear approximation also holds

$$f(\mathcal{X}\oplus{}^{\mathcal{X}}\boldsymbol{\tau}) \xrightarrow{{}^{\mathcal{X}}\boldsymbol{\tau}\to\mathbf{0}} f(\mathcal{X}) \oplus \frac{{}^{\mathcal{X}}\partial f(\mathcal{X})}{\partial \mathcal{X}}{}^{\mathcal{X}}\boldsymbol{\tau} \in \mathcal{N} \tag{2.33}$$

The derivatives can also be defined using the left-$\oplus$ and -$\ominus$ operators, yielding the *left* Jacobians. Contrary to the right Jacobians, now the infinitesimal increment $\boldsymbol{\tau} \in T_{\mathcal{E}}\mathcal{M}$ is given in the Lie algebra, and the numerator belongs to $T_{\mathcal{E}}\mathcal{N}$. Thus, the left Jacobian maps the *global* tangent spaces $T_{\mathcal{E}}\mathcal{M} \to T_{\mathcal{E}}\mathcal{N}$. The right and left Jacobians are also related by the adjoint matrices for $\mathcal{M}$ and $\mathcal{N}$. The left Jacobians will not be developed further here since only the right version of all operations is used in this thesis.

Lastly, crossed right-left Jacobians exist using the right-$\oplus$ and left-$\ominus$, or vice versa. However, these rarely come up in practice and are therefore omitted from this exposition. For the interested reader, refer to Section II-G.4 of Solà et al. [46].

2.4.7 Uncertainty

Uncertainty plays a central role in state estimation to quantify the accuracy of measurements and estimates. The final step required to use Lie theory for state estimation is to develop a way of representing the uncertainty of states on manifolds.

A local perturbation $\boldsymbol{\tau}$ around a point $\bar{\mathcal{X}} \in \mathcal{M}$ that lives in the tangent space $T_{\bar{\mathcal{X}}}\mathcal{M}$ can be defined using the right-$\oplus$ and -$\ominus$ like

$$\mathcal{X} = \bar{\mathcal{X}} \oplus \boldsymbol{\tau}, \qquad \boldsymbol{\tau} = \mathcal{X} \ominus \bar{\mathcal{X}} \in T_{\bar{\mathcal{X}}}\mathcal{M} \tag{2.34}$$

This can be used to properly define covariance matrices on the tangent space at $\bar{\mathcal{X}}$ through the standard expectation operator $E[\cdot]$ like

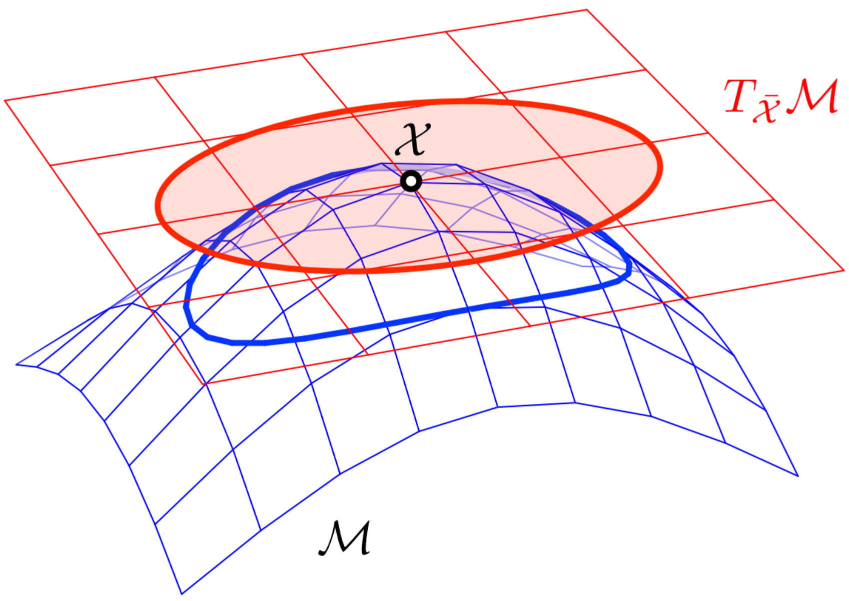


Figure 2.10: Illustration of how a covariance ellipse defined in the vector space at $\mathcal{X}$, and how it is wrapped over the manifold using $\oplus$. Image from [46], licensed under CC BY-NC-SA 4.0.

$$\boldsymbol{\Sigma}_{\mathcal{X}} \triangleq E[\boldsymbol{\tau}\boldsymbol{\tau}^{\top}] = E[(\bar{\mathcal{X}} \ominus \mathcal{X})(\bar{\mathcal{X}} \ominus \mathcal{X})^{\top}] \in \mathbb{R}^{m \times m} \tag{2.35}$$

With a definition of the covariance matrix, a Gaussian variable on manifolds can be defined as

$$\mathcal{X} \sim \mathcal{N}(\bar{\mathcal{X}}, \boldsymbol{\Sigma}_{\mathcal{X}}) \tag{2.36}$$

Notice that although the covariance matrix is written as $\boldsymbol{\Sigma}_{\mathcal{X}}$, the covariance actually describes the perturbation $\boldsymbol{\tau}$ in the tangent space. Figure 2.10 illustrate this by showing a Gaussian variable $\mathcal{X}$ with the covariance ellipse in the tangent space (in red). The covariance ellipse can be wrapped over the manifold using $\oplus$. This shows how increments that fall within the ellipse with a certain probability are mapped to the manifold. It is an important distinction that the uncertainty is defined in the tangent space for the uncertainty to be well-defined. If the dimension of the manifold d is greater than the degrees of freedom m, the covariance will be ill-defined. On the other hand, since the tangent space has dimension m, the covariance is well-defined.

Covariance can also be given in the global frame by using the left-$\oplus$ and -$\ominus$ operators. This way, the perturbation is given in the Lie algebra, i.e. the tangent space at the origin ${}^{\mathcal{E}}\boldsymbol{\tau} \in T_{\mathcal{E}}\mathcal{M}$ as

$$\mathcal{X} = \boldsymbol{\tau} \oplus \bar{\mathcal{X}}, \qquad \boldsymbol{\tau} = \mathcal{X} \ominus \bar{\mathcal{X}} \in T_{\mathcal{E}}\mathcal{M} \tag{2.37}$$

One use of specifying the covariance in the global frame is if the roll and pitch, but not the yaw, are known, thus having the 3D orientation known up to the horizontal plane. Then the covariance can be set to ${}^{\mathcal{E}}\mathbf{\Sigma} = \mathrm{diag}(\sigma_\phi^2, \sigma_\theta^2, \infty)$. Since the notion of "horizontal" is a global specification, the covariance must be given in the global frame.

The covariance matrix can also be transformed between the global and local frames using the adjoint using

$$
{}^{\mathcal{E}}\mathbf{\Sigma}_{\mathcal{X}} = \mathbf{Ad}_{\mathcal{X}} {}^{\mathcal{X}}\mathbf{\Sigma}_{\mathcal{X}} \mathbf{Ad}_{\mathcal{X}}^\top \tag{2.38}
$$

Moreover, covariance can also be propagated through a function $f : \mathcal{M} \to \mathcal{N};\ \mathcal{X} \mapsto \mathcal{Y} = f(\mathcal{X})$ by using the linearization Eq. (2.33) yielding the familiar formula

$$
\mathbf{\Sigma}_{\mathcal{Y}} \approx \frac{\partial f}{\partial \mathcal{X}} \mathbf{\Sigma}_{\mathcal{X}} \frac{\partial f}{\partial \mathcal{X}}^\top \in \mathbb{R}^{n \times m} \tag{2.39}
$$

2.4.8 Composite manifolds

A composite manifold $\mathcal{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_m\}$ is simply the concatenation of m non-interacting manifolds. This construction is useful when dealing with heterogeneous states or bundles of states. For example, a full bundle adjustment require a heterogeneous state with the robot's pose $\mathcal{X}_t \in SE(3)$ and L observed landmarks $\mathcal{X} = \{SE(3), \mathbb{R}^3_1, \ldots, \mathbb{R}^3_L\}$. Conversely, a motion-only bundle adjustment require a bundle of states $\mathcal{X}_t \in SE(3)$ over several time steps $\mathcal{X} = \{SE(3)_1, \ldots, SE(3)_T\}$.

The identity, inverse, and composition are defined element-wise for the composite manifold. For the composite manifold $\mathcal{M}$ above, we have

$$
\mathcal{E} \triangleq \begin{bmatrix} \mathcal{E}_1 \\ \vdots \\ \mathcal{E}_m \end{bmatrix}, \quad \mathcal{X}^{-1} \triangleq \begin{bmatrix} \mathcal{X}_1^{-1} \\ \vdots \\ \mathcal{X}_m^{-1} \end{bmatrix}, \quad \mathcal{X} \circ \mathcal{Y} \triangleq \begin{bmatrix} \mathcal{X}_1 \circ \mathcal{Y}_1 \\ \vdots \\ \mathcal{X}_m \circ \mathcal{Y}_m \end{bmatrix} \tag{2.40}
$$

The notation remains the same, and it should be clear from the context if the operations relate to a single manifold or a composite manifold.

All the other operators introduced earlier can also be applied element-wise in the same fashion. For example, the exponential and logarithmic maps become

$$
\mathrm{Exp}\{\boldsymbol{\tau}\} \triangleq \begin{bmatrix} \mathrm{Exp}(\boldsymbol{\tau}_1) \\ \vdots \\ \mathrm{Exp}(\boldsymbol{\tau}_m) \end{bmatrix}, \quad \mathrm{Log}\{\mathcal{X}\} \triangleq \begin{bmatrix} \mathrm{Log}(\mathcal{X}_1) \\ \vdots \\ \mathrm{Log}(\mathcal{X}_m) \end{bmatrix} \tag{2.41}
$$

These give the composite right-plus and -minus operators

$$
\mathcal{X} \oplus \boldsymbol{\tau} \triangleq \mathcal{X} \circ \mathrm{Exp}\{\boldsymbol{\tau}\} \tag{2.42}
$$

$$
\mathcal{Y} \ominus \mathcal{X} \triangleq \mathrm{Log}\{\mathcal{X}^{-1} \circ \mathcal{Y}\} \tag{2.43}
$$

These results makes it possible to define the composite Jacobian of a function $f : \mathcal{M} \to \mathcal{N}$:

$$\frac{\partial f(\mathcal{X})}{\partial \mathcal{X}} \triangleq \begin{bmatrix} \frac{\partial f_1}{\partial \mathcal{X}_1} & \cdots & \frac{\partial f_1}{\partial \mathcal{X}_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial \mathcal{X}_1} & \cdots & \frac{\partial f_m}{\partial \mathcal{X}_m} \end{bmatrix} \tag{2.44}$$

where each block $\frac{\partial f_i}{\partial \mathcal{X}_j}$ is calculated using Eq. (2.31). Moreover, the same results for approximating small perturbation Eq. (2.33) and for calculating covariance Eq. (2.35) has equivalent composite forms using the composite operators given above.

2.5 Probabilistic inference in robotics

In SLAM/VO applications, the robot's state, i.e., the unknown pose x and the position of the unknown landmarks l, is estimated using sensor measurements z with some associated uncertainty. The pose is continuous and multi-dimensional and can be represented by its probability density using the Lie theory from Section 2.4. It is also common to include the landmarks in the state vector such that $x \triangleq \{x, l\}$. In the remainder of this section, lowercase letters are used for random variables, and uppercase letters are used for a set of them.

Hence, to estimate the states and landmarks X from a set of observed measurements Z, we are interested in estimating the conditional density

$$p(X \mid Z) \tag{2.45}$$

This quantity is called the posterior, and estimating this quantity is called *probabilistic inference*.

This takes advantage of the Bayesian probability frameworks and can better utilize information about our belief and trust in specific estimates and sensor measurements. This is done by using the covariance information related to the different variables.

2.5.1 MAP inference

Maximum A-Posteriori (MAP) estimation is used to find the states X that maximizes Eq. (2.45) given the observed measurements. Using Bayes' law, the MAP estimate becomes

$$X^{MAP} = \arg\max_{X} p(X \mid Z) = \arg\max_{X} \frac{p(Z \mid X)p(X)}{p(Z)} \tag{2.46}$$

The quantity $p(Z \mid X)$ is called the likelihood and quantifies the chance of observing the observed measurements Z given that the robot is in the current state X. The quantity $p(X)$ is the priors on the states.

Since the denominator of Eq. (2.46) is only dependent on Z but the optimization is done over X, the denominator $p(Z)$ can be dropped. Furthermore, since only the posterior as a function of X is of interest, the likelihood function $l(X;\ Z) \propto p(Z \mid X)$ is defined to emphasize this fact. [13] An example of the likelihood function will be presented in Section 2.5.2. Thus, Eq. (2.46) can be rewritten as

$$X^{MAP} = \arg\max_X p(Z \mid X)p(X) = \arg\max_X l(X;\ Z)p(X) \tag{2.47}$$

2.5.2 MAP inference under Gaussian assumption

The multivariate Gaussian distribution is commonly used in state estimation in robotics. The probability density of the multivariate Gaussian distribution is

$$\mathcal{N}(x; \mu, \Sigma) = \frac{1}{\sqrt{|(2\pi)^n \Sigma|}} \exp\left\{ -\frac{1}{2} ||x - \mu||^2_\Sigma \right\} \tag{2.48}$$

where $\mu \in \mathbb{R}^n$ is the mean, $\Sigma \in \mathbb{R}^{n \times n}$ is the covariance matrix, $|\cdot|$ is the determinant, and

$$||x - \mu||^2_\Sigma = (x - \mu)^\top \Sigma^{-1}(x - \mu) \tag{2.49}$$

is the squared Mahalanobis distance. [13] The fraction coefficient in Eq. (2.48) is the normalization factor and ensures the integral over the state space equals 1.

In state estimation, it is convenient to model measurements as being corrupted by zero-mean Gaussian noise, i.e.

$$z = h(x) + \eta \qquad \eta \sim \mathcal{N}(0, R) \tag{2.50}$$

where $h(\cdot)$ is a *measurement prediction function*, and R is the covariance of the measurement noise. The conditional distribution of z therefore becomes

$$p(z \mid x) = \mathcal{N}(z; h(x), R) = \frac{1}{\sqrt{|2\pi R|}} \exp\left\{ -\frac{1}{2} ||h(x) - z||^2_R \right\} \tag{2.51}$$

The likelihood function introduced in Section 2.5.1 can, in the Gaussian case, be written as

$$l(x;\ z) = \exp\left\{ -\frac{1}{2} ||h(x) - z||^2_R \right\} \tag{2.52}$$

The measurement prediction function is derived from the known sensor geometry and is often nonlinear. Equation (2.52) is Gaussian in z, but not Gaussian in x due to the nonlinear measurement prediction function. Even if h were linear, the density would be degenerate because z is typically of lower dimension than x. For example, if the state included the x- and y-position and yaw of the robot, while the measurement is the range and bearing to a landmark, we would have $x \in \mathbb{R}^3$ and $z \in \mathbb{R}^2$. Hence, multiple measurements are needed to have a proper probability density and fully constrain all variables. If this is not achieved, the MAP estimate will fail because a unique maximizer of Eq. (2.47) is not available. [13]

Another common assumption is that the measurements Z are independent from each other. Under this assumption, the likelihood $p(Z \mid X)$ can be written as the product

$$p(Z \mid X) = p(z_1, \ldots, z_k \mid X) = \prod_{i=1}^{k} p(z_i \mid X) \tag{2.53}$$

Inserting this into Eq. (2.47) yields

$$X^{MAP} = \arg\max_X p(X) \prod_{i=1}^{k} p(z_i \mid X) \tag{2.54}$$

When both the priors and the likelihoods are Gaussian, this becomes

$$X^{MAP} = \arg\max_X \left(\exp\left\{ -\frac{1}{2} ||X - \mu_X||^2_{\Sigma_X} \right\} \prod_{i=1}^{k} \exp\left\{ -\frac{1}{2} ||h_i(X) - z_i||^2_{R_i} \right\} \right) \tag{2.55}$$

where the normalization coefficient is omitted.

Because the logarithm is a monotonically increasing function, applying it to the objective function of Eq. (2.55) (i.e., the expression inside the parenthesis) does not change the optimum. Conversely, the negative logarithm transforms the maximization problem into a minimization problem but keeps the same optimum. Thus, taking the negative logarithm of the objective function gives

$$\begin{aligned} X^{MAP} &= \arg\max_X \left(-\log\{Eq.\ (2.55)\}\right) \\ &= \arg\min_X \left(\frac{1}{2} ||X - \mu_X||^2_{\Sigma_X} + \sum_{i=1}^{k} \frac{1}{2} ||h_i(X) - z_i||^2_{R_i} \right) \end{aligned} \tag{2.56}$$

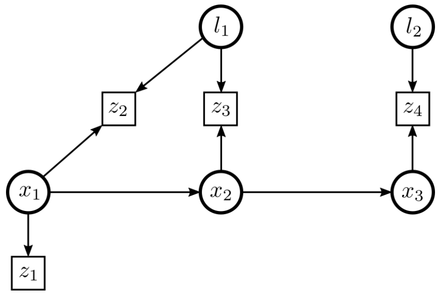
This is the familiar non-linear (weighted) least-squares problem and can be solved using any non-linear least-squares solver, e.g., Gauss-Newton and Levenberg-Marquardt. Each iteration of these algorithms solves a linearized problem to obtain an increment to move the current best estimate towards the optimum. The problem is linearized around this current best estimate, and it is therefore known as the *linearization point*.

In conclusion, when all the variables are assumed to be distributed according to the Gaussian distribution, the MAP estimate can be found by solving a Non-linear Least Squares (NLS) problem.

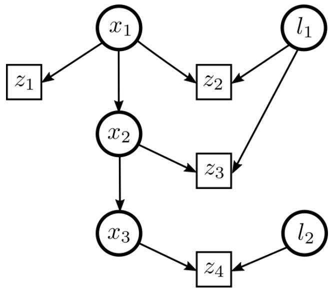
The MAP problem can also be formulated using the Lie theory presented in Section 2.4. Since increments, Jacobians, and probability distributions can be defined on the manifold, the exact derivations for the MAP solution given in this section can be followed for variables in a Lie group. The difference is that instead of the state $x \in \mathbb{R}^n$, we have $x \in \mathcal{N}$, where e.g. $\mathcal{N} = SE(3)$.

2.6 Bayes net

Bayesian network, also known as Bayes nets, is a graphical model for modeling inference problems. It is also a helpful way of thinking about how sensor measurements are generated and how they are affected by the robot's pose. Given the location of a landmark, the robot's pose when observing that landmark, and the geometry of how a sensor measurement is created, it is easy to predict what the measurement will be. If the landmark



(a) A Bayes net.



(b) One topological sort that clearly illustrate parent-child relationship.

Figure 2.11: A Bayes net of an example SLAM problem. The robot detects a landmark l_j from pose x_i which creates a measurement z_k. Note that the direction of the arrow point in the direction from the conditional parent variable, i.e. $p(x_2 \mid x_1)$ points *from* x_1 *to* x_2. Figure 2.11a is recreated from [13].

or robot moves, the measurement will be different. Furthermore, the uncertainty of that measurement is called a *noise model* and can be assumed or learned. These components constitute a conditional probability, which is the core of Bayes nets. This section will briefly introduce Bayes nets for modeling inference problems. It is primarily based on [13, sec. 5.4]. The subsequent sections will show how it is related to factor graphs and how it is used to update and solve nonlinear inference problems incrementally.

Formally, a Bayes net is a directed acyclic graph (DAG) where the nodes represent variables θ_j, and the edges represent conditional dependencies. The set of all random variables is denoted $\Theta = \{\theta_1, \dots, \theta_n\}$. A Bayes net then defines the joint probability function $p(\Theta)$ over all the variables in Θ as the product of conditional densities like

$$p(\Theta) \triangleq \prod_j p(\theta_j \mid \pi_j) \tag{2.57}$$

where π_j is an assignment of all the *parents* of θ_j. Since a DAG always can be topologically sorted [10], each variable θ_j will either have a set of parent variables on which it is conditionally dependent or have no parents, i.e., $\pi_j = \emptyset$ is the empty set. The density becomes a prior for θ_j. Hence, in a Bayes net, the factorization of the joint density is dictated by its graph structure and the conditional dependence relationships between nodes and their parents.

Figure 2.11a shows an example Bayes net for a simple SLAM problem. The random variables are $\Theta = \{X, Z\}$, i.e. the set of all unknown robot poses and landmarks $X = \{x_1, x_2, x_3, l_1, l_2\}$ as well as all the measurements $Z = \{z_1, \dots, z_4\}$. The poses and landmarks are represented using circles, and measurements are represented using boxes. This distinction is made since measurements are observed and therefore have no children. Their parents are the variables that affects the measurement. The arrows in Fig. 2.11a point in the direction from the conditional parent variables to the current variable. E.g., the arrow between x_1 and x_2 encodes the conditional probability $p(x_2 \mid x_1)$. Figure 2.11b shows one topological sort for the Bayes net, and clearly illustrate the relationship between the parents and children, where the parents are physically higher in the figure.

The joint density $p(X, Z)$ is the following product of conditional densities as per the definition of a Bayes net Eq. (2.57):

$$
\begin{aligned}
p(X, Z) &= p(x_1, x_2, x_3, l_1, l_2, z_1, z_2, z_3, z_4) && \text{(2.58a)} \\
&= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2) && \text{(2.58b)} \\
&\times p(l_1)p(l_2) && \text{(2.58c)} \\
&\times p(z_1 \mid x_1) && \text{(2.58d)} \\
&\times p(z_2 \mid x_1, l_1)p(z_3 \mid x_2, l_1)p(z_4 \mid x_3, l_2) && \text{(2.58e)}
\end{aligned}
$$

The division of the factors above illustrate four qualitatively different sets of densities:

- Equation (2.58b) represents a *Markov chain* on the poses. The conditional densities $p(x_{t+1} \mid x_t)$ can be information about the robot's movement derived from e.g. prior knowledge, known control inputs, or integration of inertial measurements.
- Equation (2.58c) represents *priors* on the landmarks' position. However, this is often omitted in SLAM when there is no prior map known.
- Equation (2.58d) represents an *absolute pose measurement* for the first pose x_1. This is essential as it determines the origin of the world frame.
- Equation (2.58e) represents the *measurements of the landmarks* from the robot. In this example, they are range-bearing measurements originating from e.g. a laser sensor.

These four types of densities are the most common in visual SLAM. However, other practical considerations are not represented here, like data association.

While Bayes nets are a convenient way of modeling and thinking about inference problems in robotics, better ways exist to solve the inference problems. Factor graphs are one way of doing this.

2.7 Factor graphs

A factor graph is another graphical model for modeling and solving inference problems in robotics. Similar to Bayes nets, factor graphs allow for intuitive modeling and representation of relationships between variables in a SLAM problem. Unlike Bayes nets, however, they are better suited for calculating the MAP estimate due to the non-Gaussian likelihood functions. [13] This section will give a brief introduction to factor graphs and how they can be leveraged in solving SLAM problems. It is based primarily on chapters 1 and 2 of Dellaert and Kaess [13].

Formally, a factor graph is a bipartite graph built up of two types of nodes: *variables* x_j and *factors* ϕ_i. The edges of the graph always connect variable nodes with factor nodes, and never connect nodes of the same type. The set of variable nodes adjacent to a

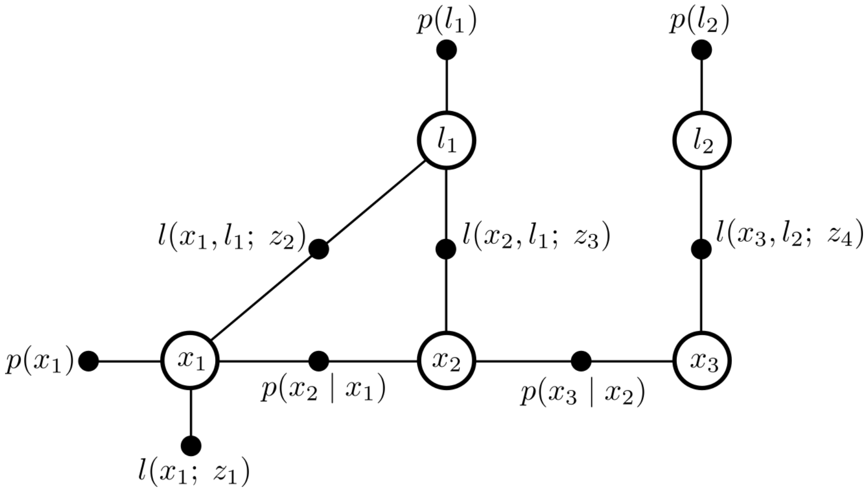


Figure 2.12: The factor graph representing the same example SLAM problem shown as a Bayes net in Fig. 2.11a. The factors as they are given in Eq. (2.60) are shown. This figure is adapted from Fig. 1.3 in [13].

factor node ϕ_i is written as $X_i = \mathcal{N}(\phi_i)$. These definitions defines the factorization of the function $\phi(X)$ as

$$\phi(X) = \prod_i \phi(X_i) \tag{2.59}$$

In other words, each factor is a function of only its neighboring variables, which encodes the variables' independence relationship, making up the entire graph. A factor graph can be used to specify any factored function $\phi(X)$ by individually modeling each factor. In inference problems, this is usually applied to a posterior density $p(X \mid Z)$.

Any Bayes net can trivially be converted to a factor graph. Recall that each node in a Bayes net denotes a conditional density for a variable and its parent nodes. Bayes net variable nodes, e.g., states and landmarks, are converted into a variable node in the factor graph and a factor node connecting a variable node and its parents. Bayes net measurement nodes are converted into a factor node connecting the variables it depends on, and the known measurement is given as a fixed parameter in the factor.

Consider the SLAM example given in Section 2.6. After conditioning the joint probability Eq. (2.58) on the measurements Z and employing the likelihood notation from Section 2.5.2, we get the following posterior

$$\begin{aligned} p(X \mid Z) &\propto p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2) && \text{(2.60a)} \\ &\times p(l_1)p(l_2) && \text{(2.60b)} \\ &\times l(x_1;\ z_1) && \text{(2.60c)} \\ &\times l(x_1, l_1;\ z_2)l(x_2, l_1;\ z_3)l(x_3, l_2;\ z_4) && \text{(2.60d)} \end{aligned}$$

This is converted to a factor graph following the procedure outlined above. It is clear that this represents a factored function with only unknowns, so each factor in Eq. (2.60) corresponds to a factor in the factor graph. This factorization is therefore represented as a factor graph as

$$\begin{aligned}\phi(x_1, x_2, x_3, l_1, l_2) &= \phi_1(x_1)\phi_2(x_2, x_1)\phi_3(x_3, x_2) && \text{(2.61a)}\\ &\times \phi_4(l_1)\phi_5(l_2) && \text{(2.61b)}\\ &\times \phi_6(x_1) && \text{(2.61c)}\\ &\times \phi_7(x_1, l_1)\phi_8(x_2, l_1)\phi_9(x_3, l_2) && \text{(2.61d)}\end{aligned}$$

This is also represented visually as can be seen in Fig. 2.12. The qualitative description of the different factors that was given for Eq. (2.58) hold true for the factors in Eq. (2.61).

A factor in the factor graph can analogously be thought of as a spring. The spring is connected between the variables in the factor, and the length of the spring is analogous to the measurement. For example, an odometry measurement constrains two variables to each other, so they cannot be too far apart or too close. Moreover, the covariance of the factor is analogous to the stiffness of the spring. The smaller the covariance, the stiffer the spring. When doing inference using the factor graph, we can think of the springs pulling on the variables to make them reach a state of equilibrium. This pulling corresponds to the iterative solving of the NLS problem.

2.7.1 Nonlinear MAP inference using factor graphs

Since factor graphs can represent a posterior density, it is a valuable construct for performing MAP inference. The MAP problem on a factor graph can be stated as

$$X^{MAP} = \arg\max_X \phi(X) = \arg\max_X \prod_i \phi_i(X_i) \tag{2.62}$$

As shown in Section 2.5.2, solving the MAP problem under the Gaussian assumption is the same as solving a NLS problem. Under the Gaussian assumption, each factor takes the form

$$\phi_i(X_i) \propto \exp\left\{-\frac{1}{2}||h_i(X_i) - z_i||^2_{\Sigma_i}\right\} \tag{2.63}$$

Following the same procedure as in Section 2.5.2 by taking the negative logarithm of the objective function gives the NLS formulation

$$X^{MAP} = \arg\min_X \sum_i ||h_i(X_i) - z_i||^2_{\Sigma_i} \tag{2.64}$$

It is crucial to attach a prior to the first pose variable to fix it in space. To understand why, imagine a factor graph consisting of only relative factors (e.g., odometry factors, range-bearing factors, etc.). Without a prior factor on the pose, the poses can be anywhere in the space since we only have relative information. By attaching a prior, we have a starting point from which the relative measurements can evolve from.

An important note is that the measurements z_i are typically of lower dimension than the unknown states X_i. Therefore, it is necessary to have more measurements to constrain and determine the problem fully. Whether or not the problem is fully determined can be understood by looking at the number of unknowns versus the number of equations.

For example, imagine a reduces version of the factor graph in Fig. 2.12 with only x_1, x_2, and l_1. Each state $x_1, x_2 \in SE(3)$ contribute 6 unknowns, and $l_1 \in \mathbb{R}^3$ contribute 3 unknowns, for a total of 15 unknown variables. Imagine that the factor connecting the states and the landmarks are projection factor. Then, they contribute two equations from the 2D pixel locations each, for a total of 4 equations. Moreover, the factor connecting the states is a relative pose measurement, i.e. $x_2 \ominus x_1 \in T_{x_1}SE(3)$, giving 6 equations. This is only a total of 10 equations. The remaining equations to fully determine the problem comes from the prior on x_1 that fixes the pose to the world frame. In total, there are 16 equations and 15 unknowns. Since there are more equations than unknowns, the problem is solvable.

Variable ordering is an important aspect of efficient solving of non-linear MAP problems. The variable ordering refers to the order of the unknown poses and landmarks in the vector X. In short, the order of the variables affects the sparsity of the matrices used to solve the NLS problem, and this significantly affects the speed at which the linear systems can be solved. However, this will not be explored in detail here as it is more relevant for efficient and practical implementations than for understanding how factor graphs can be used for robotic inference problems. The interested reader is referred to chapters 3 and 4 of Dellaert and Kaess [13] for more information.

2.7.2 Using factor graphs in code

Several libraries exist for using factor graphs in code. Since, under the Gaussian assumption, a factor graph can be reduced to a NLS problem, any NLS solver can be used. For example, the Google Ceres [1] is one library commonly seen in SLAM solutions. However, to facilitate easier modeling, other libraries have been developed.

GTSAM is one such library. It is implemented in C++ and is open-source, available on Github[4]. It enables the user to model the problem using a factor graph and to solve the problem to obtain the posterior density. Multiple predefined factors are already implemented, such as range-bearing factors, camera projection factors, odometry/between factors, and more. The user can also create custom factors by simply defining the measurement prediction function and the Jacobian with respect to each variable involved in the factor. This way, GTSAM can also evaluate complex Jacobians using the chain rule.

2.8 Incremental Smoothing and Mapping

Solving the full SLAM problem as shown in Section 2.7.1 is known as Smoothing and Mapping (SAM). Smoothing refers to the fact that the whole trajectory is estimated in

[4]`https://github.com/borglab/gtsam`

a batch update and, if applicable, also a map. This is opposed to filtering, which only keeps the latest pose. Smoothing is generally more accurate but at the cost of being more computationally demanding. Therefore, several ways of doing SAM incrementally have been developed.

Incremental Smoothing and Mapping (iSAM) is one way of solving SAM problems incrementally. It was originally introduced in Kaess et al. [31] using incremental matrix factorization but did relinearization by doing periodic batch updates. This is a sub-optimal way of doing relinearization because new measurements often have only a local effect [30], so batch updates perform many unnecessary calculations. iSAM2 was therefore introduced in Kaess et al. [30]. iSAM2 formulates the problem in terms of a Bayes tree and applies incremental variable reordering and fluid relinearization to alleviate the issues in the original iSAM formulation. This section will introduce the Bayes tree data structure before briefly explaining how it is applied in iSAM2.

2.8.1 Bayes tree

A factor graph can be converted to a Bayes net using the elimination algorithm (for details, see Algorithm 3.1 in Dellaert and Kaess [13]). A Bayes net from this process satisfies a special property called *chordal*, meaning that undirected cycles of length greater than three have a chord, i.e., an edge between non-consecutive vertices in the cycle [13]. It is still a Bayes net with the joint density Eq. (2.58), but can be rewritten as a tree by identifying cliques in the chordal net. The resulting tree is known as a Bayes tree.

Bayes trees were first introduced in Kaess et al. [29]. The nodes represents the cliques C_k of the underlying chordal Bayes net. The conditional density $p(F_k \mid S_k)$ is defined per node, where $S_k = C_k \cap \Pi_k$ are the separators and $F_k = C_k \setminus S_k$ are the frontal variables. The clique can be partitioned into these two types and written as $C_k = F_k \ : \ S_k$. [13] Figure 2.13b shows how the cliques are structured as a tree, and Fig. 2.13a shows the corresponding variables in the Bayes net from the factor graph in Fig. 2.12. The joint density $p(X)$ of the Bayes tree is given as

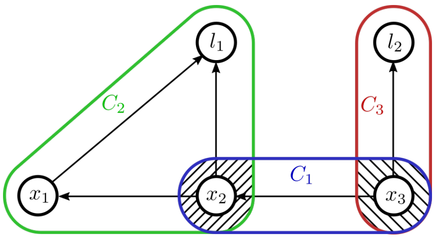
$$p(X) = \prod_k p(F_k \mid S_k) \tag{2.65}$$

The root $C_r = F_r$ have no separators because it have no parents, and corresponds to a simple prior $p(F_r)$.

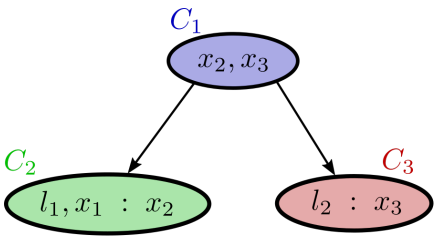
2.8.2 Incremental inference

Incremental inference corresponds to simply editing the Bayes tree. The key insight is that when a factor between x_i and x_j is added, it will only affect the paths between the clique containing x_i and x_j and the root clique. [29]

An example update to the Bayes tree in Fig. 2.13 is illustrated in Fig. 2.14, where a new factor is inserted between x_1 and x_3. First, the affected part highlighted in the top left is converted to a factor graph, and the new factor is inserted (top right). The factor graph is then converted to a chordal Bayes net (bottom right) and is lastly converted to a Bayes net, and the unmodified ”orphan” sub-tree is added back.



(a) Chordal Bayes net



(b) Corresponding Bayes tree

Figure 2.13: A chordal Bayes net generated from the factor graph in Fig. 2.12 and the corresponding Bayes tree. The cliques are marked and color-coded, and the separators are the areas of overlapping cliques marked with diagonal lines.

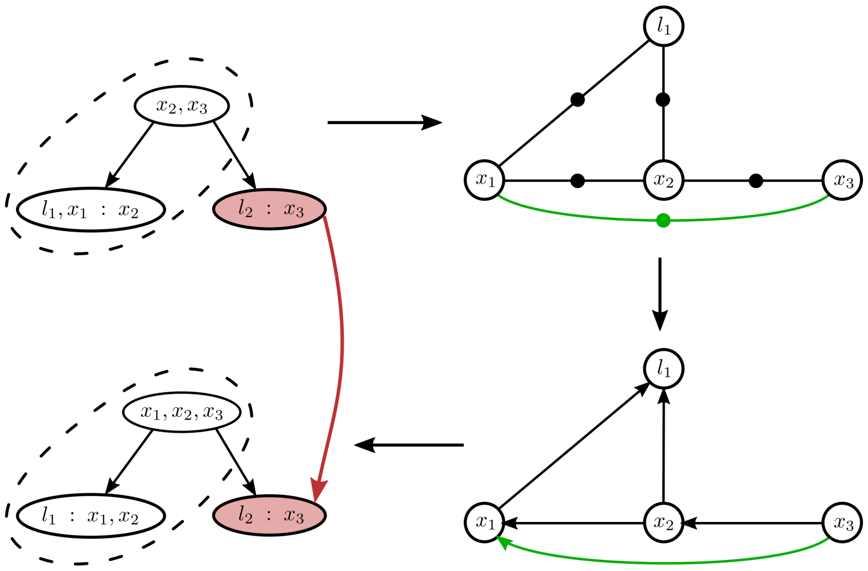


Figure 2.14: Bayes tree update illustrated. Note that x_1 has moved to the root clique and is now a separator in the child clique. The red child clique is unaffected and simply moved after the update. Figure inspired by Fig. 3 from Kaess et al. [29] and Fig. 5.6 from Dellaert and Kaess [13].

To get the state estimates after updating the Bayes tree, a delta vector $\boldsymbol{\Delta}$ is computed and added to the current linearization point X, i.e., $X + \boldsymbol{\Delta}$. This delta is found using back-substitution starting from the root clique and working towards the leaves. Since new updates typically only have a local effect, the back-substitution is stopped when Δ does not change the states by more than a given threshold.

2.8.3 Incremental variable reordering

As briefly mentioned in Section 2.7.1, the variable order is essential for solving the MAP problem efficiently. Finding the optimal order is NP-hard [30], so it is common to use a heuristic such as Column Approximate Minimum Degree (COLAMD). For incremental inference, Kaess et al. [30] found that the variables could be reordered after every incremental update. Since the update only affects a sub-tree of the full Bayes tree, COLAMD could be applied to the affected sub-tree. However, this will only minimize the fill-in, i.e., dependencies between variables that were previously independent, for the current step, and does not take into account the fact that recent variables are more likely to be affected in a new update. [30]

Therefore, a constrained COLAMD (CCOLAMD) is used, which forces the most recent access variables to the end of the variable ordering, i.e., to the root clique. This reduces the expected cost of incorporating new measurements because measurements typically have a local effect and will, therefore, most likely only affect the most recent variables.

2.8.4 Fluid relinearization

To avoid excessive computations when relinearizing, variables are only relinearized when needed. This is done by checking that variables in $\boldsymbol{\Delta}$ have a larger change than a given threshold and updating only those variables. [30] When a variable is relinearized, all cliques that contain the affected variable must be updated. However, this is significantly cheaper than updating the whole tree. [13]

2.9 IMU preintegration

IMU sensors are often used together with cameras in VIO systems due to the complementary nature of IMUs and cameras; cameras provide measurements of the scene geometry while IMUs renders metric scale and the gravity vector observable. [20] IMUs also provide inter-frame motion estimates because IMUs usually operate at a higher rate than cameras. However, this discrepancy in sensor rate proves challenging in tightly-coupled sensor fusion systems. If we were to build a factor graph where each node corresponds to each IMU measurement, the graph would grow too large and become intractable to do inference on.

To this end, a technique to only insert nodes for keyframes and add a motion factor summarizing the IMU measurements is used. This technique is called *IMU preintegration* and was originally proposed in [37]. It was later extended in Forster et al. [20] by exploiting the manifold structure of $SO(3)$ and $SE(3)$ groups using Lie theory. Figure 2.15

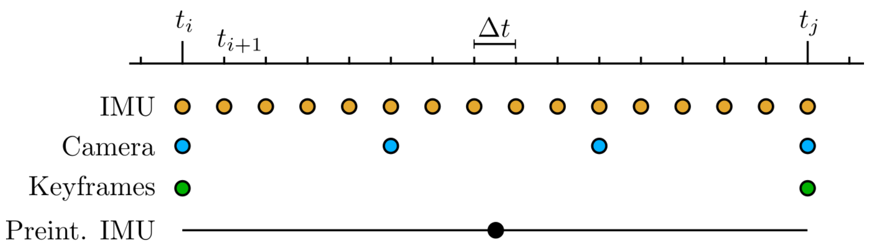


Figure 2.15: Illustration showing the different rates of the camera and IMU, and how the IMU preintegration factor is summarizing the IMU measurements between the keyframes. Based on Fig. 5 in [20].

shows an illustration showing the different sensor rates and how the preintegration factor summarizes the IMU measurements between two consecutive keyframes.

This section will only summarize the main results related to IMU preintegration from Forster et al. [20] and will not delve deep into the derivations. For a complete overview of the derivations, refer to the original paper Forster et al. [20] and its supplementary paper Forster et al. [21].

There are two problems associated with the preintegration of IMU measurements. The first is related to how to handle changes in the linearization point. The second is how to handle the bias. The former is solved by performing the integration in a local frame, making the result agnostic to changes in linearization point. The latter is solved by assuming the bias is constant between keyframes and using update equations when the bias change during optimization instead of reintegrating the measurements.

The IMU is assumed to be coincident with the body frame. Then, IMU measurements $\tilde{\mathbf{a}}^b(t)$ and $\tilde{\boldsymbol{\omega}}^b_{wb}(t)$ are given in the body frame $\mathcal{F}_b$ and are affected by additive white Gaussian noise $\boldsymbol{\eta}$ and a slowly varying bias $\mathbf{b}$

$$\tilde{\mathbf{a}}^b(t) = \mathbf{R}^\top_{wb}(t)(\mathbf{a}^w(t) - \mathbf{g}^w) + \mathbf{b}_a(t) + \boldsymbol{\eta}_a(t) \tag{2.66}$$

$$\tilde{\boldsymbol{\omega}}^b_{wb}(t) = \boldsymbol{\omega}^b_{wb}(t) + \mathbf{b}_g(t) + \boldsymbol{\eta}_g(t) \tag{2.67}$$

where the prefix subscript indicates the frame the quantity is expressed in. The state of the IMU is given by the $SE(3)$ element $[\mathbf{R}_{wb} \,|\, \mathbf{t}^w]$, with corresponding kinematic model

$$\dot{\mathbf{R}}_{wb} = \mathbf{R}_{wb}(\boldsymbol{\omega}^b_{wb})^\wedge \qquad \dot{\mathbf{v}}^w = \mathbf{a}^w \qquad \dot{\mathbf{t}}^w = \mathbf{v}^w \tag{2.68}$$

Let the time between IMU samples be given by Δt. Assume the acceleration and angular velocity are constant over the interval $[t, t + \Delta t]$. By applying forward Euler integration and inserting for true acceleration $\mathbf{a}^w$ and true angular velocity $\boldsymbol{\omega}^b_{wb}$ from Eqs. (2.66) and (2.67), the states at $t + \Delta t$ becomes

$$\mathbf{R}(t+\Delta t) = \mathbf{R}(t)\,\mathrm{Exp}\left((\tilde{\boldsymbol{\omega}}(t) - \mathbf{b}_g(t) - \boldsymbol{\eta}_{gd}(t))\Delta t\right) \tag{2.69a}$$

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \mathbf{g}\Delta t + \mathbf{R}(t)(\tilde{\mathbf{a}}(t) - \mathbf{b}_a(t) - \boldsymbol{\eta}_{ad}(t))\Delta t \tag{2.69b}$$

$$\mathbf{t}(t+\Delta t) = \mathbf{t}(t) + \mathbf{v}(t)\Delta t + \frac{1}{2}\mathbf{g}\Delta t^2 + \frac{1}{2}\mathbf{R}(t)(\tilde{\mathbf{a}}(t) - \mathbf{b}_a(t) - \boldsymbol{\eta}_{ad}(t))\Delta t^2 \tag{2.69c}$$

where the frame subscripts are dropped for readability. The frame should be unambiguous in the remainder of this section. The additive white noise is also converted to its discrete-time equivalent by dividing the covariance by Δt.

Equations (2.69a)–(2.69c) are valid for one IMU measurement. To create a single ”measurement” summarizing multiple measurements in the interval between the two keyframes at $k = i$ and $k = j$, the equations will be applied repeatedly. However, doing this directly will make the equations dependent on the state at time t_i. Hence, the equations are rewritten to be relative motion increments that are independent on the state at time t_i

$$\Delta\mathbf{R}_{ij} = \mathbf{R}_i^\top\mathbf{R}_j = \prod_{k=i}^{j-1}\mathrm{Exp}\left((\tilde{\boldsymbol{\omega}}_k - \mathbf{b}_{a,k} - \boldsymbol{\eta}_{gd,k})\Delta t\right) \tag{2.70a}$$

$$\Delta\mathbf{v}_{ij} = \mathbf{R}_i^\top(\mathbf{v}_j - \mathbf{v}_i - \mathbf{g}\Delta t_{ij}) = \sum_{k=i}^{j-1}\Delta\mathbf{R}_{ik}(\tilde{\mathbf{a}}_k - \mathbf{b}_{a,k} - \boldsymbol{\eta}_{ad,k})\Delta t \tag{2.70b}$$

$$\begin{aligned}\Delta\mathbf{t}_{ij} &= \mathbf{R}_i^\top(\mathbf{t}_j - \mathbf{t}_i - \mathbf{v}_i\Delta t_{ij} - \frac{1}{2}\mathbf{g}\Delta t_{ij}^2)\\ &= \sum_{k=i}^{j-1}\frac{3}{2}\Delta\mathbf{R}_{ik}(\tilde{\mathbf{a}}_k - \mathbf{b}_{a,k} - \boldsymbol{\eta}_{ad,k})\Delta t^2\end{aligned} \tag{2.70c}$$

where $\Delta t_{ij} \triangleq \sum_{k=i}^{j}\Delta t$, $\Delta\mathbf{R}_{ik} \triangleq \mathbf{R}_i^\top\mathbf{R}_k$, and $(\cdot)_k \triangleq (\cdot)(t_k)$ are the respective quantity evaluated at time t_k.

Equations (2.70a)–(2.70c) still depend on the biases and the noise terms. To deal with this, the bias terms are assumed to be known and the noise terms are isolated. Later, Forster et al. [20] develops update equations for when the bias changes.

Isolating the noise terms from Eqs. (2.70a)–(2.70c) using Eq. (2.33) yields the *preintegrated measurement model*:

$$\Delta\tilde{\mathbf{R}}_{ij} = \Delta\mathbf{R}_{ij}\,\mathrm{Exp}\left(\delta\phi_{ij}\right) \tag{2.71a}$$

$$\Delta\tilde{\mathbf{v}}_{ij} = \Delta\mathbf{v}_{ij} + \delta\mathbf{v}_{ij} \tag{2.71b}$$

$$\Delta\tilde{\mathbf{t}}_{ij} = \Delta\mathbf{t}_{ij} + \delta\mathbf{t}_{ij} \tag{2.71c}$$

where the compound measurements are the relative motion states from Eqs. (2.70a)–(2.70c) without the noise terms plus a random noise described by the random vector $\begin{bmatrix}\delta\phi_{ij}^\top & \delta\mathbf{v}_{ij}^\top & \delta\mathbf{t}_{ij}^\top\end{bmatrix}^\top$. More analysis of the noise terms in the random vector shows all terms can be written as a linear combination of zero-mean Gaussian densities, meaning the entire random vector can be characterized by

$$\begin{bmatrix}\delta\phi_{ij}^\top & \delta\mathbf{v}_{ij}^\top & \delta\mathbf{t}_{ij}^\top\end{bmatrix}^\top \sim \mathcal{N}(\mathbf{0}_{9\times 1}, \mathbf{\Sigma}_{ij}) \tag{2.72}$$

This analysis can be found divided across the paper Forster et al. [20] and the supplementary material Forster et al. [21].

Finally, the update terms are given. Given a bias update $\mathbf{b} \leftarrow \bar{\mathbf{b}} + \delta\mathbf{b}$, the compound measurements Eqs. (2.71a)–(2.71c) can be updated using a first-order expansion as

$$\Delta\tilde{\mathbf{R}}_{ij} \approx \Delta\tilde{\mathbf{R}}_{ij}(\bar{\mathbf{b}}_{g,i})\,\mathrm{Exp}\left(\frac{\partial\Delta\bar{\mathbf{R}}_{ij}}{\partial\mathbf{b}_g}\delta\mathbf{b}_g\right) \tag{2.73a}$$

$$\Delta\tilde{\mathbf{v}}_{ij} \approx \Delta\tilde{\mathbf{v}}_{ij}(\bar{\mathbf{b}}_{g,i}, \bar{\mathbf{b}}_{a,i}) + \frac{\partial\Delta\bar{\mathbf{v}}_{ij}}{\partial\mathbf{b}_g}\delta\mathbf{b}_{g,i} + \frac{\partial\Delta\bar{\mathbf{v}}_{ij}}{\partial\mathbf{b}_a}\delta\mathbf{b}_{a,i} \tag{2.73b}$$

$$\Delta\tilde{\mathbf{t}}_{ij} \approx \Delta\tilde{\mathbf{t}}_{ij}(\bar{\mathbf{b}}_{g,i}, \bar{\mathbf{b}}_{a,i}) + \frac{\partial\Delta\bar{\mathbf{t}}_{ij}}{\partial\mathbf{b}_g}\delta\mathbf{b}_{g,i} + \frac{\partial\Delta\bar{\mathbf{t}}_{ij}}{\partial\mathbf{b}_a}\delta\mathbf{b}_{a,i} \tag{2.73c}$$

where $\Delta\bar{(\cdot)}_{ij} = \Delta\tilde{(\cdot)}_{ij}(\bar{\mathbf{b}}_{g,i}, \bar{\mathbf{b}}_{a,i})$. The derivation for these update formulas can be found in Forster et al. [21, Sec. 1.2].

The preintegrated compound mesurements Eqs. (2.71a)–(2.71c) together with the noise characteristic Eq. (2.72) and bias update equations Eqs. (2.73a)–(2.73c) are all the required components to define the residuals that make up a between factor. The residuals $\mathbf{r}_{\mathcal{I}_{ij}} = [\mathbf{r}_{\Delta\mathbf{R}_{ij}}^\top, \mathbf{r}_{\Delta\mathbf{v}_{ij}}^\top, \mathbf{r}_{\Delta\mathbf{t}_{ij}}^\top]^\top \in \mathbb{R}^9$ are

$$\mathbf{r}_{\Delta\mathbf{R}_{ij}} = \mathrm{Log}\left(\left(\Delta\tilde{\mathbf{R}}_{ij}(\bar{\mathbf{b}}_{g,i})\,\mathrm{Exp}\left(\frac{\partial\Delta\bar{\mathbf{R}}_{ij}}{\partial\mathbf{b}_g}\delta\mathbf{b}_g\right)\right)^\top\left(\mathbf{R}_i^\top\mathbf{R}_j\right)\right) \tag{2.74a}$$

$$\begin{aligned}\mathbf{r}_{\Delta\mathbf{v}_{ij}} &= \mathbf{R}_i^\top(\mathbf{v}_j - \mathbf{v}_i - \mathbf{g}\Delta t_{ij}) \\ &\quad - \left[\Delta\tilde{\mathbf{v}}_{ij}(\bar{\mathbf{b}}_{g,i}, \bar{\mathbf{b}}_{a,i}) + \frac{\partial\Delta\bar{\mathbf{v}}_{ij}}{\partial\mathbf{b}_g}\delta\mathbf{b}_{g,i} + \frac{\partial\Delta\bar{\mathbf{v}}_{ij}}{\partial\mathbf{b}_a}\delta\mathbf{b}_{a,i}\right]\end{aligned} \tag{2.74b}$$

$$\begin{aligned}\mathbf{r}_{\Delta\mathbf{t}_{ij}} &= \mathbf{R}_i^\top(\mathbf{t}_j - \mathbf{t}_i - \mathbf{v}_i\Delta t_{ij} - \frac{1}{2}\mathbf{g}\Delta t_{ij}^2) \\ &\quad - \left[\Delta\tilde{\mathbf{t}}_{ij}(\bar{\mathbf{b}}_{g,i}, \bar{\mathbf{b}}_{a,i}) + \frac{\partial\Delta\bar{\mathbf{t}}_{ij}}{\partial\mathbf{b}_g}\delta\mathbf{b}_{g,i} + \frac{\partial\Delta\bar{\mathbf{t}}_{ij}}{\partial\mathbf{b}_a}\delta\mathbf{b}_{a,i}\right]\end{aligned} \tag{2.74c}$$

These residuals fit into the measurement prediction function and measurement formulation from Section 2.5.2, i.e. $r = h(x) - z$. In this context, the measurement prediction function predicts the relative change between two keyframes. For the position and velocity residuals, these are the expressions before the minus sign, and the expressions after the minus sign are the measurement. For the rotation, this is done on the manifold by using $\ominus$ to calculate the residual. The transposed expression to the left is the measurement, and the expression to the right is the measurement prediction.

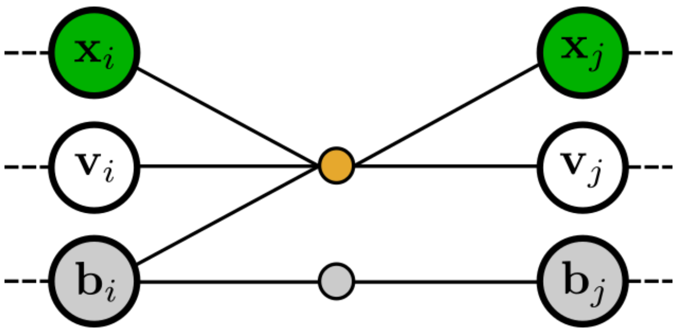


Figure 2.16: Exemplative piece of a factor graph using IMU preintegration factors. The orange circle represent the relative pose residuals Eqs. (2.74a)–(2.74c). The gray circle represent the bias residuals Eq. (2.77). Note that additional measurements are needed to fully constrain the problem as all factors from IMU preintegration are only relative measurements.

As previously mentioned, the bias is modeled as a slowly varying quantity. This is done by using a Brownian motion model, i.e., integrated white noise. This is also called random walk, and is expressed as

$$\dot{\mathbf{b}}_g(t) = \boldsymbol{\eta}_{bg}, \qquad \dot{\mathbf{b}}_a(t) = \boldsymbol{\eta}_{ba} \tag{2.75}$$

Integrating this over a time interval $[t_i, t_j]$ gives

$$\mathbf{b}_{g,j} = \mathbf{b}_{g,i} + \boldsymbol{\eta}_{bgd}, \qquad \mathbf{b}_{a,j} = \mathbf{b}_{a,i} + \boldsymbol{\eta}_{bad} \tag{2.76}$$

where $\mathbf{b}_{g,i} \triangleq \mathbf{b}_g(t_i)$. The discrete noises $\boldsymbol{\eta}_{bgd}$ and $\boldsymbol{\eta}_{bad}$ have zero mean and covariance $\boldsymbol{\Sigma}_{bgd} \triangleq \Delta t_{ij}\mathrm{Cov}(\boldsymbol{\eta}_{bg})$ and $\boldsymbol{\Sigma}_{bad} \triangleq \Delta t_{ij}\mathrm{Cov}(\boldsymbol{\eta}_{ba})$. This means the discrete covariance increase linearly from the continuous covariance with the length of the time interval.

Thus, the bias residual $\mathbf{r}_{\mathbf{b}_{ij}} \in \mathbb{R}^6$ between keyframes $k = i$ and $k = j$ is simply

$$||\mathbf{r}_{\mathbf{b}_{ij}}||^2 = ||\mathbf{b}_{g,j} - \mathbf{b}_{g,i}||^2_{\Sigma_{bgd}} + ||\mathbf{b}_{a,j} - \mathbf{b}_{a,i}||^2_{\Sigma_{bad}} \tag{2.77}$$

In summary, Eqs. (2.74a)–(2.74c) and (2.77) constitute the IMU preintegration factor between two keyframes $k = i$ to $k = j$. This is illustrated in Fig. 2.16.

2.10 Evaluation measures

To evaluate an odometry algorithm, its estimates can be compared against the ground truth or another trajectory estimate. This section will present the Absolute Pose Error (APE) that measures the global accuracy of an estimate, and the Root Mean Square Error (RMSE) which summarizes multiple errors in one scalar.

2.10.1 Root Mean Square Error

The Root Mean Square Error (RMSE) is often used to measure the difference between predicted values $\hat{\mathbf{x}}$ and observed/ground truth values $\mathbf{x}$. For N values, the RMSE is defined as

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} e_i^2} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{x}_i - \hat{x})^2} \tag{2.78}$$

where e_i is any error term, and $\hat{x}_i$ and x_i are the components of $\hat{\mathbf{x}}$ and $\mathbf{x}$, respectively. The reason for squaring and then taking the square root is so error terms of equal magnitude but opposite sign will not cancel each other.

2.10.2 Absolute Pose Error

The Absolute Pose Error (APE) is a measure to investigate the global accuracy and consistency of an odometry trajectory. [26] Given two poses, the ground truth $\mathbf{T}_{w,gt}(t) \in SE(3)$ and the odometry estimate $\mathbf{T}_{w,est}(t) \in SE(3)$, at the same timestamp t, the APE is defined as

$$\begin{aligned}\mathbf{APE}(t) &= \mathbf{T}_{w,est}(t) \ominus \mathbf{T}_{w,gt}(t) = \mathbf{T}_{w,est}(t)^{-1}\mathbf{T}_{w,gt}(t) \\ &= \mathbf{T}_{est,gt}(t) \in SE(3)\end{aligned} \tag{2.79}$$

where $\ominus$ is the (right) minus operator on the $SE(3)$ manifold introduced in Section 2.4.4. The APE can either be evaluated as a full pose error or it can be broken down into its translational and rotational components:

$$APE_{trans}(t) = ||\mathrm{trans}\{\mathbf{APE}(t)\}|| \tag{2.80}$$

$$APE_{rot}(t) = ||\,\mathrm{Log}\,(\mathrm{rot}\{\mathbf{APE}(t)\})|| \tag{2.81}$$

where $\mathrm{trans}\{\cdot\}$ and $\mathrm{rot}\{\cdot\}$ extract the translational and rotational part of the error, respectively. The rotational part gives the Rodrigues parameters of the rotation, where the magnitude is the angle rotated, and the normalized vector is the axis of rotation. [14]

Different statistics can be calculated on the APEs for all timestamps. For example, for RMSE in Eq. (2.78), the APEs Eqs. (2.80) and (2.81) are used as the error, e.g. $e_i = APE_{trans}(t_i)$ and $N = T$ the length of the dataset.

Chapter 3

STIO algorithm

This chapter will give an overview of the implementation details surrounding the modifications of the SVO front-end. It will also present the design of the GTSAM back-end. The modified SVO algorithm for 16-bit thermal image is dubbed Semi-direct Thermal-Inertial Odometry (STIO).

3.1 Semi-direct Visual Odometry

Semi-direct Visual Odometry (SVO) is a semi-direct front-end algorithm that utilizes direct methods to track and triangulate features and feature-based (indirect) methods for optimization of structure and motion. It also maintains a map of 3D landmarks. Two papers ([22] and [23]) have been released, where the second extends the first with additional feature types, multi-camera setup, and more. This section will introduce the original SVO algorithm from Forster et al. [22] and the relevant extensions from Forster et al. [23].

SVO is released as an open-source project on Github called `rpg_svo_pro_open`[1]. This repository and the theory from Forster et al. [23] is the basis for the modifications done in this thesis to make it work with 16-bit images.

3.1.1 Overview

SVO uses two parallel threads: the motion estimation thread does the front-end tracking to estimate the movement of the camera, and the mapping thread keeps track of the triangulated landmarks in a map of the environment. This division is illustrated in Fig. 3.1.

The motion estimation thread finds the relative pose between two consecutive images by minimizing the photometric error between pixels corresponding to the projected location of 3D landmarks in the sparse model-based image alignment step, see Fig. 3.2. The 2D locations are further refined in the feature alignment step by aligning the pose to the tracked landmarks in the map and with the previous keyframes, see Fig. 3.3. Finally,

[1]`https://github.com/uzh-rpg/rpg_svo_pro_open`

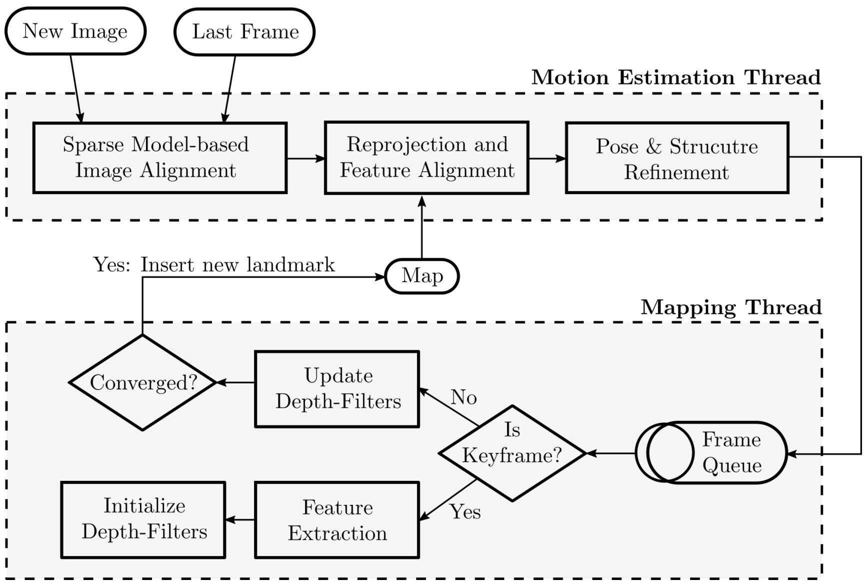


Figure 3.1: Overview of the SVO algorithm showing the two main threads. Figure adapted from original in Forster et al. [23].

the pose and structure are refined by minimizing the reprojection error introduced by the previous feature alignment by performing motion-only, structure-only, and full bundle adjustment, see Fig. 3.4.

The mapping thread initializes and updates probabilistic depth filters to estimate the 3D position of the 2D features. New depth filters are initialized when a new keyframe is selected. For every other image, the depth filters are updated in a Bayesian fashion using the correspondences found in the motion estimation thread. When the uncertainty in depth for a given point is small enough, the point is considered converged and is inserted into the map for use in motion estimation.

Keyframes are selected based on heuristics. The relative pose between the newest keyframe and all other keyframes is calculated. A new keyframe is selected if the relative Euclidean distance and the relative angle are greater than a configurable threshold.

In the subsequent sections, each step in the motion estimation thread and the depth filters are explained in more detail.

3.1.2 Sparse model-based image alignment

As mentioned above, the sparse model-based image alignment step finds the relative motion $\mathbf{T}_{k,k-1}$ between two consecutive images I_{k-1} and I_k. This is illustrated in Fig. 3.2. This is found by minimizing the sum of intensity residuals

$$\mathbf{T}_{k,k-1} = \arg\min_{\mathbf{T}} \sum_{\mathbf{u}\in\bar{\mathcal{R}}} \rho[\delta I(\mathbf{T},\mathbf{u})] \tag{3.1}$$

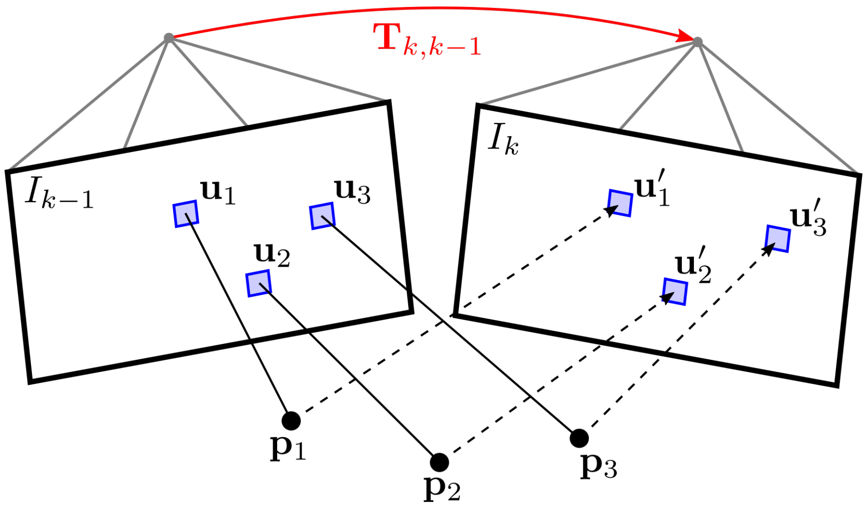


Figure 3.2: Illustration of the sparse image alignment step. The relative pose $\mathbf{T}_{k,k-1}$ in red is the parameter to optimize while the direct intensity error in blue between the two images are the optimization cost. Figure adapted from Fig. 2 in [22].

where the intensity residual δI is the photometric error between the pixels corresponding to the same 3D landmark, and $\rho[\cdot]$ is a cost function. Assuming the residuals are normally distributed with unit variance, this cost function becomes the 2-norm $\rho[\cdot] = \frac{1}{2}||\cdot||^2$, and the resulting optimum is the maximum likelihood estimate of the relative pose. In practice, the distribution has a heavier tail, so a more robust cost function such as the Huber norm must be used. The residual is defined as

$$\delta I(\mathbf{T}, \mathbf{u}) = I_k(\pi(\mathbf{T} \cdot \pi^{-1}(\mathbf{u}, d_\mathbf{u}))) - I_{k-1}(\mathbf{u}) \quad \forall \mathbf{u} \in \bar{\mathcal{R}} \tag{3.2}$$

where $\mathbf{u}$ is the 2D pixel location of the landmark in the previous image, which is back-projected and subsequently projected into the newest image. $\bar{\mathcal{R}}$ is the image region for which the depth is known at time $k-1$ and for which the back-projected points are visible in the newest image, i.e.

$$\bar{\mathcal{R}} = \left\{\mathbf{u} \mid \mathbf{u} \in \mathcal{R}_{k-1} \wedge \pi(\mathbf{T} \cdot \pi^{-1}(\mathbf{u}, d_\mathbf{u})) \in \Omega_k\right\} \tag{3.3}$$

where $\mathcal{R} \subseteq \Omega$ is the subset of the image domain Ω for which the depth is known.

Contrary to other direct methods, in SVO the depth $d_{\mathbf{u}_i}$ is known for sparse feature locations $\mathbf{u}_i$. To make the optimization less sensitive to noise, a small 4×4 patch around the feature location is used and is denoted as the vector $\mathbf{I}(\mathbf{u}_i)$. The residual Eq. (3.2) is then applied element-wise to each pixel in the patch and assumes the same depth for the entire patch. The optimization problem that is solved to find the relative pose then becomes

$$\mathbf{T}_{k,k-1} = \underset{\mathbf{T}_{k,k-1}}{\arg\min}\, \frac{1}{2} \sum_{\mathbf{u}_i \in \bar{\mathcal{R}}} ||\delta \mathbf{I}(\mathbf{T}_{k,k-1}, \mathbf{u}_i)||^2 \tag{3.4}$$

Figure 3.2 illustrates this optimization problem.

Equation (3.4) is nonlinear in $\mathbf{T}_{k,k-1}$ due to the projection functions in $\delta\mathbf{I}$. Thus, the optimization must be solved using an iterative nonlinear solver such as the iterative Guass-Newton algorithm. An incremental update $\mathbf{T}(\xi) \in SE(3)$ is parametrized with a twist $\xi \in \mathfrak{se}(3)$. In each iteration j, the increment is computed and applied to the current estimate using

$$\hat{\mathbf{T}}^{j+1}_{k,k-1} \leftarrow \hat{\mathbf{T}}_{k,k-1} \oplus -\xi^j = \hat{\mathbf{T}}^{j}_{k,k-1} \circ \mathbf{T}(\xi^j)^{-1} \tag{3.5}$$

where Eqs. (2.18c) and (2.21a) are used. The increment is computed using the intensity residual

$$\delta\mathbf{I}(\xi, \mathbf{u}_i) = \mathbf{I}_k(\pi(\hat{\mathbf{T}}_{k,k-1} \cdot \mathbf{p}_i)) - \mathbf{I}_{k-1}(\pi(\mathbf{T}(\xi) \cdot \mathbf{p}_i)) \tag{3.6}$$

where $\mathbf{p}_i = \pi^{-1}(\mathbf{u}_i, d_{\mathbf{u}_i})$. Note that the patches are not warped to increase computing speed. Warping can be neglected as the frame-to-frame motion is assumed to be small.

If a motion or rotation prior is available from another sensor modality, e.g., from an IMU, it can be used here as an initial guess to the optimization in Eq. (3.4). If no motion prior is available, the initial guess for $\mathbf{T}_{k,k-1}$ is set to the identity transformation. This is valid since the motion between two frames is relatively small. The sparse model-based image alignment is also done hierarchically on an image pyramid, where it starts on the coarsest level. This helps the patches align when there is no overlap at the full resolution.

Equation (3.6) is on the inverse compositional formulation from Baker and Matthews [2]. This formulation makes the Jacobian of Eq. (3.4) constant over all iterations of the Gauss-Newton algorithm. This results in a significant speed-up. The details surrounding the derivation of the Jacobian is left to the original SVO paper Forster et al. [22] and to Baker and Matthews [2].

3.1.3 Reprojection and feature alignment

The previous step aligned the new image to the previous frame to find the relative pose $\mathbf{T}_{k,k-1}$. Initial guesses for the features' pixel location in the new image were also found. However, due to inaccuracies in the landmarks' 3D positions and thus also the camera world pose, the initial guesses can be improved. The camera pose should also be aligned to the map rather than to the previous frame to reduce drift.

All 3D landmarks from the map that are visible from the estimated camera pose are projected into the image resulting in an estimate of the corresponding 2D feature positions $\mathbf{u}'_i$, as seen in Fig. 3.3. For each feature, the closest keyframe r is identified. The feature alignment step then optimizes all 2D feature positions $\mathbf{u}_i$ in the new image by minimizing the photometric error of a patch in the new image against a reference patch in the keyframe r. The cost function is therefore

$$\mathbf{u}'_i = \arg\min_{\mathbf{u}'_i} \frac{1}{2}||I_k(\mathbf{u}'_i) - \mathbf{A}_i \cdot I_r(\mathbf{u}_i)||^2 \quad \forall\, i \tag{3.7}$$

where $\mathbf{A}_i$ is an affine wrapping which accounts for the fact that a keyframe is typically further away from the new image than the previous image. Contrary to the previous step, a larger patch of 8×8 pixels is used in the feature alignment. The optimization Eq. (3.7) is solved using the inverse compositional Lucas-Kanade algorithm [2, sec. 3.2].

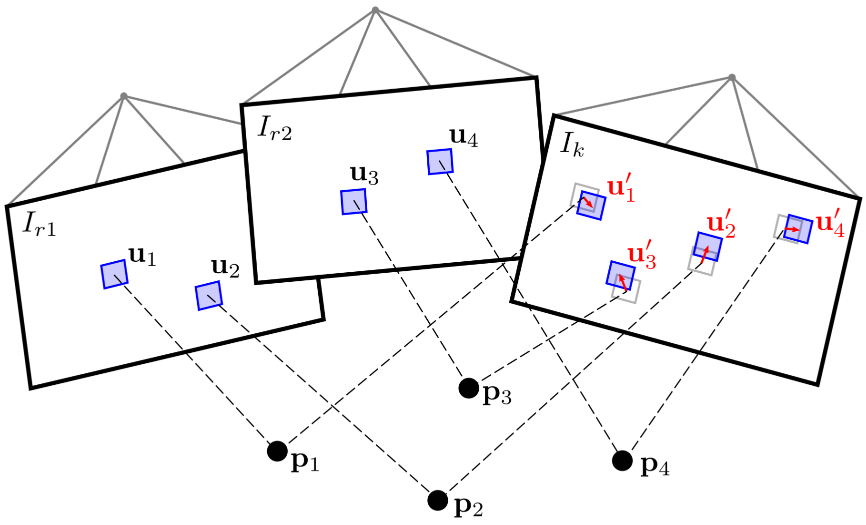


Figure 3.3: Illustration of the reprojection and feature alignment step. The pixel locations $\mathbf{u}'_i$ in red are the variables to optimize, and the intensity errors in blue are the optimization cost. Notice that the newest frame I_k is matched with the keyframes I_{r_j} and not the necessarily newest frames. Figure adapted from Fig. 3 in [22].

Forster et al. [23] introduced a new type of feature. Originally, only corners were used, but the new features allow for edgelet features as well, i.e., a small line segment on a straight edge. However, this cannot be optimized in two dimensions as it can be anywhere along the edge. Therefore, in solving Eq. (3.7), an edgelet feature is only optimized along the normal direction $\mathbf{n}$ of the edge it sits on. This means the update for $\mathbf{u}'_i$ becomes $\mathbf{u}'_i \leftarrow \mathbf{u}'_i + \delta u_i \cdot \mathbf{n}$ for the edgelet features, where $\delta u_i \in \mathbb{R}$. For corner features, the update equation becomes $\mathbf{u}'_i \leftarrow \mathbf{u}'_i + \mathbf{\Delta u}_i$.

The feature alignment step can be seen as a relaxation of the epipolar constraint to achieve subpixel accuracy of the features' positions.

3.1.4 Pose and structure refinement

The final step of the SVO algorithm is pose and structure refinement, illustrated in Fig. 3.4. This step will increase the accuracy of the pose estimate and the 3D landmarks in the map. This is done in two rounds. The first is motion-only bundle adjustment where only the pose $\mathbf{T}_{k,w}$ is optimized by solving Eq. (3.8). The second is structure-only bundle adjustment, where the 3D landmarks are optimized by solving Eq. (3.9) for the set of all 3D landmarks $\mathcal{P}^w$. In both cases, the optimization uses reprojection errors in the objective function instead of photometric errors like in the previous steps.

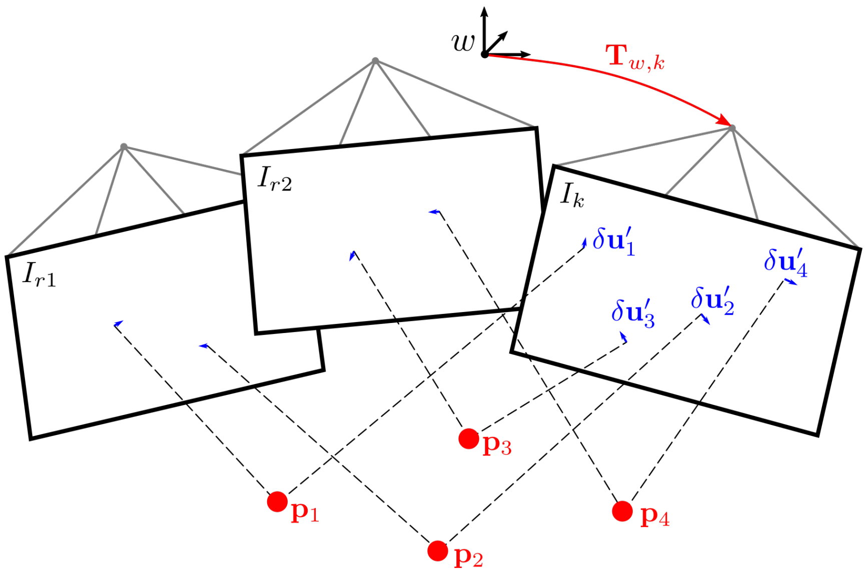


Figure 3.4: Illustration of the pose and structure refinement step. The world pose $\mathbf{T}_{w,k}$ in red and the 3D landmark positions $\mathbf{p}_i$ in red are optimized separately using the reprojection errors $\delta\mathbf{u}'_i$ in blue. Notice that the newest frame I_k is matches with the keyframes I_{r_j}, Figure adapted from Fig. 4 in [22].

$$\mathbf{T}_{k,w} = \underset{\mathbf{T}_{k,w}}{\arg\min} \frac{1}{2} \sum_i ||\mathbf{u}_i - \pi(\mathbf{T}_{k,w} \cdot \mathbf{p}_i^w)||^2 \tag{3.8}$$

$$\mathcal{P}^w = \underset{\mathcal{P}^w}{\arg\min} \frac{1}{2} \sum_i ||\mathbf{u}_i - \pi(\mathbf{T}_{k,w} \cdot \mathbf{p}_i^w)||^2, \quad \mathcal{P}^w = \{\mathbf{p}_i^w\} \tag{3.9}$$

An optional local bundle adjustment can be done after the motion- and structure-only bundle adjustments are done. All close keyframes and the 3D landmarks are jointly optimized in this local bundle adjustment. This will give greater accuracy but will increase computational cost.

The pose and structure refinement step is redundant when using SVO in a VIO system. In this scenario, SVO only acts as the front-end, and the back-end system will do the same as the pose and structure refinement step when the MAP problem is solved.

3.1.5 Probabilistic depth filters

The mapping thread estimates the depth of the observed features using probabilistic depth filters introduced in Vogiatzis and Hernández [53]. A depth filter is associated with a 2D feature, and the filter is updated in a Bayesian fashion using the subsequent frames that observe the same feature. Features being updated by the depth filters are called *seeds*. When the uncertainty is small enough, the depth estimate has converged, and the feature

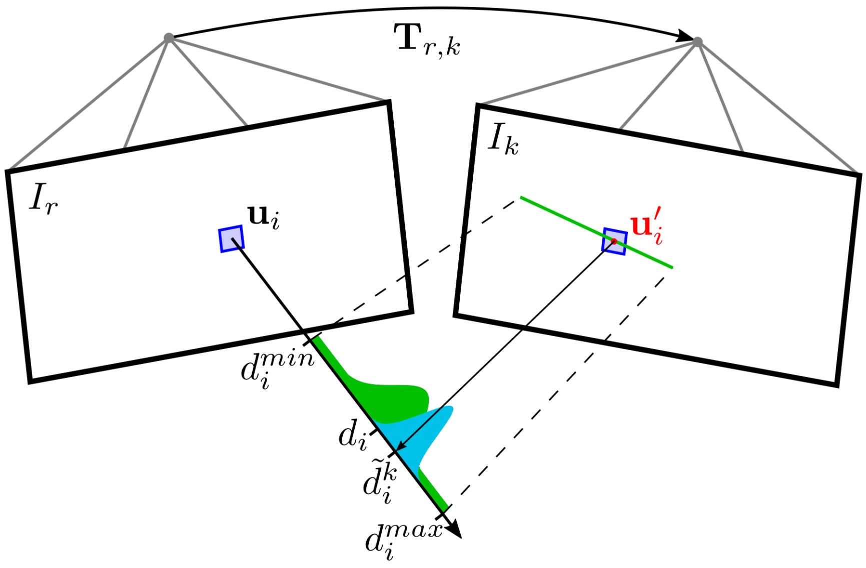


Figure 3.5: Illustration of the update of the depth filters. The green density is the measurement likelihood mixture model and the turquoise density is the posterior after the update. The green line is the epipolar line used for finding a match using the intensity error indicated by the blue squares. The red point is the feature in the newest frame I_k that is back-projected to create a depth measurement. Figure adapted from Fig. 5 in [22].

and seed are upgraded to a landmark by using the estimated depth to insert a 3D landmark into the map. The new landmarks are immediately used for motion estimation.

Every depth filter is associated with a keyframe r. When a new keyframe is selected, FAST features are detected. To keep an even distribution of features in the image, the image is divided into cells and the features are pruned to have one feature per cell. The pruning is done by selecting the FAST corner with the highest Shi-Tomasi score in each cell of a grid over the image unless there is already a tracked feature in that cell. If no corner feature can be found in a cell, an edgelet feature with the highest gradient is initialized instead. Corner features are preferred as they provide more information than edgelet features since corners are fixed in both dimensions. Filters are then initialized with high uncertainty in depth for every feature in the keyframe. For every subsequent frame $\{I_k, \mathbf{T}_{k,w}\}$ the patch on the epipolar line with the highest correlation to the reference patch around the feature $\mathbf{u}_i$ is found. The epipolar line is easily computed using the relative pose $\mathbf{T}_{r,k}$ and the ray passing through $\mathbf{u}_i$. The point of highest correlation $\mathbf{u}'_i$ is then used to triangulate a depth $\tilde{d}_i^k$.

The depth measurement $\tilde{d}_i^k$ from the triangulation is modeled as a Gaussian and Uniform mixture density. This models the measurement as being normally distributed around the true depth d_i while including the possibility of being an outlier measurement in the interval $[d_i^{min}, d_i^{max}]$ with uniform probability. The mixture density can be expressed as

$$p(\tilde{d}_i^k \mid d_i, \rho_i) = \rho_i \mathcal{N}(\tilde{d}_i^k \mid d_i, \tau_i^2) + (1 - \rho_i)\mathcal{U}(\tilde{d}_i^k \mid d_i^{min}, d_i^{max}) \tag{3.10}$$

where ρ_i is the inlier probability and τ_i^2 is the variance assuming an inlier measurement. τ_i^2 can be computed by assuming a one pixel variance on the pixel location $\mathbf{u}_i'$ and back-projecting the variance to get a variance for the depth measurement. [41]

The depth filter is updated using recursive Bayesian estimation. This is explained in Vogiatzis and Hernández [53], and the main result will be summarized here with an adapted notation to match the rest of the SVO material. Assume all the measurements $\tilde{d}_i^1, \dots, \tilde{d}_i^n$ are independent. With Eq. (3.10) as the measurement likelihood, the posterior of the true depth d_i and inlier probability ρ_i has the form

$$p(d_i, \rho_i \mid \tilde{d}_i^1, \dots, \tilde{d}_i^n) \propto p(d_i, \rho_i) \prod_k p(\tilde{d}_i^k \mid d_i, \rho_i) \tag{3.11}$$

where $p(d_i, \rho_i)$ is the prior for true depth, and the inlier probability is assumed to be uniform.

Since the posterior is multiple mixture models multiplied together, it has no simple expression and must therefore be modeled using a dense 2D histogram. However, due to memory and computational limitation, this is not feasible for real-time applications. Thus, a parametric approximation is derived.

The posterior Eq. (3.11) was found to be close to uni-modal. Therefore, a uni-modal approximation is used. A good approximation is the mixture model of a Beta distribution for inlier probability and a Gaussian distribution for the depth. This can also be shown to have the smallest Kullback Leibler divergence from the true posterior and is therefore a good fit. The approximate posterior is therefore defined as

$$q(d_i, \rho_i \mid a_n, b_n, \mu_n, \sigma_n) \triangleq Beta(\rho_i \mid a_n, b_n)\mathcal{N}(d_i \mid \mu_n, \sigma_n^2) \tag{3.12}$$

where a_n and b_n relate to how many inlier and outlier measurements have occurred, and μ_n and σ_n^2 is the mean and variance of the Gaussian depth estimate.

Finally, the new posterior q^+ after a new measurement would not have the same distribution as Eq. (3.12). This is solved using moment matching. The parameters of the approximate, parametric posterior q are calculated such that the first and second-order moments for d_i and ρ_i match the new posterior q^+. The depth filter update process is illustrated in Fig. 3.5.

3.1.6 Initialization

The front-end is initialized by using the five-point relative pose algorithm from Nister [39] to obtain the pose of the first two keyframes and an initial map. In broad terms, this algorithm finds the relative pose between the keyframes from the essential matrix using only (minimum) five point correspondences. The correspondences are found by tracking keypoints using the Kanade-Lucas-Tomasi (KLT) tracker algorithm [36, 51]. The average disparity in pixels is calculated, and when this is above a certain threshold, the five-point algorithm is run, and the front-end is considered initialized.

3.2 STIO front-end modifications

The front-end component of STIO builds upon the front-end from SVO. The front-end is modified to work with 16-bit images, both in the image matching steps and feature detection. This section will present the modifications done to the front-end.

3.2.1 16-bit matching

Due to the limitation of re-scaling the 16-bit thermal image to 8-bit discussed in Section 2.1, we want to have the front-end support the full 16-bit images directly. This required changing the direct matching code for the sparse image alignment and the feature alignment steps. It also required changing the feature detection code.

The direct matching code was implemented using pointer arithmetic and reinterpret casting in C++. In short, the image data is represented as an array of bytes. For 8-bit images, one byte corresponds to one pixel. When the array is indexed to retrieve a pixel, this is simply an offset from the base pointer that point to the first pixel. Hence, the pixel at index `i` has the memory address `base_pointer + i`. On the other hand, for 16-bit images, we have two bytes per pixel. If we indexed the array as an array of bytes, the resulting pointer location would not correspond to a pixel. Casting the image byte data to a `uint16_t` data type tells C++ to interpret the bytes of the array as 16-bit wide integers. Then, when indexing the array, the pointer location points to the desired pixel since each index increments the pointer by two bytes. The use of pointer arithmetic and reinterpret casting made the modification to 16-bit matching simple. The challenge was correctly identifying all the relevant places in the code that needed to be changed.

3.2.2 Image alignment gain and offset estimation

When a thermal camera does FFC as described in Section 2.1, there can occur a sudden and significant shift in intensity values. This breaks the assumption of little to no intensity change between consecutive images in the sparse image alignment step of the SVO front-end. This can be a major problem in tracking features between frames and need to be remedied. The SVO implementation includes illumination gain and offset estimation that can account for large changes in illumination. This section presents the theory behind this as it is implemented in the code of `rpg_svo_pro_open`.

The sparse image alignment minimizes the photometric error (5) from Forster et al. [22]. To introduce illumination gain and offset estimation, the error term is modified to be

$$\delta I(\mathbf{T}, \mathbf{u}) = I_k(\pi(\mathbf{T} \cdot \pi^{-1}(\mathbf{u}, d_{\mathbf{u}}))) - (a \cdot I_{k-1}(\mathbf{u}) + b) \tag{3.13}$$

where $a, b \in \mathbb{R}$. a is the estimated illumination gain and b is the estimated illumination offset. The gain increase ($a > 1$) or decrease ($a < 1$) the contrast in the image. The offset increase ($b > 0$) or decrease ($b < 0$) the brightness of the image.

The optimization (7) from Forster et al. [22] is also duly modified to optimize with respect to a and b as well, i.e.

$$\{\mathbf{T}_{k,k-1}, a, b\} = \underset{\substack{\mathbf{T}_{k,k-1}\\ a,b}}{\arg\min} \frac{1}{2} \sum_{i \in \bar{\mathcal{R}}} ||\delta I(\mathbf{T}_{k,k-1}, \mathbf{u}_i)||^2 \tag{3.14}$$

where the addition of a and b in the list of optimized variables is emphasized. Although these values are optimized, their final values are of no interest and is not used in later calculations. They are merely added to aid the sparse image alignment optimization.

3.2.3 Feature detection on histogram equalized images

One way the front-end was modified to support 16-bit images was to do feature detection on a histogram equalized image. This is inspired by ROTIO [32]. Since ROTIO only does feature detection when it needs new features, feature detection is only dependent on the contrast in a single frame and does not need consistency over multiple frames. Exploiting this fact, the contrast is improved using histogram equalization as seen in Section 2.1.1 and the FAST corner detector is used on the resulting 8-bit image.

Since SVO also only detects new features when a new keyframe is selected, the same technique is applied. Feature detection in SVO is done on an image pyramid. Hence, when new features are needed, the frame is firstly histogram equalized, then an image pyramid is created from the equalized image. This order of operations is important to ensure that the smaller pyramid levels' intensities are not equalized differently from the source image. The FAST corner detector is then run on the equalized image pyramid to find distinct features.

3.2.4 16-bit FAST corner detection

Another way the front-end is modified to support 16-bit images was to modify the FAST corner detector to work directly on the full 16-bit image. The SVO implementation uses the library `fast_neon`[2] for the FAST feature detection. This library uses machine-generated code based on pointer arithmetic to access the image values. This made it easy to convert to 16-bit using the same technique as for the direct matching code described above. This allows us to use the 16-bit image pyramid in the feature detection step.

To distinguish the two implementations of the FAST corner detector, the detector working directly on 16-bit images will be called FAST16 and the standard detector working on 8-bit images will be called FAST8.

3.3 GTSAM back-end implementation

The SVO implementation in `rpg_svo_pro_open` includes a VIO back-end based on Open Keyframe-based Visual-Inertial SLAM (OKVIS) [35]. This uses a factor graph formulation that is solved as an NLS problem (see Section 2.7.1). OKVIS is therefore implemented using the Ceres Solver [1], which is a general-purpose NLS solver.

[2]`https://github.com/uzh-rpg/fast_neon`

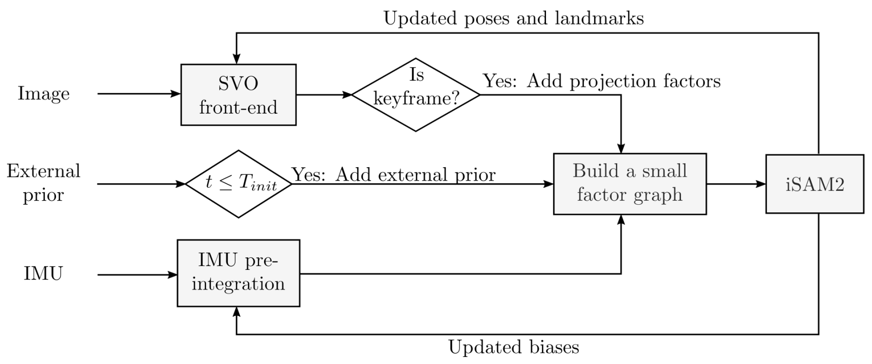


Figure 3.6: Schematic overview of the GTSAM back-end showing what information is passed to which module at which time.

While Ceres is a good choice to *solve* large scale NLS problems, a back-end implemented in GTSAM can simplify future development. Being made for modeling and solving problems formulated as a factor graph, adding, e.g., new sensor modalities or loop closure constraints will be quick and easy. GTSAM also have several common SLAM factors already implemented, thus reducing the amount of code required for new functionality. For these reasons, a simple back-end was implemented using GTSAM. This section will present the design of the back-end.

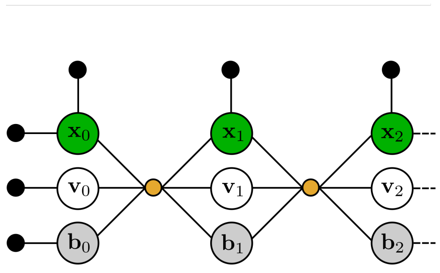
In the remainder of this thesis, the back-end based on Ceres will be referred to as the *Ceres back-end*, while the back-end based on GTSAM will be referred to as the *GTSAM back-end*.

3.3.1 Overview

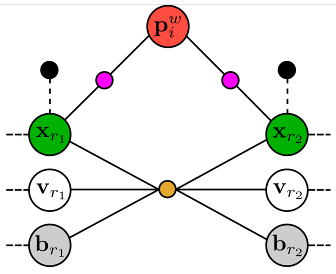
Figure 3.6 shows a schematic overview of the GTSAM back-end. IMU measurements are integrated by the IMU preintegrator according to the theory presented in Section 2.9. To ease initialization of the IMU biases, external priors are added for the first T_{init} seconds and for the two first keyframes used for initializing the front-end. IMU preintegration factors are added between each prior. This is explained in more detail in Section 3.3.3. After T_{init} seconds, the SVO front-end start processing the images. For each keyframe it selects, projection factors are added for all the visible landmarks in that keyframe. IMU preintegration factors are added between each keyframe. This is explained in more detail in Section 3.3.2. This builds a small factor graph that can be added to the iSAM2 optimizer, and one iteration of the iSAM2 algorithm is run. The updated results for poses and landmarks are used to update the position of all keyframes and landmarks tracked by the front-end. Likewise, the IMU preintegration is reset and updated with the newest biases.

3.3.2 Factor graph

This section will present the small factor graph that is created and added to the iSAM estimator. Figure 3.7b shows the factor graph for two keyframes r_1 and r_2. An IMU



(a) IMU preintegration factors and external pose priors.



(b) Projection factors and IMU preintegration factors

Figure 3.7: Illustration of the factor graph design. (a) shows the initialization system where the IMU preintegration factors (orange circles) and the pose priors (black circles) are run initially for a predetermined duration. (b) shows how projection factors (pink circles) are added between keyframes and a landmark, and how an IMU preintegration factor is added between two keyframes.

preintegration factor is added between the state variables $\mathbf{x}$, $\mathbf{v}$, and $\mathbf{b}$ for each keyframe. Landmarks are added as projection factors between landmark i and the keyframe r_j that observed it. The following paragraphs will detail each factor type and how they are implemented in code.

The projection factor creates a constraint between a landmark and the pose from which the camera observes that landmark. The projection function uses the pinhole projection function Eq. (2.10) to create a measurement prediction. The measurement $\mathbf{u}_i$ is the feature's final pixel location after the sparse model-based image alignment step from Section 3.1.2 and the feature alignment step from Section 3.1.3. Hence, for a landmark $\mathbf{p}_i^w$ observed at pose $\mathbf{x}_{r_j}$ and a given measurement covariance $\mathbf{\Sigma}_i^{proj} \in \mathbb{R}^{2\times 2}$, the projection factor have the form

$$\phi_{ij}^{proj}(\mathbf{x}_{r_j}, \mathbf{p}_i^w) \propto \exp\left\{-\frac{1}{2}||\mathbf{u}_i - \pi(\mathbf{p}_i^w;\ \mathbf{T}_{w,c_j})||^2_{\mathbf{\Sigma}_i^{proj}}\right\} \tag{3.15}$$

Note that the pose is of the *camera* is used in the projection function, and this can be calculated using the pose of the body frame and the body to camera extrinsic calibration using $\mathbf{T}_{w,c_j} = \mathbf{T}_{w,r_j}\mathbf{T}_{b,c}$. The measurement covariance $\mathbf{\Sigma}_i^{proj} = \sigma\mathbb{I}_{2\times 2}$ where $\mathbb{I}_{2\times 2}$ is the 2×2 identity matrix. Since the feature pixel location can be at different levels of the image pyramid, we set $\sigma = 2^{l_i}$ where l_i is the level at which the feature is extracted. The full image is level 0, so features extracted at that resolution has a covariance of 1 pixel.

In the code, this is implemented using GTSAM's `GenericProjectionFactor`. This factor constrains the landmark's 3D position with the pose using the projection on the camera at that pose. It is important to provide the body to camera extrinsic calibration as the argument of the constructor `body_P_sensor` $= \mathbf{T}_{b,c}$.

Projection factors are only added for landmarks that have two or more observations from keyframes. This is because landmarks with only one observations is not constrained enough to be observable, and can cause the optimization to fail.

The IMU measurements are preintegrated according to the theory presented in Section 2.9. In GTSAM, there are two ways of doing IMU preintegration. The old way splits the relative motion from the preintegration and the random walk of the bias into two factors. The new way creates one factor that includes the preintegration residuals and the bias random walk residuals. In the code, the new way is used, and that is what will be presented here.

New IMU measurements are preintegrated using the GTSAM class that combines IMU and bias into one, `PreintegratedCombinedMeasurements`. This class implements the preintegration theory presented in Section 2.9 and stores the relative motion increments Eqs. (2.70a)–(2.70c). Similarly to the projection factor, if the IMU frame is not coincident with the body frame, an extrinsic body to IMU calibration must be provided.

To turn the relative motion increments into a between factor, the preintegrated measurement object is given to the `CombinedImuFactor` class. This creates one factor that constrain the pose and velocity states using the relative motion increment residuals Eqs. (2.74a)–(2.74c), and the bias states using the bias random walk residuals Eq. (2.77). The white square in Fig. 3.7b illustrate how this factor connects the states to each other.

After running the optimizer to find the newest estimate, the IMU preintegration is reset and given the newest biases. This needs to be done because the preintegration only saves relative information. If the integration were not reset, the next factor would contain the relative motion information between the latest keyframe and the third latest keyframe, and not between the latest and the penultimate keyframe.

3.3.3 Initialization using external pose priors

Initialization is an integral part of any VIO system. It establishes the scale of the system to achieve correct estimation of depth of landmarks during triangulation. It also gives the IMU a good starting point by finding initial estimates of the biases. [43] For monocular VIO systems, it is common to perform an initialization motion in which all degrees of freedom are excited to let the IMU biases converge.

Monocular initialization can be a complicated process, so a shortcut used in this thesis to focus on the development of the rest of the back-end. The shortcut is to use an external pose system to give accurate poses as priors and insert IMU preintegration factors between the pose priors. These pose priors can come from a motion capture system or another state estimation pipeline using. Regardless of the source of the pose prior, the initialization routine is the same. The goal of this initialization is to let the IMU biases converge and to have some nodes in the factor graph that determine the scale of the system.

Figure 3.7a shows the initialization using external pose priors. Without processing the image frames, the IMU measurements are preintegrated and added as constraints between two nodes. Since keyframes are not selected yet, new nodes are added at every frame. At each node, a prior is added to the pose. The priors are assumed to be approximately ground truth, thus giving them a small covariance. This graph is optimized, and estimates for the poses, velocities, and biases are obtained. We let this system run for a predetermined duration T_{init}. Afterward, the front-end is started, and the system is run like any other VIO system. Pose priors are added to the first two keyframes as seen in Fig. 3.7b to help set the scale in the problem.

3.4 Datasets

To test the 16-bit thermal front-end, three datasets with thermal image data and IMU data were used. These were originally captured for testing the KTIO algorithm [33]. They are captured using a MAV fitted with FLIR's Tau2 LWIR camera and a VectorNav VN100 IMU. The Tau2 camera provides thermal images at 640×512 resolution at 20 Hz, and the VN100 IMU provides measurements at 200 Hz. The thermal camera is intrinsically calibrated using a custom thermal checkerboard [40], and temporo-spatially calibrated with the IMU using *Kalibr* [24] Moreover, an Ouster OS1-64 Light Detection and Ranging (LiDAR) sensor is also placed on-board the MAV to provide a LiDAR odometry estimate using the LiDAR Odometry and Mapping (LOAM) algorithm [57] to compare against. The following sections will briefly introduce each of the thermal datasets.

For developing and testing the GTSAM back-end, the European Robotics Challenge (EuRoC) datasets [6] were used. This contains synchronized stereo images, IMU measurements, and ground truths from motion capture systems and laser scanners. Only the left image stream is used in the GTSAM back-end.

3.4.1 Vicon Arena

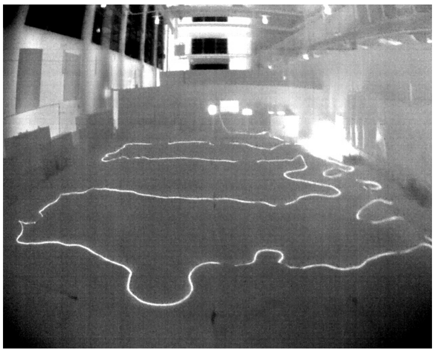
This dataset was captured in a constructed indoor scene where the MAV is flying in a rectangular pattern at a fixed height. The room was completely dark, and the scene was furnished with thermally active objects such as a space heater, a computer, and heating cables on the floor. Figure 3.8a shows a frame of the whole scene, and Fig. 3.8b shows a close-up of the right corner with the space heater and computer are placed. Lastly, the room was also equipped with a Vicon motion capture system that provides ground truth pose measurements.

Figure 3.8c and Fig. 3.8d shows the frame just before and just after the thermal camera performs FFC. These is a noticeable change in contrast and illumination between the two frames.

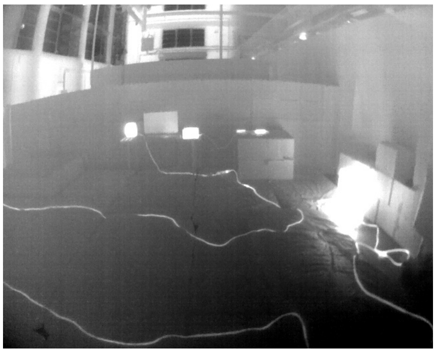
3.4.2 Urban Parking Lot

This dataset was captured in a parking garage at the University of Nevada, Reno, USA. Figure 3.9 shows some example frames from the dataset. Several cars had recently had their engines running which is evident from the heat emitting from the car hoods. There is also some contrast in the image from the roof and the floor having different emissivity. However, Figs. 3.9a, 3.9b and 3.10c shows large thermally flat areas taking up the majority of the frame, with the high gradient areas being concentrated to the edges of the frame.

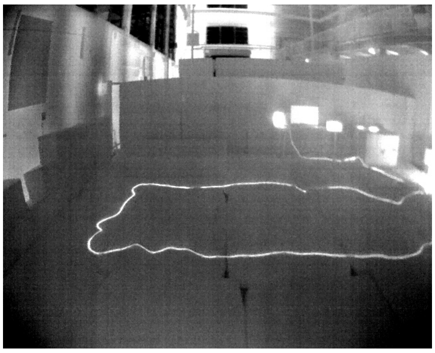
The urban parking lot has an abundance of good LiDAR features, and the LOAM estimate can therefore be considered as ground truth. This will not be discussed further here, but the interested reader is referred to the author's specialization project [17] for more details.



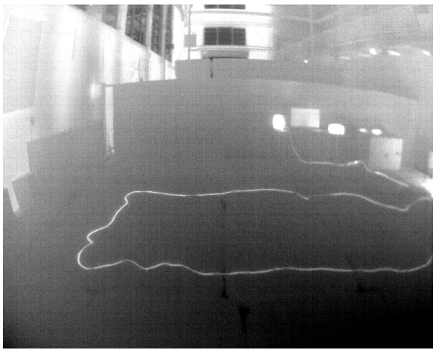
(a) Whole Vicon room scene.



(b) Close-up of right corner with space heaters.

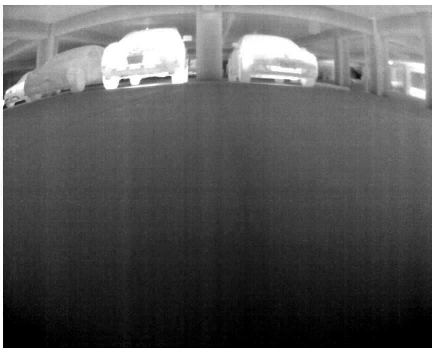


(c) Before FFC.

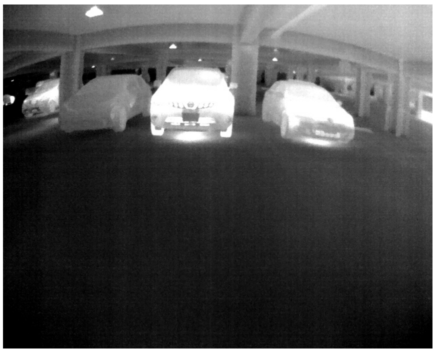


(d) After FFC.

Figure 3.8: Four frames from the Vicon arena thermal dataset. Notice the illumination change between Fig. 3.8c and Fig. 3.8d. The illumination change is most notable on the left and far wall. Note that the images are re-scaled using histogram equalization as described in Section 2.1.1.

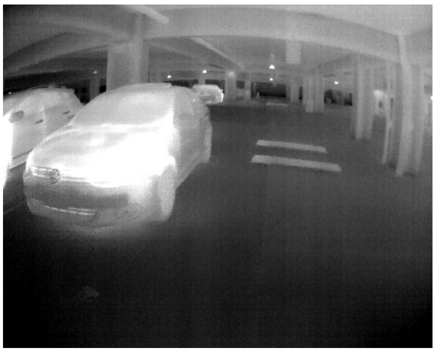


(a) View before taking off.



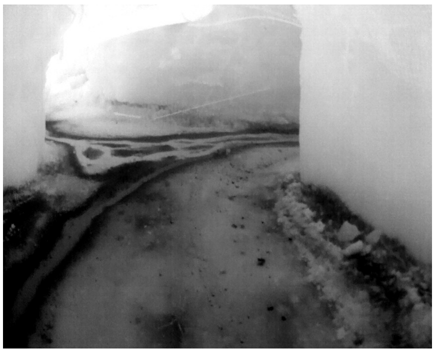
(b) Thermal hot spots from hot cars.

(c) Large thermally flat areas in the frame.

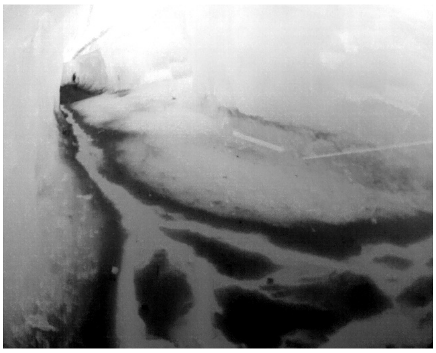


(d) Some spacers on the ground with different emissivity than the ground.

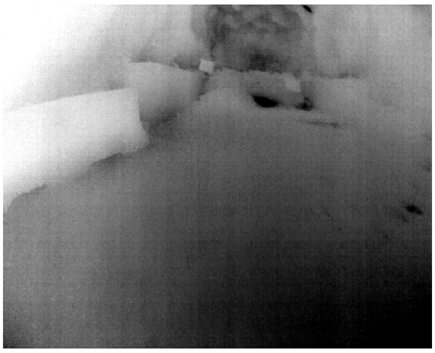
Figure 3.9: Four example frames from the Urban Parking Lot dataset. Notice that a lot of the frame is taken up by the uniformly heated ground that is largely without distinct features.



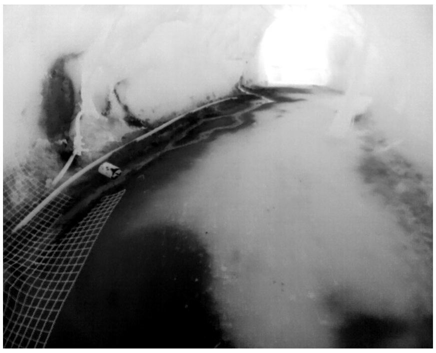
(a) Starting area.



(b) Hallway after starting area.



(c) Thermally flat corner.



(d) Another hallway frame.

Figure 3.10: Four example frames from the Active Gold Mine dataset. Notice the large thermally flat walls with lack of distinct features.

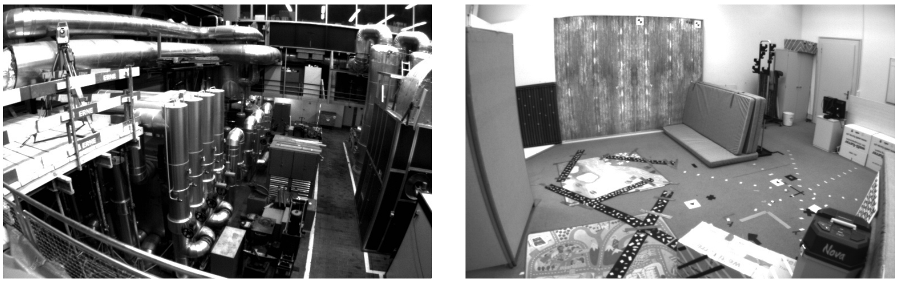
3.4.3 Active Gold Mine

This dataset was captured in an active underground gold mine in northern Nevada, USA. Figure 3.10 shows some example images from the dataset. The walls are noticeable thermally flat, but the ground in the hallways seen in Fig. 3.10b and Fig. 3.10d has more variety where the puddles and the ground emit different levels of thermal radiation. On the other hand, Fig. 3.10c shows a thermally flat corner with little contrast. This dataset also includes obscurants from airborne dust. [33]

The active gold mine has some straight hallways that are not ideal for LiDAR odometry, but the hallways are interrupted by frequent junctions that make the geometry of the mine not self-similar. This means the LOAM estimate is close to the ground truth.

3.4.4 EuRoC

The EuRoC datasets were recorded in two different environments. In each environment, datasets of varying difficulty were recorded. The more difficult datasets include faster and more aggressive motion. The datasets were recorded using a MAV with an IMU and



(a) EuRoC Leica (Machine Hall) **(b)** EuRoC Vicon

Figure 3.11: Example frames from the EuRoC datasets environments.

stereo camera setup. The MAV also had external motion capture markers. For all datasets, ground truth for position, velocity, and IMU biases are given from a batch optimization solution. The datasets were recorded by Burri et al. [6] at ETH Zürich and are available online.[3]

The first environment is a machine hall at ETH Zürich. An example image is shown in Fig. 3.11a. A Leica laser positioning system provides position measurements but not orientation measurements. The easy dataset ”Machine Hall 01” will be used in this work. This dataset will be referred to as the *Leica dataset*.

The second environment is in a Vicon motion capture room with extra textured objects for easy feature detection. Figure 3.11b shows an example image. In the Vicon room, full pose measurements are available from the Vicon motion capture system. The easy dataset ”Vicon Room 1 01” will be used in this work. This dataset will be referred to as the *Vicon dataset*.

Furthermore, the Leica dataset includes an initialization movement that excites the IMU along each axis. This is important for monocular systems to converge the biases. On the other hand, the Vicon dataset does not include this motion.

3.5 Experiments

3.5.1 16-bit front-end

To tune, evaluate, and compare the 16-bit front-end modifications in isolation, the front-end will be run alone, i.e., without help from the IMU. This will look qualitatively at the difference in feature extraction, how robust the features are, and the impact of noise by comparing the FAST16 corner detector versus the FAST8 corner detector on histogram equalized images. The full 16-bit radiometric thermal image will be used for inter-frame matching in both cases.

The two 16-bit methods will be evaluated on the thermal datasets presented in Section 3.4. The Vicon Arena will test the front-end in conditions with a thermally diverse

[3]`https://projects.asl.ethz.ch/datasets/doku.php?id=kmavvisualinertialdatasets`

scene, while the Urban Parking Lot and the Active Gold Mine will test in real-life conditions but with different environments.

3.5.2 STIO with Ceres back-end

Using the experiences from the front-end experiments, the STIO system using Ceres back-end will be run on the same datasets. This will test the performance of the TIO system using both the FAST16 detector and the FAST8 detector on histogram equalized images.

The estimated poses will be compared against ground truth from a Vicon motion capture system in the Vicon Arena. In the Urban Parking Lot and the Active Gold Mine, LiDAR odometry algorithm LOAM will be used in place of the Vicon motion capture ground truth.

The estimates will also be compared to another TIO system ROTIO [32, sec. IV-B]. This is a modified version of the Robust Visual-Inertial Odometry (ROVIO) VIO algorithm [3]. The modifications are the same as STIO when using FAST8 on histogram equalized images, i.e., feature detection on histogram equalized images, but direct matching on the full 16-bit images.

As a control, the unmodified 8-bit SVO front-end with the Ceres back-end will be used solely on histogram equalized images for both matching and feature extraction.

The evaluation metrics introduced in Section 2.10 will be used to assess the accuracy and consistency of the STIO estimates. The open-source implementation EVO[4] [26] will be used to calculate the metrics and create the plots.

3.5.3 IMU and external prior initialization

The initialization system using preintegrated IMU measurements and external priors will be tested to investigate the observability of the IMU biases. This will be tested using both a full pose prior and only a position prior using the EuRoC datasets. The EuRoC Vicon room dataset has full pose priors, and the machine hall dataset only has position priors.

3.5.4 GTSAM back-end

Lastly, the GTSAM back-end will be tested on the EuRoC datasets. Despite much debugging effot, the author could not make the GTSAM back-end work. The results for the GTSAM back-end will therefore be partial. The focus will instead be on trying to understand why it does not work.

[4] `https://github.com/MichaelGrupp/evo`

Chapter 4

Results

This chapter will present the results from the experiments introduced in Section 3.5. The 16-bit thermal front-end will be presented first, followed by the STIO results. Then, the biases from the IMU preintegration and external priors will be presented, and lastly, partial qualitative results from the failed GTSAM back-end will be presented.

However, before delving into the results, a quick note on naming. The 16-bit thermal front-end has two variants: 16-bit matching and 16-bit feature detection using FAST16, and 16-bit matching and 8-bit feature detection using FAST8 on histogram equalized images. To avoid tedious repetitions and over-explanation, the former will be referred to as simply *FAST16*, and the latter as *FAST8*. They will be talked about as ”different front-ends,” but it is here reiterated that they only differ in the feature *extraction* method. Moreover, whether ”FAST” refers to the front-end or the feature detection algorithm will be clear from the context. The same naming convention will be used in the STIO results. Lastly, the original SVO front-end using the Ceres back-end applied to histogram equalized images only will be referred to as the ”8-bit STIO front-end”.

4.1 16-bit thermal front-end

4.1.1 Vicon Arena

Figure 4.1 shows the number of tracked features and average disparity during initialization. The sawtooth shape resulted from the initializer selecting a new reference frame when the number of tracked features fell below a predetermined threshold. Features were detected on this new reference frame, causing the increase in features before slowly decreasing because some features were not tracked in the following frames. The selection of a new reference frame also caused spikes in the disparity plot. The fact that the front-end lost so many features suggests that some detected features were not features of the scene but rather false-positive features originating from noise and other artifacts in the image.

Both front-ends initialized at approximately the same time. The two plots are slightly shifted because some false-positive matches by FAST16 caused just enough features to be

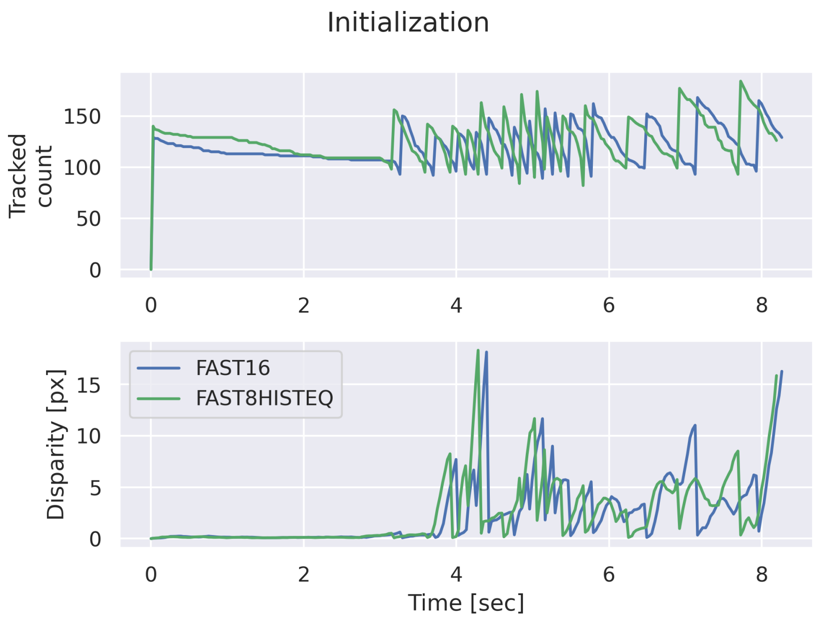


Figure 4.1: Initialization data using only the front-end on the Vicon Arena dataset.

lost, and a new reference frame was selected.

The initialization disparity threshold was 15 for both front-ends. This was found to be a good trade-off between being large enough for good point cloud initialization but small enough for easy initialization. Increasing the threshold rendered the front-end unable to initialize in a reasonable time. Decreasing it made the tracking perform poorly due to a bad initial point cloud. Lastly, the front-end did not initialize at around 4 seconds, despite the average disparity being larger than 15, because there were not enough tracked points.

Figure 4.2 shows, from top to bottom, the number of tracked features in the sparse model-based image-alignment step, the number of matches in the reprojection step, and the total number of tracked landmarks, converged seeds not yet inserted as landmarks, and unconverged seeds. Both front-ends tracked the same number of features and landmarks. On the other hand, FAST8 was tracking more seeds than FAST16. As Fig. 4.4 try to show, FAST8 detected more seeds than FAST16. For the most part, the seeds were short-lived and did not move with the scene, confirming that they were false-positive detections. This was likely due to noise build-up in the sensor that was exaggerated by the histogram equalization. A close look at Fig. 4.4b also shows some horizontal and vertical lines that were sensor artifacts and could be a contributing factor to the noisy seed initialization.

FAST16 and FAST8 did, for the most part, detect and track the same features. However, as Fig. 4.3 shows, FAST16 detected features in the fully saturated areas of the frame, i.e. the hottest areas in the scene. This is a space heater with vertical grills that give it some texture in the full 16-bit image that FAST16 can use as features, but FAST8 cannot. However, the total number of tracked features in the two frames was approximately equal.

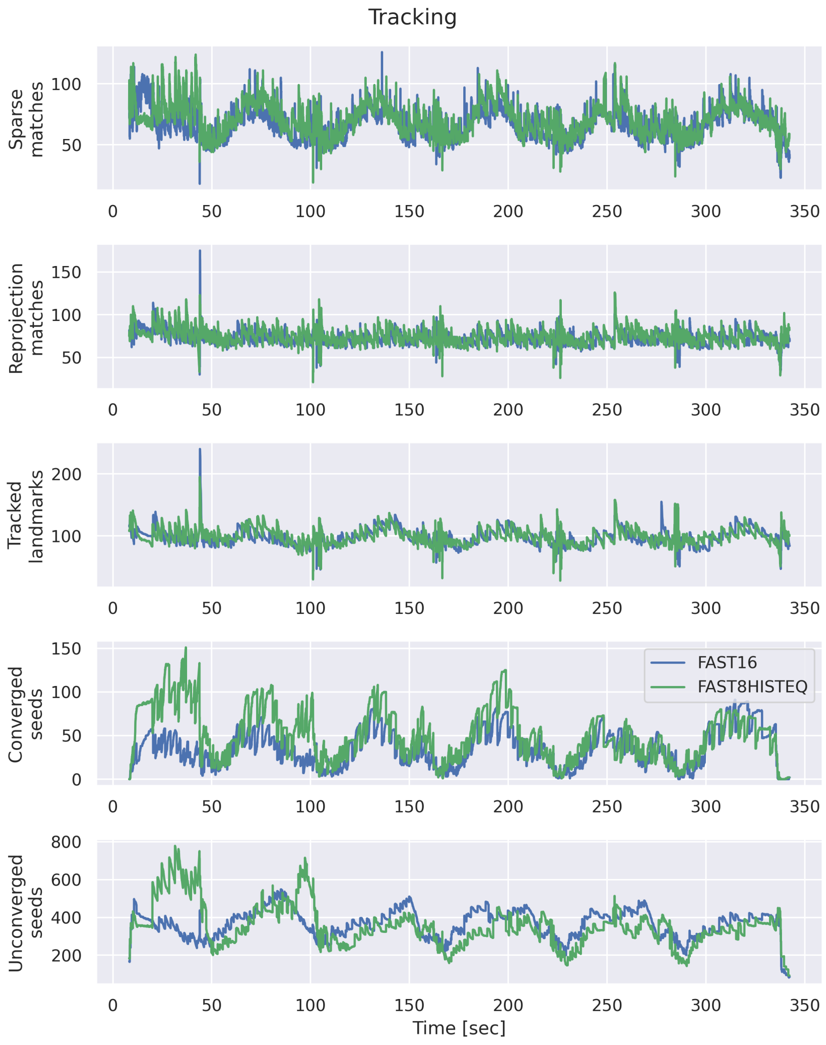
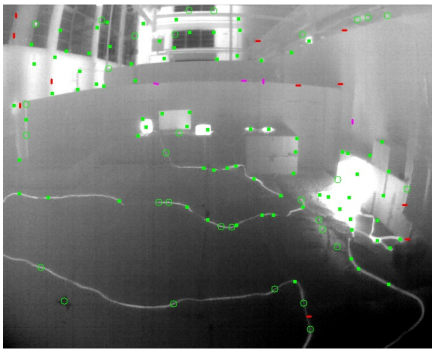


Figure 4.2: Tracking data using only the front-end on the Vicon Arena dataset.


(a) Features detected using FAST16.

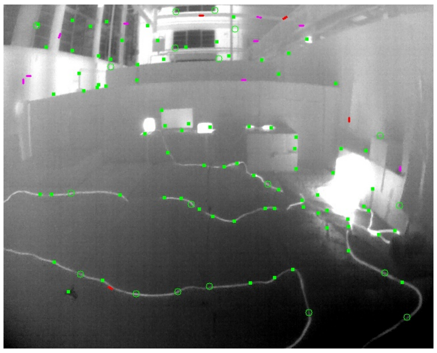
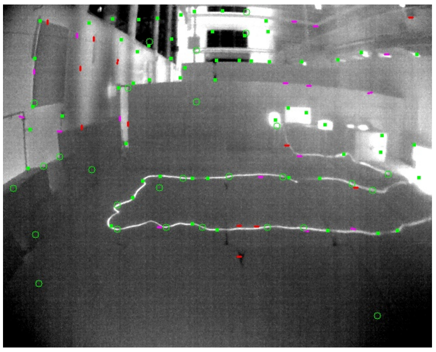

(b) Features detected using FAST8 on histogram equalized image.

Figure 4.3: Comparison of detected features for the right corner in the Vicon Arena dataset. Notice that there are features detected on the fully saturated parts of the space heaters.


(a) FAST16

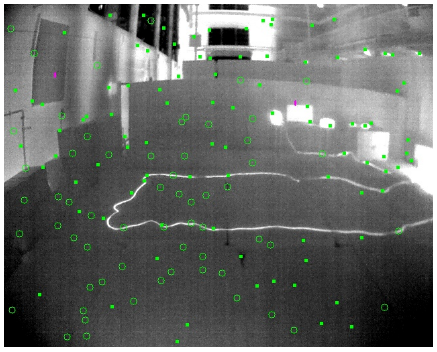

(b) FAST8

Figure 4.4: Tracked features and seeds by FAST16 and FAST8. Green squares are tracked landmarks, green circles are converged seeds not yet upgraded to landmarks, and purple lines are tracked edgelet landmarks. Notice that FAST16 have fewer false-positive seeds.

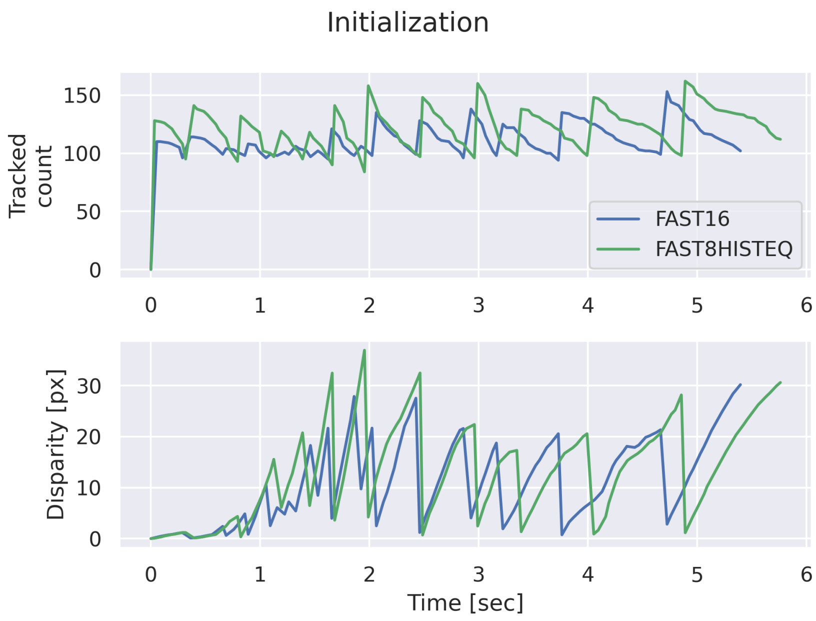


Figure 4.5: Initialization data using only the front-end for the Urban Parking Lot dataset.

4.1.2 Urban Parking Lot

Figure 4.5 shows the initialization data for the Urban Parking Lot dataset. Both front-ends initialized at almost the same time, and the plots look similar to those for the Vicon Arena, although the duration between the peaks is longer. A larger disparity threshold was also used and was doubled from the Vicon Arena. This indicates that the detected points were easier to track. One explanation for this is that there were stronger gradients making the direct alignment easier. As Fig. 4.7 shows, many points were located on and around the hot cars. The strong gradient can therefore come from high temperature differences and varying emissivity in the materials on the car and the surroundings.

Figure 4.6 shows that FAST8 matched overall ever so slightly more features than FAST16 in the sparse model alignment step. In a similar fashion to the Vicon Arena, FAST8 also tracked more seeds than FAST16. Indeed, FAST8 converged more seeds than FAST16, indicating that it tracked them long enough for their depth to converge. However, both front-ends tracked the same number of landmarks, indicating that although FAST8 converged more seeds, the quality of the features was not good enough to keep them as landmarks because they fail reprojection too many times.

Figure 4.7 show that FAST16 detected features in the hot, saturated parts on the car hoods, while FAST8 did not. Despite this, FAST16 did not track more features. This is the same result as observed in the Vicon Arena. On the other hand, FAST8 detected more edgelet features on the beams in the ceiling that FAST16 did not track.

Figure 4.7 also show that the thermally flat ground makes up around half of the image

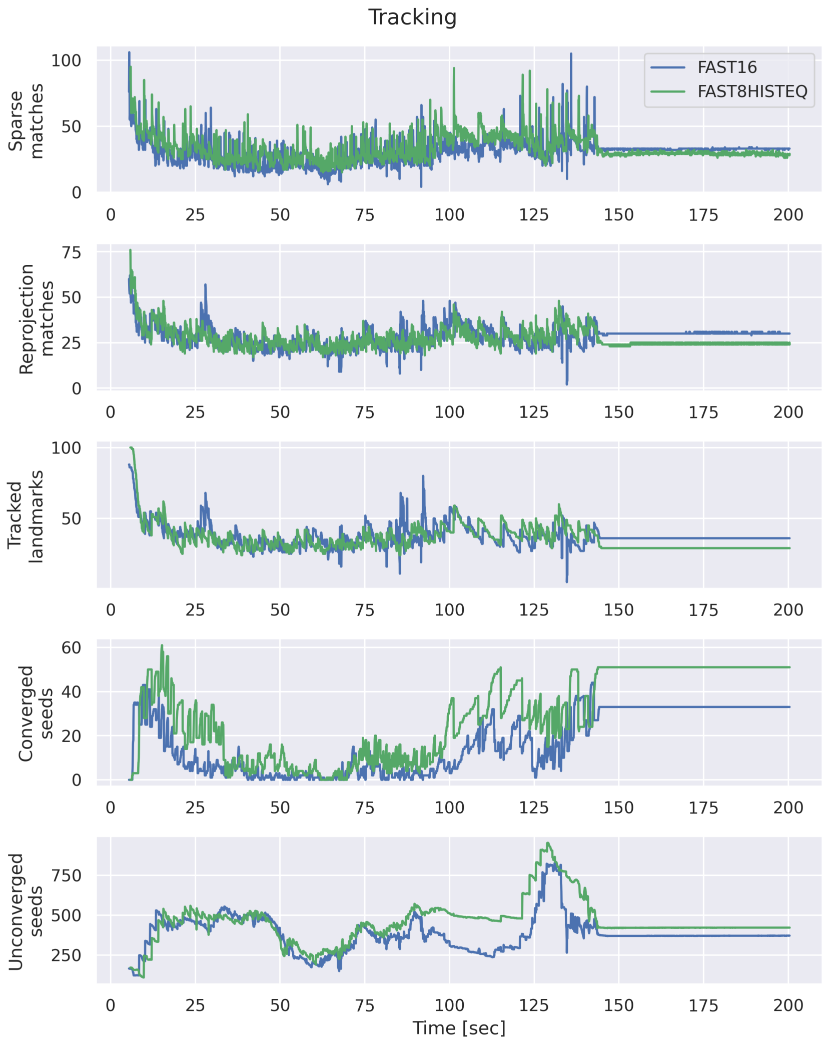


Figure 4.6: Tracking data using only the front-end for the Urban Parking Lot dataset.

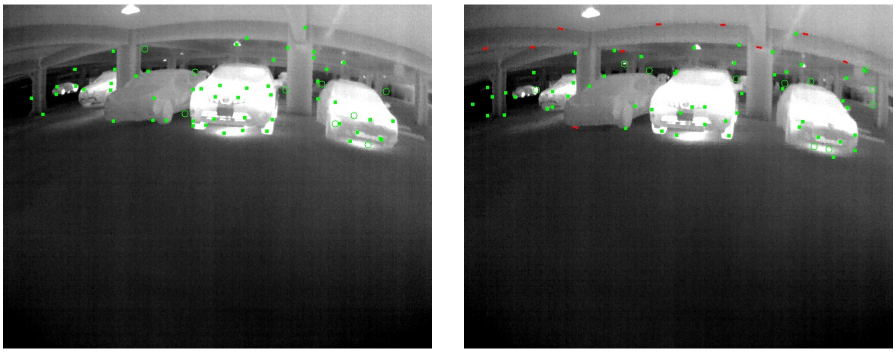

(a) FAST16 **(b)** FAST8

Figure 4.7: Comparison of detected features for the car hoods in the Urban Parking Lot dataset. Notice that there are features detected on the fully saturated parts of the car hoods using FAST16. On the other hand, the red lines are converged edgelet features not yet upgraded to landmarks that are only detected using FAST8.

and that features were only detected in the upper half. Since SVO detects features on a grid with a maximum of one feature per cell, this caused few total tracked features. In an attempt to detect and track more features, the grid cell size was reduced. While this did increase the number of features, it also caused the estimated trajectory to move in the wrong direction. This was likely due to imperfect structure estimation and erroneous feature tracking due to features being on self-similar textures.

4.1.3 Active Gold Mine

Similar initialization results as the two other datasets can be seen in Fig. 4.8. More features were detected, but they were quickly lost because the front-end could not track them. Both front-ends used the same disparity threshold of 20, but FAST8 initialize after FAST16 because it lost too many features just before reaching the threshold.

Figure 4.9 shows that the first 18 seconds FAST8 tracked more features overall. This was despite FAST16 using a smaller FAST threshold. The smaller threshold was needed to maintain enough features. At 18 seconds, the MAV did a quick dip forward that caused both front-ends to lose track of several features. After this point, the two front-ends tracked approximately the same number of features until the thermally flat corner, indicated by the vertical line.

The thermally flat corner has little texture, and both front-ends struggled to find features to track. Figure 4.10 show the features detected by FAST16 and FAST8. It is clear from the frames that FAST8 found many features in the noise. This is also reflected in the unconverged seeds plot in Fig. 4.9, where the count spikes after the black line. However, none of them could converge, since the number of converged seeds went to zero. This confirms that the seeds were noisy measurements and not real features.

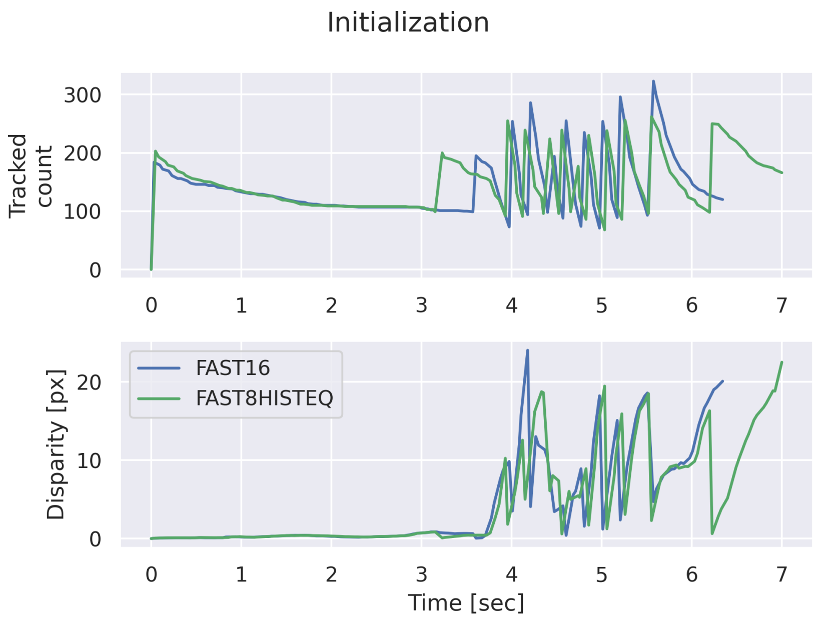


Figure 4.8: Initialization data using only the front-end for the Active Gold Mine dataset.

4.2 STIO using Ceres

4.2.1 Gravity vectors

The gravitation magnitude was estimated per dataset by averaging each component of the linear acceleration measurements from the IMU over a short period of no motion. This excludes the effects of acceleration due to motion and ensures the bias stays constant. The magnitude was then calculated by taking the L2-norm of the averaged measurement vector. The magnitudes for each thermal dataset were estimated to be

$$g_{\text{vicon arena}} = 9.576 \ {}^{\text{m}}\!/_{\text{s}^2} \tag{4.1a}$$
$$g_{\text{urban parking lot}} = 10.164 \ {}^{\text{m}}\!/_{\text{s}^2} \tag{4.1b}$$
$$g_{\text{active gold mine}} = 10.080 \ {}^{\text{m}}\!/_{\text{s}^2} \tag{4.1c}$$

The thermal datasets were recorded in northern Nevada, USA. The magnitude of the gravity vector is not constant on the Earth's surface and depends on several factors like altitude and latitude since the Earth is not a perfect sphere. A United States Geological Survey (USGS) report [28] measured the gravity in Reno, Nevada to be $g = 9.797 \ {}^{\text{m}}\!/_{\text{s}^2}$ when rounded to three decimal places. However, the estimated trajectories quickly diverged on all datasets using this value. It was therefore vital to use the estimated magnitudes in Eq. (4.1) instead.

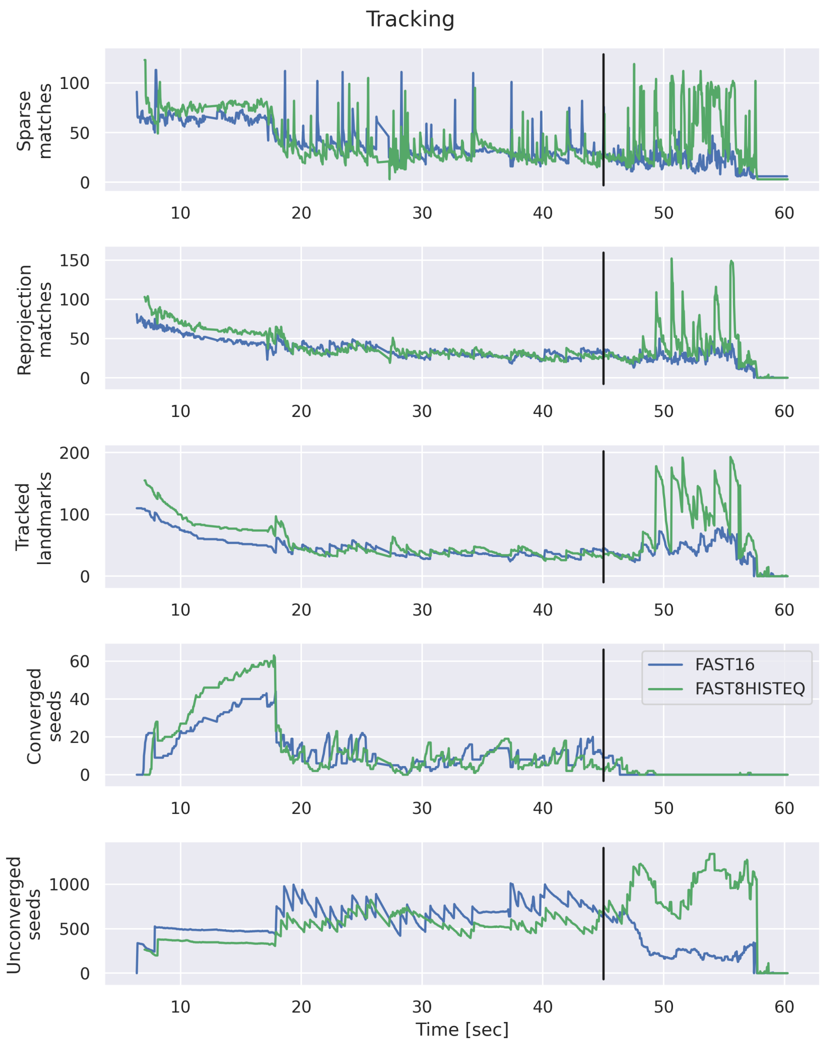
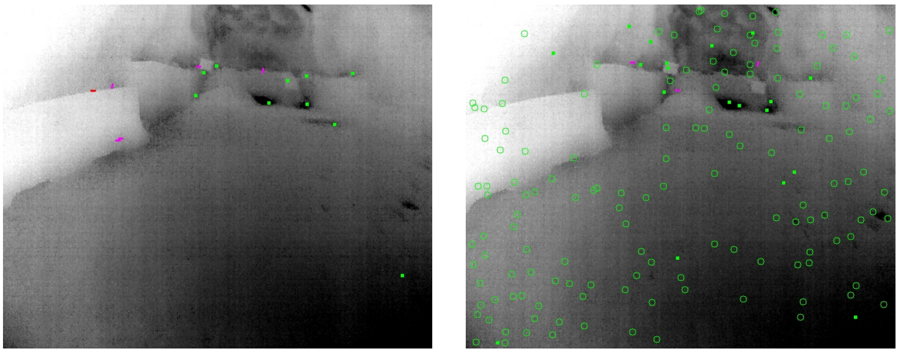


Figure 4.9: Tracking data for Active Gold Mine. The vertical line marks the start of the thermally flat corner showed in Fig. 3.10c.



(a) With FAST16 there are significantly less noise, but still few tracked features.

(b) With FAST8 on a histogram equalized image there is a lot of noise and few tracked features.

Figure 4.10: Comparison of the thermally flat corner of the Active Gold Mine dataset using FAST16 and FAST8 on a histogram equalized image.

The measured gravity magnitudes in Eq. (4.1) differ from the theoretical value likely because of bias build-up in the IMU. The biases also depend on the temperature, which could affect the bias differently in the indoor Vicon Arena and the outdoor Urban Parking Lot and Active Gold Mine.

Using these values, STIO was run on each of the thermal datasets. Table 4.1 summarizes the translational and rotational APE and maximum value of the trajectories from STIO with FAST16 and FAST8, for the 8-bit STIO, and for ROTIO.

4.2.2 Vicon Arena

All methods finished the entire dataset. Row 1-4 in Table 4.1 show that all STIO versions outperformed ROTIO in terms of translational APE (in fairness however, ROTIO was not tuned to the same degree as STIO). Among the 16-bit STIO, FAST16 have slightly smaller translational errors than FAST8. Unexpectedly, the 8-bit STIO have smaller translation APE than both 16-bit versions. On the other hand, the 8-bit version has worse rotation APE. Figure 4.11 show that the 8-bit trajectory coincided with the ground truth better than any of the other methods. It also shows that ROTIO estimated the most incorrect scale. Lastly, Figs. 4.13a and 4.13b show the APE over time. In particular, Fig. 4.13a show that FAST8 slowly drifted away from the ground truth, while FAST16 and the 8-bit STIO stayed consistently close. Moreover, Fig. 4.13c show that all STIO methods have narrower distributions of translation APE compared to ROTIO. On the other hand, Fig. 4.13d shows that rotation APE was overall worst for the 8-bit STIO and comparably equal for the 16-bit versions. However, FAST16 have the largest rotation APE. This is due to the loss of track at the very end when the MAV descended to land.

Figure 4.14 shows the estimated biases for each method. No ground truth for the biases is available, but they can be compared against each other. The gyroscope biases all quickly converged to approximately the same value. On the other hand, the accelerometer biases varied more. The spike at the beginning for FAST16 is because the FAST16 front-end had

Table 4.1: Summary of translational and rotational APE for each TIO method on each dataset. The duration column informs how long each method kept track before diverging. The duration in parenthesis under the dataset name is the total duration of the dataset.

Dataset	**Method**	**Duration [s]**	**APE translation [m]**		**APE rotation [°]**	
			RMSE	**Max**	**RMSE**	**Max**
Vicon Arena (342.35 s)	FAST16	342.35	0.4100	0.6539	0.6778	4.4777
	FAST8 hist.eq.	342.35	0.6276	1.2164	0.6465	1.8008
	ROTIO	342.35	1.0284	2.4714	1.0469	2.1976
	8-bit hist.eq.	342.35	0.2393	0.5436	1.0946	2.4507
Urban Parking Lot (200.42 s)	FAST16	200.42	2.4872	5.1533	2.8508	5.6659
	FAST8 hist.eq.	200.42	2.9897	5.8081	3.1768	6.9486
	ROTIO	200.42	1.5451	2.9642	0.7813	1.9943
	8-bit hist.eq.	200.42	3.5713	4.9870	7.6961	11.953
Active Gold Mine (159.62 s)	FAST16	110.81	10.019	16.664	14.211	22.564
	FAST8 hist.eq.	137.61	8.1502	14.843	13.761	18.941
	ROTIO	159.62	1.2693	1.9905	2.6760	6.9045
	8-bit hist.eq.	159.62	2.4736	5.8050	4.9044	9.4295

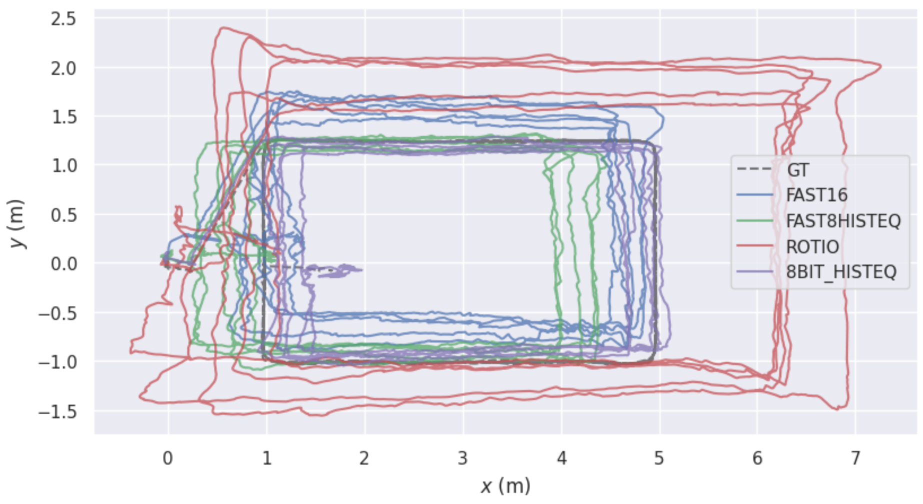
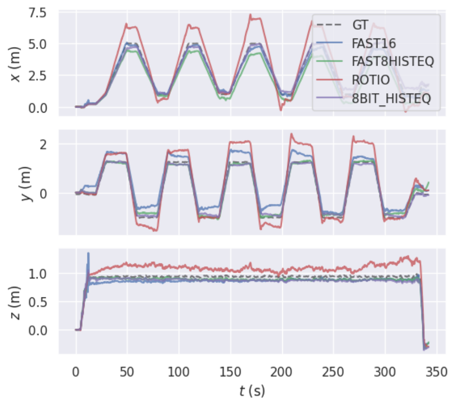
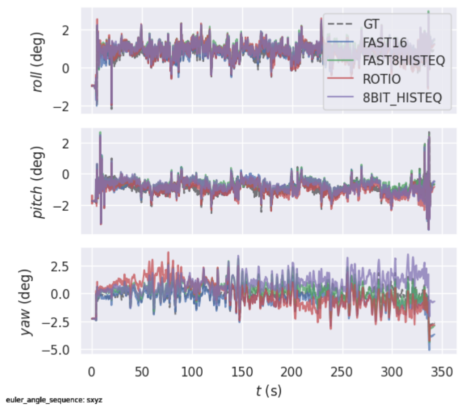


Figure 4.11: Top-down view of the Vicon Arena estimates. The dashed line is the ground truth, the blue line is FAST16, green line is FAST8, red line is ROTIO, and magenta line is the 8-bit STIO. This color scheme will be used in all subsequent figures.



(a) XYZ



(b) Roll, pitch, yaw

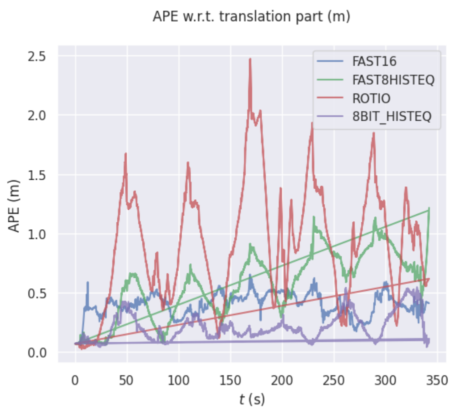
Figure 4.12: Each component of the pose estimates for the Vicon Arena. The data is origin aligned so all trajectories evolve from the same point.

to reinitialize shortly after starting.

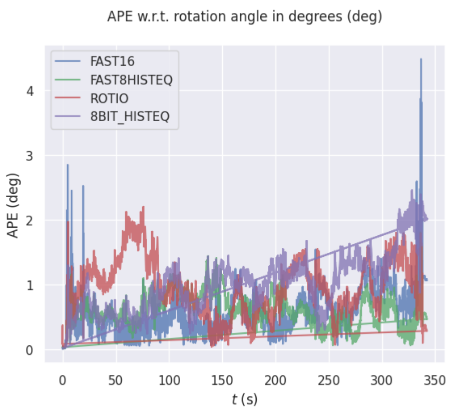
Figure 4.15a show the number of tracked features by STIO during initialization. All methods initialized at approximately the same time, but FAST16 immediately lost track and had to reinitialize. This caused the height estimate to overshoot, which can be seen in the z plot in Fig. 4.12a. Despite this overshoot, FAST16 was able to correct the height and stayed close to the ground truth for the rest of the dataset. Both 16-bit methods also initialized quicker than when the front-end was run alone. This shows the effectiveness of having rotation priors from the IMU to help the matching.

As seen in Fig. 4.15b, both 16-bit STIO front-ends tracked the same number of features and landmarks. This was despite the fact that FAST8 used a larger grid cell size and a larger FAST threshold than FAST16. On the other hand, the 8-bit STIO tracked more features than both 16-bit methods. This was using exactly the same parameters as the 16-bit STIO with FAST8. However, the number of tracked landmarks is noticeably noisier. Despite tracking many features, it also tracked significantly more false-positive features. The 8-bit version also detected and tracked more edgelet features than both 16-bit versions. Figure 4.16 shows a comparison of the features detected by FAST8 and the 8-bit version. Notice that on the heating cable on the ground, there are more edgelet features. Figure 4.16 also shows that there were more features overall with the 8-bit version.

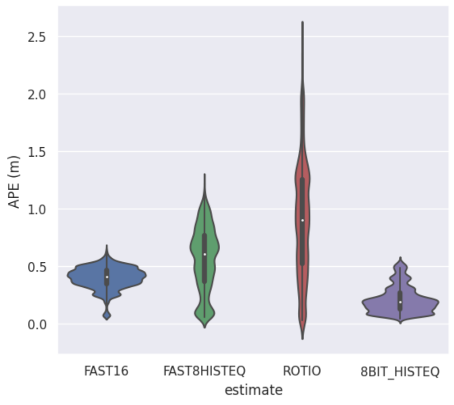
Lastly, the illumination gain and offset estimation was essential to be able to retain track during data interruption when the thermal camera performed FFC. The front-end would consistently fail without using gain and offset estimation. Additionally, having the IMU also helped to match the features after the FFC. This is opposed to the front-end only, where the estimate would often incorrectly estimate the translation after the FFC.



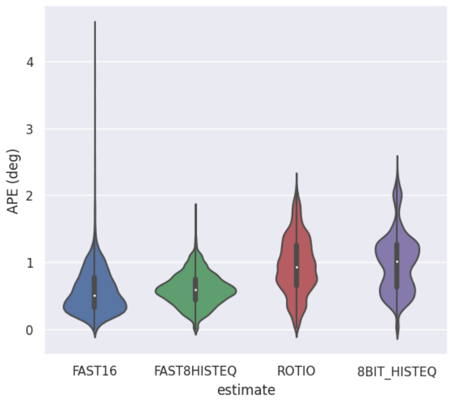
(a) Translational APE



(b) Rotational APE

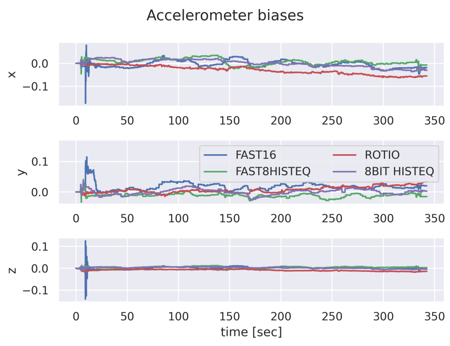


(c) Translational APE distribution

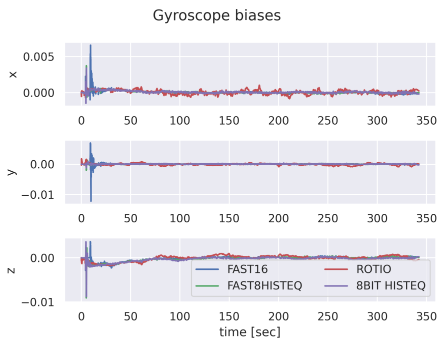


(d) Rotational APE distribution

Figure 4.13: Plots showing the translational and rotational APE over time and their distributions. In the distribution plots (c) and (d), the white dot represent the median value. Note that the straight lines in (a) and (b) are graphical glitches and not part of the data.

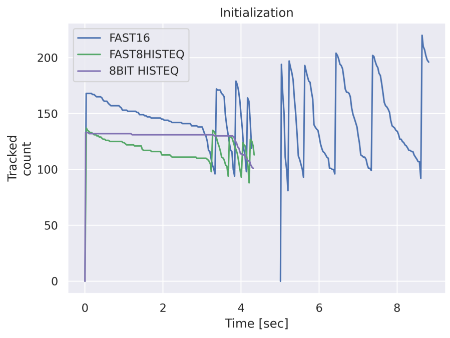


(a) Accelerometer biases

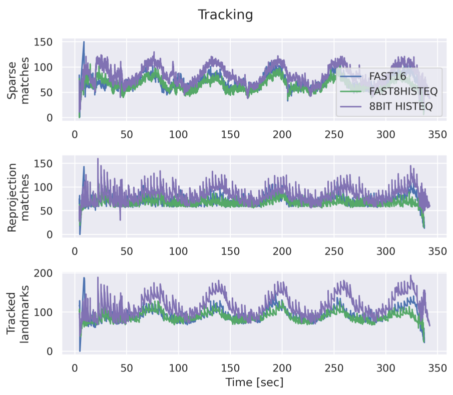


(b) Gyroscope biases

Figure 4.14: Biases for Vicon Arena. No ground truth is available.

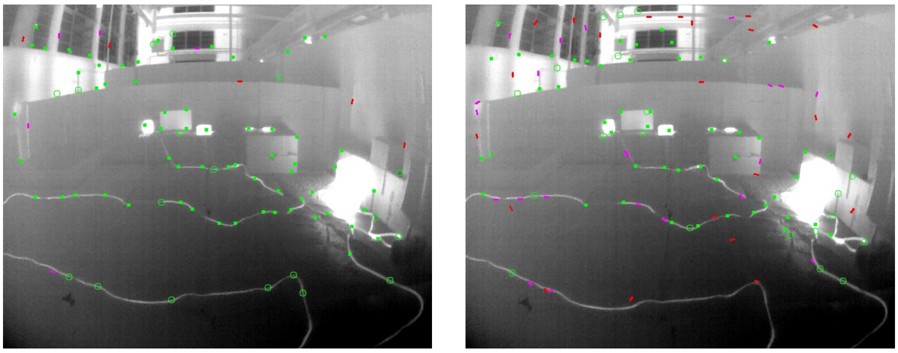


(a) Number of tracked features during initialization.



(b) Tracking data after initialization.

Figure 4.15: Initialization and tracking data for the STIO versions for the Vicon Arena.



(a) FAST8 **(b)** 8-bit STIO

Figure 4.16: Comparison of detected and tracked features for 16-bit using FAST8 and 8-bit STIO. Notice that both detected and tracked more or less the same corner features, but that the 8-bit STIO detected and tracked more edgelet features.

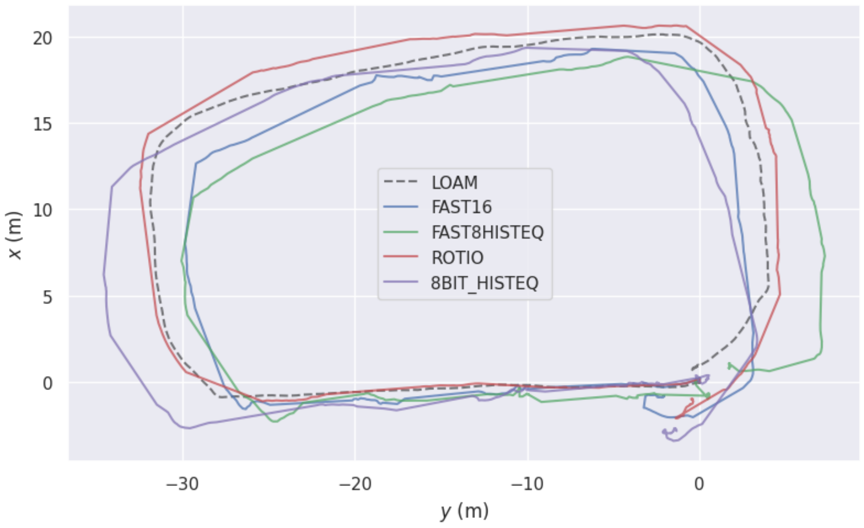
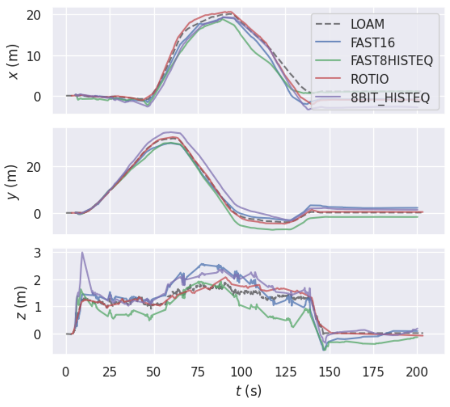
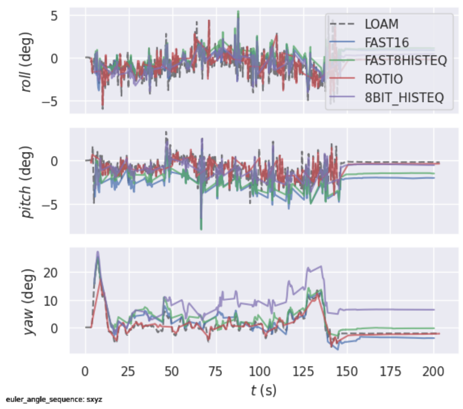


Figure 4.17: Top-down view of the Urban Parking Lot estimates. The ROTIO estimate clearly followed the ground truth better than any STIO estimate.



(a) XYZ



(b) Roll, pitch, yaw

Figure 4.18: Each component of the pose estimates for the Urban Parking Lot. The data is origin aligned so all trajectories evolve from the same point.

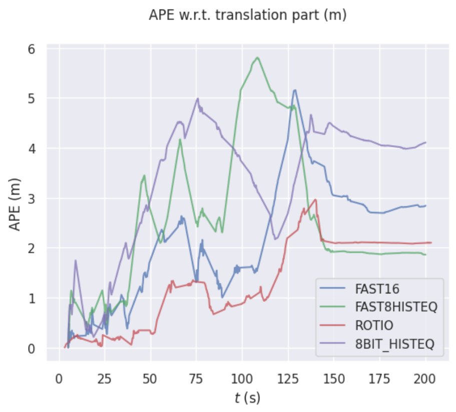
4.2.3 Urban Parking Lot

All methods finished the entire dataset. Row 5-8 in Table 4.1 shows that contrary to the Vicon Arena, ROTIO have smaller APE on all counts than all STIO methods for this dataset. This is also seen in Figs. 4.19c and 4.19d where ROTIO have the narrowest distributions of APE for both translation and rotation, and with the smallest median. Figures 4.17 and 4.18a also show that both 16-bit STIO methods estimated a too short trajectory, while the 8-bit STIO overshot slightly in the y-direction. It also show that the 8-bit version wrongly estimated the height. It overshot drastically at the beginning, but converged to the correct height after a while. However, at around 50 seconds, the height started deviating again. This coincides with the thermally flat area seen in Fig. 3.9c.

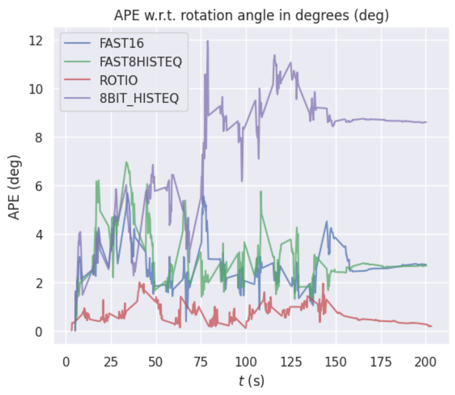
Figure 4.20 show the biases for all methods. The gyroscope biases quickly converged for the x- and y-components, while the z-component had more variation. On the other hand, the accelerometer bias x- and y-components seem more random for STIO than for ROTIO. The z-component converged quicker but to a different value than ROTIO.

Figure 4.21a shows the number of features tracked during initialization. All STIO methods initialized at the same time and were faster than when the front-end was run without IMU. Both 16-bit methods triangulated approximately the same number of features, and the 8-bit version triangulated almost double the number of points. However, all methods lost almost all features right after they had initialized. This is seen in Fig. 4.21b with the sudden fall in number of features as the beginning. The points were removed by the outlier rejection. This is likely due to erroneous translation estimation in the beginning. This can best be seen from Fig. 4.19a where all STIO algorithms had large translational APE at the start.

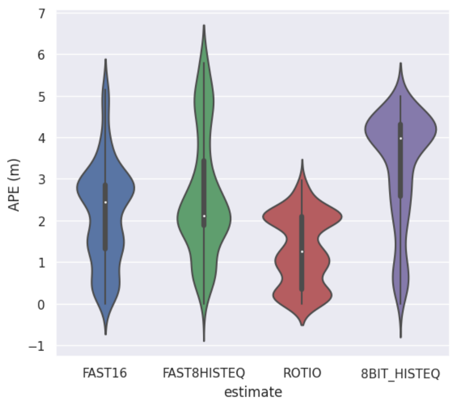
The 16-bit STIO front-ends were generally unable to track many features as seen in Fig. 4.21b. However, compared to when the front-end was run alone, the grid cell size could be reduced significantly more, which allowed for more detected and tracked features.



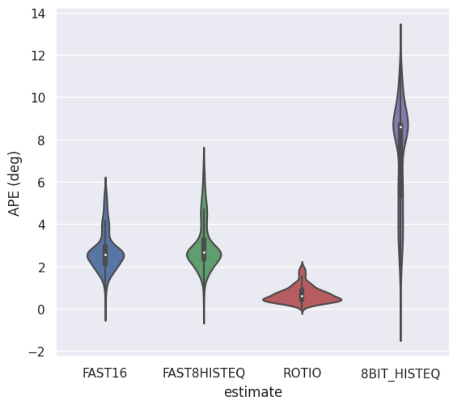
(a) Translational APE



(b) Rotational APE

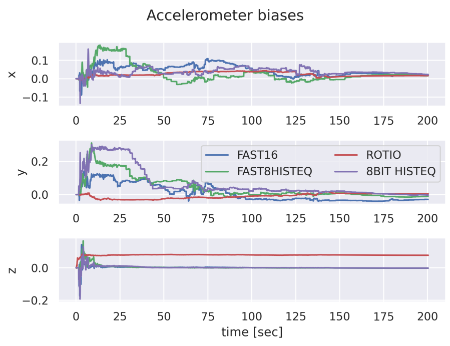


(c) Translational APE distribution

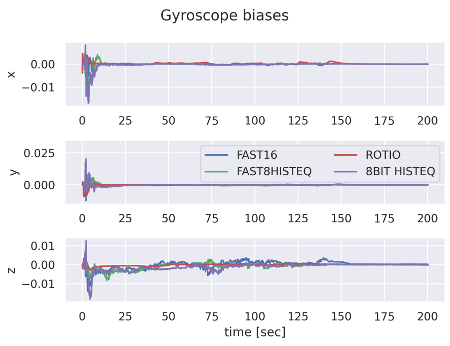


(d) Rotational APE distribution

Figure 4.19: Plots showing the translational and rotational APE over time and their distributions. Notice that the ROTIO distributions are narrower and have smaller median than any of the STIO methods.

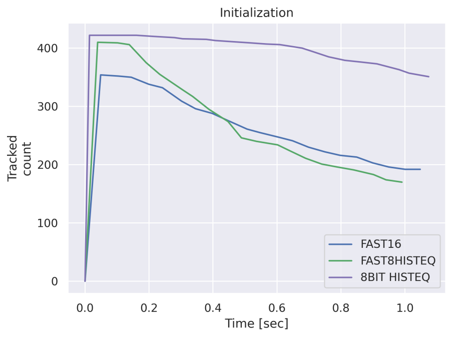


(a) Accelerometer biases

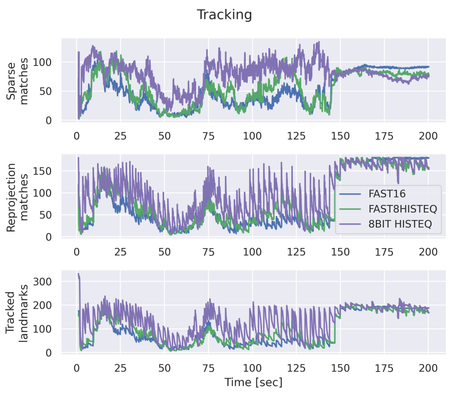


(b) Gyroscope biases

Figure 4.20: Biases for Urban Parking Lot. No ground truth available.



(a) Number of tracked features during initialization.



(b) Tracking data after initialization.

Figure 4.21: Initialization and tracking data for the STIO versions for the Urban Parking Lot.

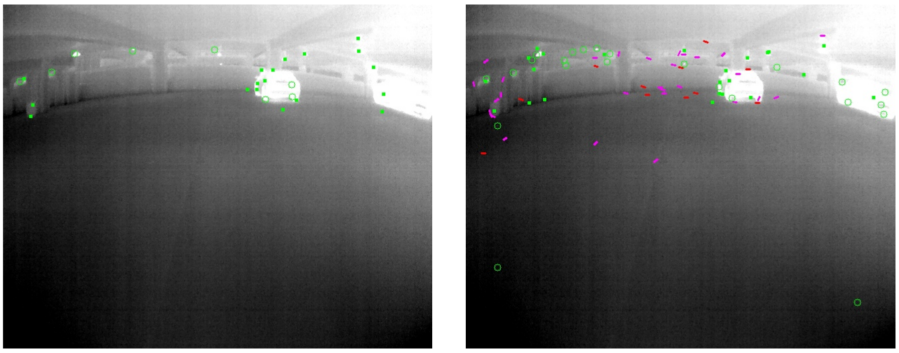

(a) FAST8 **(b)** 8-bit STIO

Figure 4.22: Comparison of detected and tracked features by the 16-bit using FAST8 and the 8-bit STIO. Note that the 8-bit STIO detects many more edgelet features, but also some false-positive features.

This shows the improvements motion priors from an IMU had for both matching and correct trajectory estimation. Furthermore, FAST8 detected and tracked more features than FAST16. Similar to the Vicon Arena, FAST16 required a smaller threshold than FAST8 to have reliable estimates. On the other hand, the 8-bit version tracked more features than both 16-bit versions using exactly the same front-end parameters as 16-bit with FAST8. However, similarly to the Vicon Arena, the 8-bit front-end tracked more false positive features. This is evident from the noisy plots, which show that features and landmarks were initialized and tracked but were quickly removed again.

Between around 45 to 65 seconds in Fig. 4.21b, the number of tracked features dropped drastically. This was the thermally flat area seen in Fig. 3.9c. Neither of the 16-bit STIO front-ends was able to detect many features in this area, and in both cases, the number of tracked features fell below 10. In this time frame, the STIO estimates relied heavily on dead reckoning from the IMU. It still maintained a couple of tracked features, but the threshold for minimum required tracked features had to be turned down to 5 so the front-end would not lose track in this area. However, while it could not track many features, tracking just a couple was better than none at all. On the other hand, the 8-bit STIO version tracked more features in this time frame. However, the features were often quickly lost or removed. Figure 4.22 shows the detected features in one frame in the thermally flat area. The 8-bit version detected more features, especially edgelet features. On the other hand, FAST8 only detected a few corner features. Figure 4.22b also show that the 8-bit version did indeed detect and track more false-positive features.

4.2.4 Active Gold Mine

Both 16-bit STIO methods were not able to complete the entire dataset before failing, while ROTIO and the 8-bit STIO were able to complete the entire dataset. FAST8 diverged later than FAST16. Figure 4.23 shows a top-down view of the estimated trajectory. The gaps are the parts of the trajectory where the front-end completely lost track in the

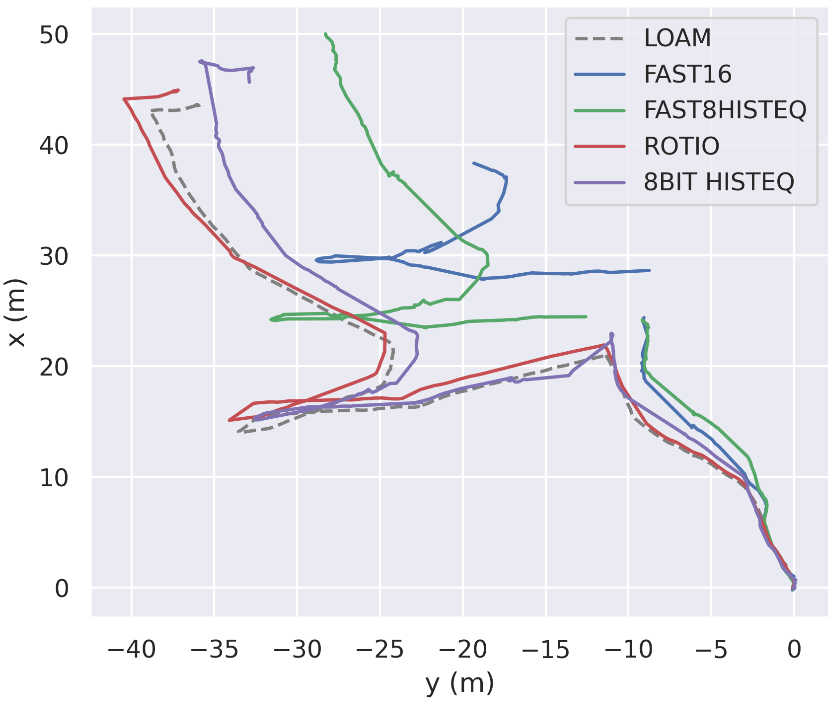
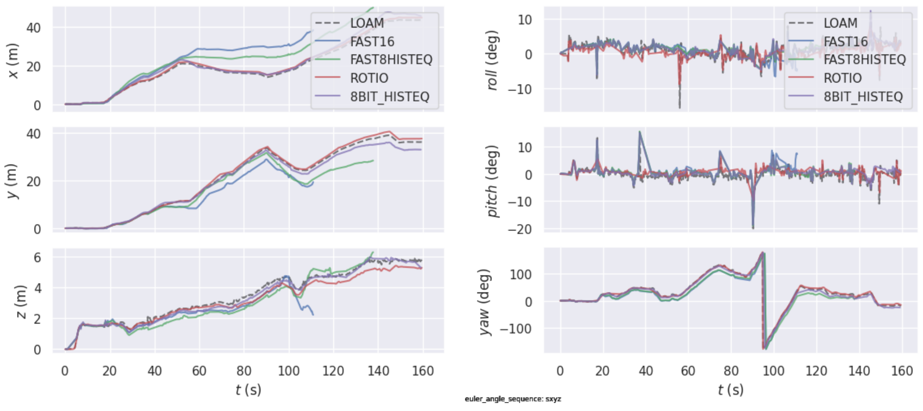


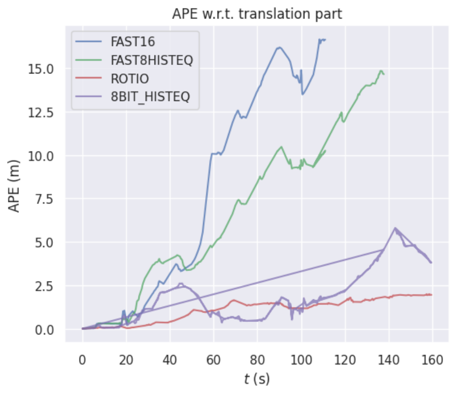
Figure 4.23: Top-down view of the Active Gold Mine estimates. The gaps are when the front-end lost track in the thermally flat corner. Note that the x-axis is vertical and the y-axis is horizontal.



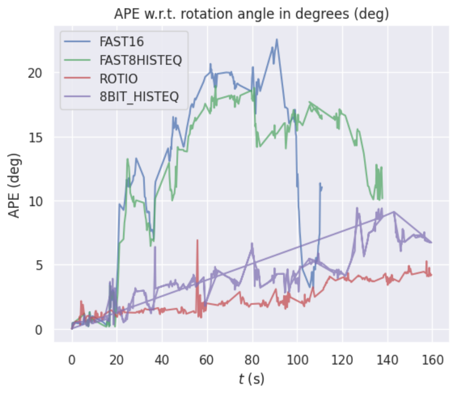
(a) XYZ

(b) Roll, pitch, yaw

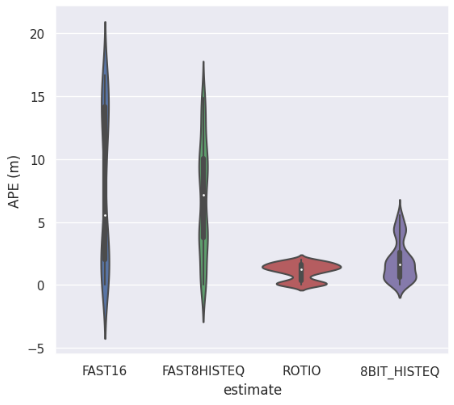
Figure 4.24: Each component of the pose estimates for the Active Gold Mine. The data is origin aligned so all trajectories evolve from the same point.



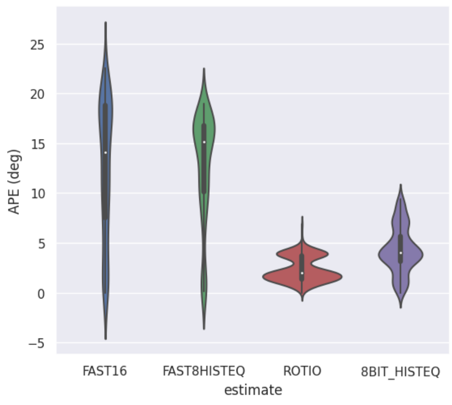
(a) Translational APE



(b) Rotational APE

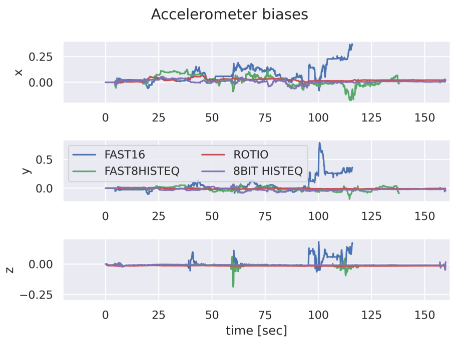


(c) Translational APE distribution

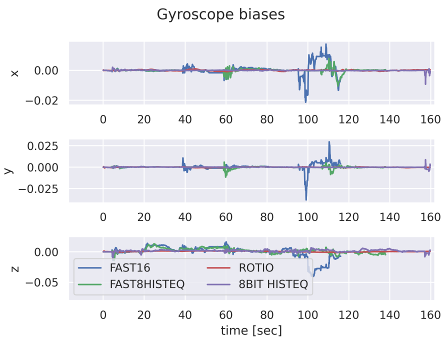


(d) Rotational APE distribution

Figure 4.25: Plots showing the translational and rotational APE over time and their distributions. The 16-bit STIO distributions are stretch out and multi-modal because they were close to the ground truth for a period in the beginning, and then diverged. Note that the straight lines in (a) and (b) are graphical glitches and not part of the data.

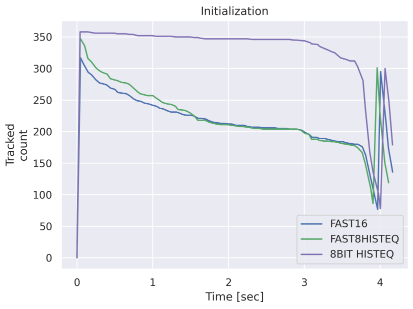


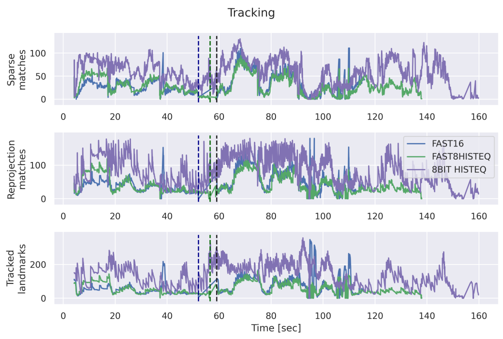
(**a**) Accelerometer biases



(**b**) Gyroscope biases

Figure 4.26: Biases for Active Gold Mine. No ground truth available.





(**a**) Number of tracked features during initialization.

(**b**) Tracking data after initialization. The vertical lines indicate the time frame in which the 16-bit STIO methods lost track in the thermally flat corner. The blue line indicate when FAST16 lost track, the green line indicate when FAST8 lost track, and the black line indicate when both regained track.

Figure 4.27: Initialization and tracking data for the STIO versions for the Active Gold Mine.

thermally flat corner and tried to reinitialize. This will be looked at in more detail later. Row 9-12 in Table 4.1 show the quantitative results. As expected, the 16-bit have large values for the metrics because they deviated from the ground truth early and were therefore always far from the ground truth.

The 16-bit STIO estimates deviated quickly in the beginning. This is clear from Figs. 4.25a and 4.25b where the errors quickly increased after 20 seconds. Figure 4.23 also show the STIO trajectories diverged from the path shortly after start. One explanation for the divergence is that the MAV did a quick pitching motion to move forward at around 18 seconds. This can be seen in Fig. 4.24b. This caused both 16-bit STIO methods to lose track of many features. This can also be observed in Fig. 4.27b, which show the number of tracked features. A similar motion happened at around 37 seconds. However, at that time, FAST16 lost track of all features and had to reinitialize. This is seen in the spike in features in Fig. 4.27b that are quickly lost again. On the other hand, FAST8 barely managed to track enough features to not lose track.

Conversely, the 8-bit STIO has comparable results to ROTIO. The 8-bit's position estimate closely followed the ground truth until around 100 seconds, when it started to deviate. It deviated for the same reason as the 16-bit, i.e., the front-end lost track of all features and had to reinitialize. Despite this, the 8-bit version was able to reinitialize and continue to give decent position estimates.

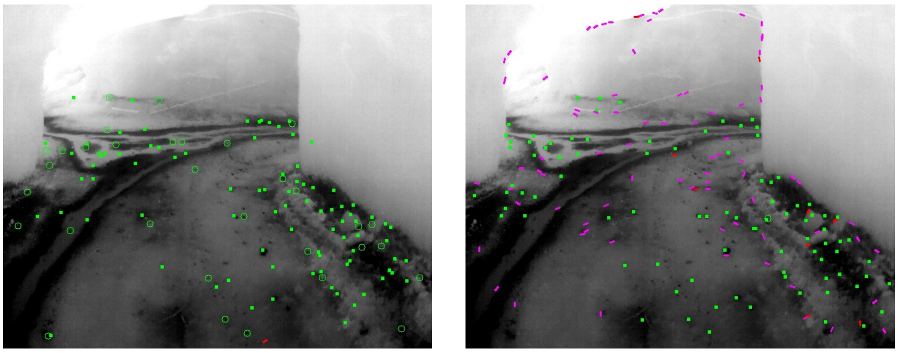
Figure 4.28 show the features detected and tracked in the start area by the 16-bit STIO using FAST8 and the 8-bit STIO. It is clear that both detected the same corner features on the ground, but the 8-bit version detected and tracked many more edgelet features.

The dashed blue line in Fig. 4.27b marks when FAST16 lost track in the thermally flat corner. Likewise, the dashed green line marks when FAST8 lost track. The black dashed line marks when both regained track. FAST16 lost track for almost 7 seconds, while FAST8 lost track for around 3 seconds. As seen in Fig. 4.23, both 16-bit STIO versions lost track at the same location, but FAST8 estimated using the IMU movement in the correct direction, while FAST16 wrongly continued forward. Figure 4.26a shows a discrepancy in the x-component of FAST16's accelerometer bias compared to the other estimates in the time frame of the thermally flat corner. Since the bias is only updated with additional measurements such as tracked landmarks, the unchanging positive bias was integrated and caused drift in the x-direction. This is exactly what is observed in Fig. 4.23 (note that the x-axis is the vertical axis). While FAST8 also lost track, it was for a shorter duration. Though it did not track many landmarks in the thermally flat corner, it seems like the few it did track helped stabilize the estimate enough to have good motion estimation when the front-end lost track.

In the thermally flat corner, the 8-bit STIO version detected strikingly many false-positive and noisy features. While Fig. 4.27b show that the front-end tracked many landmarks, the majority of them were false positives in that they are stationary in the frame and do not move with the scene. However, despite this, the 8-bit version successfully kept track through the entire thermally flat corner without deviating much from the ground truth.

FAST16 lost track for short periods multiple times during the dataset. This can be seen by the spikes in tracked features in Fig. 4.27b. Conversely, FAST8 was able to keep track in more areas where FAST16 lost track.

Nevertheless, in the end, both 16-bit STIO methods diverged because they lost track



(a) FAST8 **(b)** 8-bit STIO

Figure 4.28: Comparison of detected features by the 16-bit using FAST8 and the 8-bit STIO in the starting area of the Active Gold Mine. Note that they both detect the same corner features, but the 8-bit detect more edgelet features.

Table 4.2: Overview of the RMSE of the biases against the ground truth in the EuRoC Vicon and Leica datasets. Row 3 shows the RMSE values after the initialization motion was done in the full Leica dataset. Row 5 shows the RMSE values after the biases had converged for the shortened Leica dataset.

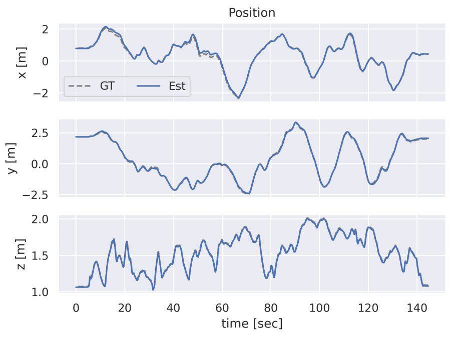
Dataset	**RMSE b_a [$^{\mathrm{m}}/_{\mathrm{s}^2}$]**			**RMSE b_g [$^{\mathrm{rad}}/_{\mathrm{s}}$]**		
	x	**y**	**z**	**x**	**y**	**z**
EuRoC Vicon	0.0224	0.4146	0.0440	0.0003	0.0006	0.0020
EuRoC Leica, full	0.3486	0.2084	0.8314	0.0420	0.0238	0.0239
EuRoC Leica, full, excl. first 25 sec.	0.0730	0.1091	0.1976	0.0116	0.0009	0.0048
EuRoC Leica, short	0.4536	0.5434	2.1011	0.2500	0.1648	0.1598
EuRoC Leica, short, excl. first 25 sec.	0.0837	0.1503	0.2288	0.0058	0.0016	0.0015

and could not recover. During reinitialization, the dead reckoning had diverged the pose estimate too far, so it quickly lost track again due to bad motion priors after the front-end had reinitialized. This caused a self-amplifying effect, and the estimate diverged quickly.

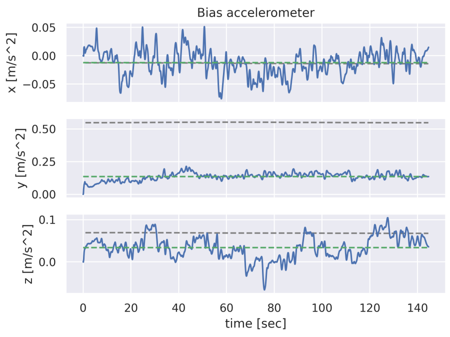
4.3 IMU and external pose initialization

4.3.1 Full pose prior from Vicon motion capture

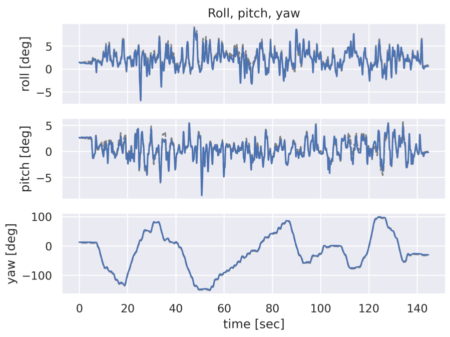
Figure 4.29 show the position, orientation, and biases against ground truth when the full pose prior was available. The position and orientation followed the ground truth almost perfectly, with only small deviations. The gyroscope biases also converged quickly to the ground truth. The x-component is noisier than the y- and z-component but moved around



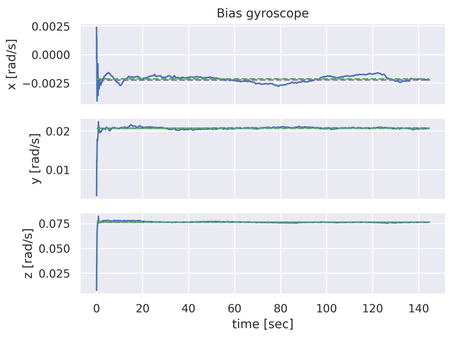
(a) Position



(b) Accelerometer bias

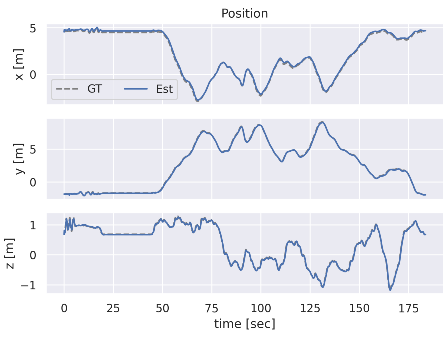


(c) Roll, pitch, yaw

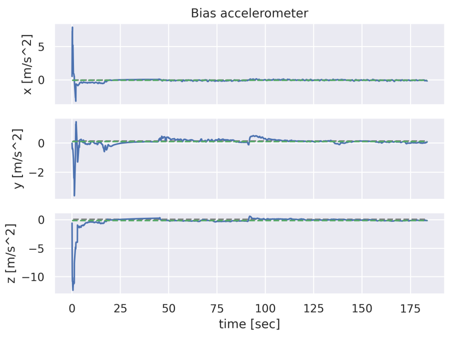


(d) Gyroscope bias

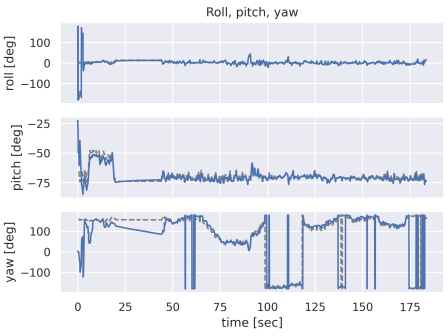
Figure 4.29: States from IMU and external full pose prior on the EuRoC Vicon room dataset. The dashed lines are the ground truth and the blue lines are the estimates. The green dashed line is the mean bias for each component.



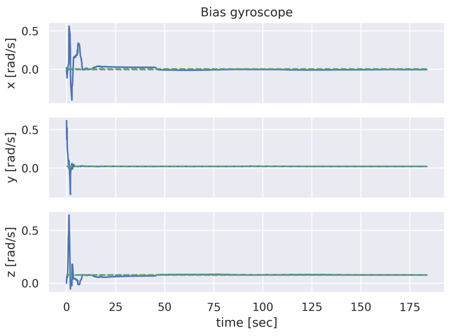
(a) Position



(b) Accelerometer bias. Note the vertical axis scale.



(c) Roll, pitch, yaw



(d) Gyroscope bias

Figure 4.30: States from IMU and external position prior on the full EuRoC Leica dataset.

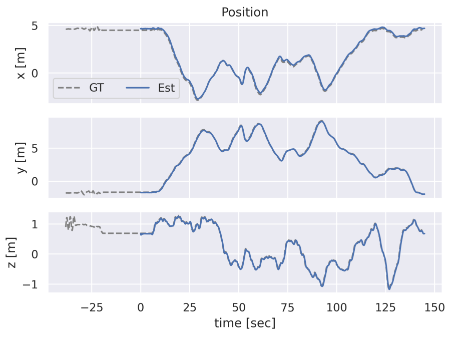
the ground truth. Notice also that the scale on the vertical axis is small. The convergence of the gyroscope biases is also evident from the small RMSE in Table 4.2.

On the other hand, the accelerometer bias never converged to the ground truth. The x-component fluctuated around the ground truth and has a mean value equal to the ground truth. The same is not true for the y- and z-components, which had a large difference between their mean and the ground truth. This is also reflected in the RMSE in Table 4.2. The y-component did converge, albeit with small fluctuations, but to a value different from the ground truth.

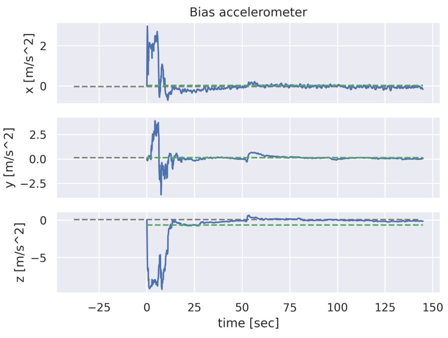
4.3.2 Position only partial prior from Leica positioning system

Figures 4.30 and 4.31 show the position, orientation, and biases against ground truth when only a position prior was available. Figure 4.30 is the results from the whole dataset which include the initialization motion, while Fig. 4.31 is the results from a subset of the data which start 40 seconds into the dataset, skipping the initialization motion.

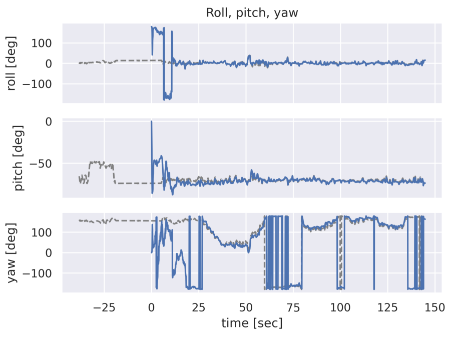
It seems from Fig. 4.30 that the initialization motion quickly converged the biases to close to ground truth. However, the scale on the vertical axes in Fig. 4.30b is too large to see the variations in the bias. Figure 4.32a shows a zoomed in look at the accelerometer



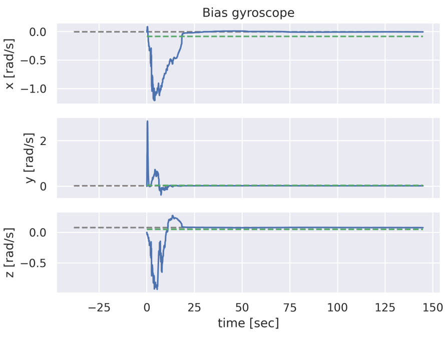
(a) Position



(b) Accelerometer bias. Note the vertical axis scale.

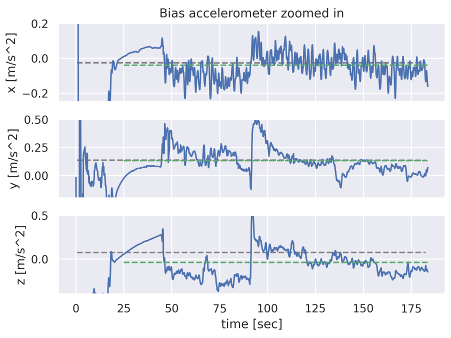


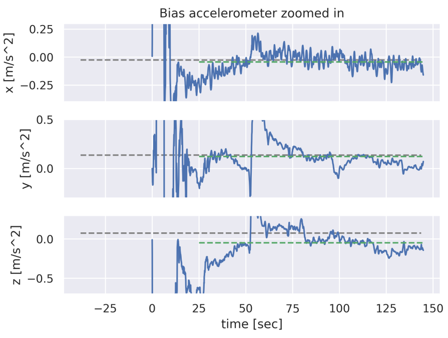
(c) Roll, pitch, yaw



(d) Gyroscope bias

Figure 4.31: States from IMU and external position prior on the shortened EuRoC Leica dataset. The shortened dataset excludes the initialization motion at the beginning and start the dataset 40 seconds in.





(a) Accelerometer bias zoomed in for the full dataset. **(b)** Accelerometer bias zoomed in for the short dataset.

Figure 4.32: Zoomed in look at the accelerometer bias for the Leica dataset. The green lines show the mean excluding the first 25 seconds.

bias, which clearly shows more fluctuations, although the means were close to the ground truth. This is also seen in the RMSE in Table 4.2 (line 2). One reason for the large RMSE were the spikes in the beginning. As line 3 in Table 4.2 show, the RMSE is smaller when excluding the initial spikes from calculations.

The roll and pitch were also accurately estimated, while the yaw had a larger error. Notice also in Fig. 4.30c that from around 25 seconds to 50 seconds, the yaw drifted while the MAV is lying still. This resulted from the gyroscope x-component being slightly different from the ground truth. It essentially integrated a constant, albeit tiny, fake angular velocity, leading to the ramp in yaw. The reason the x-component affected the yaw and not the z-component is because of the unusual orientation of the IMU where the x-axis is pointing upwards [6]. Another reason, as Fig. 3.12 in Weiss [54, p. 112] shows, is that yaw and x- and y-velocities are jointly observable (line 3), meaning motion in the horizontal plane is required to correctly estimate the yaw when only position measurement is available.

In Fig. 4.31 the biases appear to also converge after around 25 seconds of motion. This is true for the gyroscope biases, as line 5 in Table 4.2 show, but as seen in Fig. 4.32b the accelerometer biases were more noisy. Table 4.2 also show that the RMSE is large. However, when calculating RMSE for only the period after the biases seemingly have converged, the RMSE is smaller.

4.4 GTSAM back-end partial results

The author was not able to make the GTSAM back-end work. Significant effort was put into debugging and trying to figure out the root issue. However, in the end, no definitive conclusion was reached on the issue. Therefore, this section will be a presentation of what was done and the observations made during debugging. Since the estimate quickly diverged after the front-end initialized, only qualitative results will be presented, and the behavior will be compared to the working Ceres back-end. During development and testing, the EuRoC Vicon dataset was used, so that will be the basis for the following results and discussion.

First, the current state of the back-end is described. The initialization using IMU preintegration and external pose priors was run for a predetermined duration, after which the front-end initialized and triangulated an initial map. The initial map was inserted into the back-end factor graph and optimized, and the results were used to update the map. This worked for a couple of frames, but the estimated pose diverged fast. In almost all cases, the iSAM optimizer eventually crashed with an indeterminant linear system exception.

The duration of the initialization did not affect how quickly the estimate diverged. As seen in Section 4.3.1, the gyroscope biases quickly converged to the ground truth while the accelerometer biases were noisy throughout the entire dataset. Different durations ranging from 3 seconds to 30 seconds were tried, but the same diverging result was observed in all cases.

The external pose prior was added to each keyframe, even after the front-end had initialized. This will ensure that the pose is known with low uncertainty. However, the pose estimate still diverged within a second or two of the front-end being initialized. This can indicate that the landmarks are being erroneously estimated.

Outlier rejection was found to be essential for accurate estimating using the Ceres back-end. However, when used within the GTSAM back-end, strikingly many landmarks were removed as outliers. The outlier threshold was also doubled but to no avail. A landmark is marked as an outlier if the projected landmark and the corresponding feature are more than a threshold pixels apart. Therefore, landmarks that were not outliers should move mainly along the ray from the optical center passing through the measured pixel location. I.e., it should mainly change its depth in the camera frame it was observed in. However, the fact that so many landmarks were removed indicates that they move too much in the image plane and not only in depth. Since the Ceres back-end has fewer outliers, it is logical to conclude that this is due to inaccurate estimation by the GTSAM back-end.

Moreover, the reprojection step in the front-end was frequently unable to reproject many landmarks into the newest frame because of the optimization to align the features, Eq. (3.7), did not converge. This is a sign that the initial guess of feature location was too far away, so the optimization cannot find patches that overlap well. The poor initial guess was either due to bad motion estimation or a bad estimate of the 3D landmark position. From the initialization and results seen in Section 4.3, the IMU preintegration is known to work well with the external prior and, over a short period, should be able to predict the motion of the system accurately. This gives reason to expect that it was the landmark positions that were bad. However, since the motion estimation depend on the tracking of the landmarks, they are necessarily intertwined, so a combination of both reasons is likely.

Chapter 5

Discussion

5.1 Choice of front-end

SVO was chosen as the front-end algorithm to base this work on. The main reasons for choosing this method was that it uses direct matching to find feature correspondences while simultaneously triangulating a map of 3D landmarks. Additionally, an open-source C++ implementation was available, saving development time as only the parts related to direct matching had to be adapted to work with 16-bit images.

Firstly, triangulating landmarks makes fusing the visual system with inertial measurements simpler. They can easily be incorporated with the visual measurements as projection factors in a factor graph and the inertial measurements as preintegration factors. This made the overall factor graph design of the GTSAM back-end simple and familiar. This will be discussed more in Section 5.4.

Secondly, Khattak et al. [32] showed empirically that direct methods operating on the full 16-bit radiometric gave the best results for thermal odometry estimation. This motivated the choice of a direct method. Moreover, most indirect methods are based on using feature descriptors that are made to work with 8-bit images. To make these work with 16-bit images, the descriptor would need to be modified to work directly on 16-bit images, or the 16-bit thermal image would need to be rescaled using one of the methods presented in Section 2.1.1. The primary concern with the latter was if it is robust to significant illumination changes when the thermal camera performs an FFC. Since SVO use direct matching, however, the illumination changes can easily be handled by illumination gain and offset estimation. This is a strength of direct matching, as the modifications necessary to have gain and offset estimation are minimal.

5.2 STIO algorithm

The STIO algorithm and the front-end were tested on three thermal datasets in different environments, each with distinct characteristics.

STIO appears to be heavily dependent on finding suitable parameters for the environment. The grid cell size and the FAST threshold were observed to be especially important. For example, a grid cell size that worked well for the Vicon arena did not work well for the Urban Parking Lot. Exactly what grid cell size to choose is hard to know a priori and will depend on the environment.

One could reduce the grid cell size to 1 pixel and potentially make the front-end use every pixel as a feature. This should select features from strong gradient areas but would most likely track too many features and render the optimization problem intractable since it triangulated a 3D landmark for each feature. In fact, this essentially describes a semi-dense method. The semi-dense method Direct Sparse Odometry (DSO) by Engel et al. [15] works almost in this. However, DSO handles the points more cleverly by incorporating the intensity error in the optimization and not relying on reprojection errors like STIO. It also only optimizes the depth in the camera frame instead of the full 3D position. This reduces the number of variables to optimize by two *per* landmark since the pixel location determine the location in the image plane.

In conclusion, the grid cell size was observed to be an important parameter, especially in the Urban Parking Lot dataset with distinct features in only half the frame and the Active Gold Mine with large, thermally flat walls. In this thesis, this parameter was tuned to best work with the given datasets, but in real applications, this is rarely the case.

Moreover, different FAST thresholds could be used depending on whether FAST8 or FAST16 were used. FAST8 could use a larger threshold than FAST16. This seems to make FAST8 detect and subsequently track points that are more locally distinct. Points detected by FAST8 were also observed to be tracked with greater reliability. This was an unexpected result, as histogram equalization necessarily deletes some information when it converts from 16-bit to 8-bit. However, it can be explained by the fact that histogram equalization increases the contrast in the image, making some features more visible (e.g., the spots on the floor in the starting area of the Active Gold Mine, see Fig. 3.10a). Increased contrast means an increased difference between neighboring pixels, and thus the FAST threshold can be set larger.

This can indicate that the FAST threshold should be set high to have only distinct points. However, this comes at the cost of not selecting many points. On the other hand, a low threshold gave many points, but at the cost of detecting and tracking more false-positive and noisy features. Hence, no threshold value works well for all settings. Some variation in what threshold worked best was observed in the thermal dataset, although a broader range gave decent results.

The results also seem to show a relationship between the number of tracked features and the robustness of the STIO estimate. While unsurprising, it indicates that STIO is a method that lends itself to tracking many landmarks. This is contrary to ROTIO. The authors of ROVIO found that it gave comparable results to other state-of-the-art methods with fewer tracked landmarks. The rationale is that the IMU gives a good prior on the motion of the system and merely needs recurring landmark observations to stabilize the estimate. [3] They argue, therefore, that it is better to have few high-quality landmarks (the authors of ROVIO use approximately 20) than many landmarks with a risk of not rejecting outliers. ROTIO also performed well with few tracked landmarks, so the same conclusion seems to hold for ROTIO as well. ROVIO is a method designed with VIO in mind. On the other hand, SVO is originally a pure VO method with VIO capabilities added through the Ceres back-end later. The number of tracked features is a crucial parameter in monocular VO. [47] It is therefore natural that SVO track many features, and that this is not changed for the VIO modification.

The outlier rejection module was, therefore, critical to remove false-positive features. This was especially true for the FAST8 and the 8-bit STIO which seemed to detect more noisy features than FAST16. This was because histogram equalization also increases the noise's contrast, making it more visible to the 8-bit feature detectors.

A surprising result was that the 8-bit STIO version worked as well as it did. This was just the original SVO with Ceres back-end VIO implementation used on histogram equalized images. All detection and matching are done directly on the 8-bit histogram equalized image. There seem to be two main reasons for this.

The first is that it can detect and track more features than any of the 16-bit methods. In addition to detecting the same features as 16-bit FAST8 did, it also detected more edgelet features. This was because the histogram equalization increased the contrast enough where edgelet features could be detected for the front-end to detect and track them. It is believed that one reason they were not detected and tracked as often in the 16-bit versions is that the temperature difference was too small for the detector to select that edge. This could be due to the incorrect FAST threshold for edgelet features.

The second reason the 8-bit version worked so well is that the direct matching between frames is not affected much by the illumination change introduced by FFC and when hot objects enter the frame as initially thought. This might be because the frequency of images is high enough and there is little motion between each frame, making the difference between two consecutive frames small enough to track. Moreover, the illumination gain and offset estimation likely also play a prominent role in finding the correct matches. This further support the claims made by other authors ([33, 32, 18]) that direct methods are better suited for thermal data.

Lastly, the noise parameters for the IMU biases were found to be essential for accurate estimation. Initially, the bias noise terms were erroneously set to too large. This greatly affected the estimated trajectory negatively. The 16-bit STIO methods were unable to complete Urban Parking Lot and the Active Gold Mine datasets, constantly failing in the thermally flat areas where it relied heavily on the IMU for motion prediction. In both cases, STIO never managed to recover because the estimate had drifted too far due to poor dead reckoning from the IMU for too long. When the bias was too noisy, the IMU preintegration accumulated too much error in the period of no front-end tracking, causing the pose to drift too far. This illustrates the importance of correct bias noise terms, either from the datasheet or an Allan variance analysis.

5.3 IMU and external prior initialization

The initialization procedure using IMU preintegration and external priors essentially describe an IMU and GPS system. This sensor setup is known to render both the accelerometer and gyroscope biases observable. This is shown in [54], where non-linear observability analysis is done for several sensor configurations, including an IMU and GPS system.

While this system renders the biases observable in theory, the results did not show this empirically. As seen in Section 4.3, the gyroscope biases converged relatively quickly to the ground truth, but the accelerometer biases did not. In fact, the observability of the biases also depends on the system's motion. [43] Little motion means the biases might not converge. Though the MAV did move in all degrees of freedom in the datasets, it might not move enough over a short enough period to achieve a reasonable estimation of the accelerometer bias. In simulated experiments, [43] found that significant rotation was necessary to reduce the accelerometer bias sufficiently. This might also explain why the accelerometer bias was closer to the ground truth for the Leica dataset when the initialization motion was included because the MAV was sufficiently excited during that motion.

The discrepancy in bias was also not due to the back-end optimizer used. The same setup was run using a Levenberg-Marquardt (LM) optimizer instead of iSAM2, but the results were identical. The only difference was that the LM optimizer was slower than iSAM2 and not able to solve the batch problem in real-time after enough states were added. On the other hand, the iSAM2 optimizer was able to solve the whole problem in real-time without any marginalization because it only updated the affected states. Coincidentally, this also shows the effectiveness of the iSAM2 optimizer.

As far as using this as an initialization system, it could work. While the biases did not converge to the ground truth values, they are not zero like they would be if no prior information about the biases is known. [43] also found that neglecting the acceleration biases in initialization will not pose a significant negative impact on the results. However, having an external prior available is unlikely in many VIO scenarios. One conceivable scenario is if multiple modalities are fused, for example, in visual-laser-inertial odometry using a LiDAR. Nevertheless, a specially designed VIO initialization system would still be preferable, e.g., like the one used in VINS-Mono [42, 43]. This had the added benefit of also initializing the map of landmarks.

5.4 GTSAM back-end

The proposed factor graph design for the GTSAM back-end is the most basic factor graph for Visual-Inertial Odometry and is nothing new. The design is described in [13, sec. 7.1], and also forms the basis for state-of-the-art VIO and SLAM systems such as ORB-SLAM3 [9]. Furthermore, the employed GTSAM factors are elementary factors that have been thoroughly tested by the community at large. Additionally, GTSAM has several unit tests and working examples for these factors and VIO. For these reasons, there is no doubt that the design should work given correct interactions with the front-end.

The depth filters of the SVO front-end ensure good data associations by only inserting converged points into the back-end as landmarks. The STIO algorithm using the Ceres back-end also proved to work well, showing that the front-end can work with a back-end. Moreover, as seen in Section 5.3, the IMU preintegration works well.

Therefore, the only explanation is that there is a programming error such that the GTSAM back-end either does not use the correct information from the front-end or that the back-end and front-end do not pass all the necessary information between each other. The main indication is that too many landmarks are removed as outliers because they move too much after optimization. As the Ceres back-end shows, this should not be the case.

However, despite strenuous debugging efforts by the author, the root issue was not found. The GTSAM back-end code follows the same structure as the Ceres back-end, and extra care and attention were put towards making sure that the interface between the back-end and front-end is identical between the two back-ends. However, the author must have missed something during code review, or there is an underlying logical bug that is not apparent.

To further debug this, one approach would be to run the Ceres back-end for some seconds so the estimated pose and landmarks can converge. All the information from the Ceres back-end can then be used to construct an identical factor graph using GTSAM. Then, the front-end algorithm is run as two instances, one using the Ceres back-end and the other using the newly constructed GTSAM back-end. The estimates for pose and landmarks are inspected for each frame to see when they diverge. By revising the GTSAM back-end code to make the estimates identical, the bug can hopefully be identified. This is a time-consuming and strenuous process but should, given enough time, allow the programmer to identify the bug.

5.5 Future work

Section 4.2 started by presenting gravity magnitudes that are different from the theoretical value where the datasets were recorded. This discrepancy is believed to be due to bias build-up in the IMU. It would be interesting to validate or disprove this belief by estimating the biases using all the available information using the recently published Super Odometry algorithm [58]. In short, this fuses IMU preintegration factors with odometry factors from other sources, for example visual-inertial and LiDAR-inertial odometry, to constrain and estimate the IMU biases.

One issue that was observed with using histogram equalization is that some areas of the image can become over-exposed. This is because regular histogram equalization only considers the global contrast of the image. This can reduce the number of possible features when using the FAST8 feature detector on histogram equalized images. It would therefore be interesting to apply an adaptive histogram equalization instead. For example, OpenCV provide a Contrast Limited Adaptive Histogram Equalization (CLAHE) implementation [5]. This divides the image into small tiles, and regular histogram equalization is done in each cell. To avoid amplifying noise, contrast limiting is applied. The example provided in [5] shows significant improvement to an 8-bit image. To use this for feature extraction with FAST8 however, this functionality would have to be reimplemented for 16-bit images.

It was also seen that the 16-bit STIO front-ends struggled due to few tracked features. This was mitigated by reducing the grid cell size to detect more features in the high gradient areas with more information. However, it would be interesting to instead adapt other direct methods to 16-bit that directly use only the high gradient areas. For example, Large Scale Direct SLAM (LSD-SLAM) [16] and DSO [15] are potential candidates. LSD-SLAM is a semi-dense method that directly estimates a depth map for the high gradient regions carrying information. A VIO formulation using the LSD-SLAM front-end is also available [52]. DSO samples pixels from the high gradient regions and directly tracks them. DSO also employs a photometric camera calibration that could be advantageous for thermal cameras. As briefly mentioned in Section 2.1, uneven camera temperature can cause a vignetting effect in the thermal image. By using a photometrically calibrated camera, this vignetting could be accounted for. Moreover, online intrinsic parameter estimation could also make the photometric calibration adapt to changes in temperature during operation.

For the GTSAM back-end, the obvious future work is to figure out the bug and fix it. As mentioned, this could be a time-consuming and strenuous process. Once the back-end is working, it should be compared to the Ceres back-end. The points of comparison could be accuracy, reliability, and speed. How does the APE compare, and do both methods remove the same number of outliers? Is iSAM2 able to run in real-time when more landmarks are added? How does the execution time compare to the marginalization strategy employed in the OKVIS Ceres implementation?

GTSAM provide an experimental Incremental Fixed Lag (IFL) smoother. This is based on iSAM2, but marginalize out variables when they become too old. This way it keeps a fixed maximal time horizon for the iSAM to solve and prevents the problem to grow indefinitely. With a working GTSAM back-end based on iSAM2, it should be simple to change it to use IFL. This should be compared to the iSAM2 implementation in terms of speed and accuracy. Is there a significant speed-up by marginalizing out old states? How much accuracy is lost by marginalizing out old states? What is a good time lag that balance the trade-off between speed and accuracy?

Chapter 6

Conclusion

The main objectives for this master's thesis were to convert a state-of-the-art VIO algorithm to work on 16-bit radiometric thermal images, implement a factor graph-based optimization back-end for VIO using the GTSAM library, and evaluate them on real-life datasets. The chosen state-of-the-art VIO method was the Semi-direct Visual Odometry (SVO) front-end algorithm. This was chosen because it uses direct matching instead of feature descriptors and triangulates landmark positions. The former made it simple to convert the method to use 16-bit images using reinterpret casting in C++. The direct matching also proved useful as it enabled illumination change estimation when the thermal camera did FFC. The latter made it easy to incorporate the landmarks as projection factors in the GTSAM factor graph. The converted method was dubbed Semi-direct Thermal-Inertial Odometry (STIO).

Evaluation on the thermal dataset showed that the 16-bit STIO methods were heavily dependent on the parameters, especially the grid cell size and the FAST threshold. Both affected the number of tracked features and landmarks, which was observed to be an important factor for the accuracy and reliability of the estimate. Moreover, the FAST threshold could be set higher when using FAST8. This indicated that FAST8 was able to detect features that are more locally distinct and therefore track them more easily. Furthermore, the illumination gain and offset estimation was found to be vital for reliable tracking. Without it, the front-end would consistently fail after data interruption when the thermal camera did FFC. This was because the illumination was often significantly changed. Surprisingly, the original 8-bit version of SVO with the Ceres back-end used directly on only histogram equalized images often outperformed the 16-bit STIO. This was because the 8-bit version detected more edgelet features in addition to the same corner features as the 16-bit versions did. Lastly, STIO was seen to benefit from the motion priors from the IMU in matching and tracking. This also made the 16-bit STIO methods initialize markedly quicker than the VO-only counterparts.

The GTSAM back-end design was not completed due to programming errors by the author that was not identified despite much debugging. The issue is believed to be due to inadequate or incorrect data being passed between the front-end and back-end, but it is unclear if this is due to a logical bug or if some data is missing. Nevertheless, given correct information exchange between the front-end and back-end, the overarching design of the GTSAM back-end should work as it is just a basic VIO design based on IMU preintegration found in the literature and other VIO methods.

The IMU preintegration and external prior initialization routine was not seen to converge the biases despite them being observable in theory. This is believed to be because of a lack of enough exciting motion. While it could still work as an initialization routine, the external pose is rarely available in VIO systems, and it would be preferable to use a specially designed VIO initialization system instead.

Bibliography

[1] Agarwal, S., Mierle, K., Others, . Ceres solver. `http://ceres-solver.org`.

[2] Baker, S., Matthews, I., 2004. Lucas-kanade 20 years on: A unifying framework part 1: The quantity approximated, the warp update rule, and the gradient descent approximation. International Journal of Computer Vision - IJCV .

[3] Bloesch, M., Omari, S., Hutter, M., Siegwart, R., 2015. Robust visual inertial odometry using a direct ekf-based approach, ETH-Zürich, Zürich. doi:`10.3929/ethz-a-010566547`. 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); Conference Location: Hamburg, Germany; Conference Date: September 28 - October 2, 2015.

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

[5] Bradski, G., N/A. Histograms - 2: Histogram Equalization. Online tutorial at OpenCV's webpage. Accessed online June 2, 2022. URL: `https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html`.

[6] Burri, M., Nikolic, J., Gohl, P., Schneider, T., Rehder, J., Omari, S., Achtelik, M.W., Siegwart, R., 2016. The EuRoC micro aerial vehicle datasets. The International Journal of Robotics Research URL: `https://projects.asl.ethz.ch/datasets/doku.php?id=kmavvisualinertialdatasets`, doi:`10.1177/0278364915620033`.

[7] Byrnes, J., 2008. Unexploded Ordnance Detection and Mitigation. NATO Science for Peace and Security Series B: Physics and Biophysics, Springer Netherlands.

[8] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., Leonard, J.J., 2016. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. IEEE Transactions on Robotics 32, 1309–1332. doi:`10.1109/TRO.2016.2624754`.

[9] Campos, C., Elvira, R., Rodriguez, J.J.G., Montiel, J.M.M., Tardos, J.D., 2021. ORB-SLAM3: An accurate open-source library for visual, visual–inertial, and multimap SLAM. IEEE Transactions on Robotics 37, 1874–1890. URL: `https://doi.org/10.1109%2Ftro.2021.3075644`, doi:`10.1109/tro.2021.3075644`.

[10] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., 2009. Introduction to Algorithms. 3nd ed., The MIT Press.

[11] Defense Advanced Research Projects Agency (DARPA), 2020. Broad Agency Announcement Invisible Headlights (IH). URL: `https://research-authority.tau.ac.il/sites/resauth.tau.ac.il/files/DARPA-Invisible-Headlights-%20HI-HR001120S0045.pdf`.

[12] Dellaert, F., 2020. GTSAM concepts. `https://github.com/borglab/gtsam/blob/develop/GTSAM-Concepts.md`.

[13] Dellaert, F., Kaess, M., 2017. Factor graphs for robot perception. Foundations and Trends in Robotics 6, 1–139. doi:`10.1561/2300000043`.

[14] Egeland, O., Gravdahl, T., 2002. Modeling and Simulation for Automatic Control. Marine Cybernetics, Trondheim.

[15] Engel, J., Koltun, V., Cremers, D., 2016. Direct sparse odometry, in: arXiv:1607.02565.

[16] Engel, J., Schöps, T., Cremers, D., 2014. Lsd-slam: Large-scale direct monocular slam, in: Computer Vision – ECCV 2014, Springer International Publishing, Cham. pp. 834–849.

[17] Falck, E., 2021. Retrofitting a multi-link robot with a LiDAR-visual-intertial sensor suite for sensor fusion (Project thesis). Technical Report. NTNU.

[18] Flemmen, H.D., 2021. ROVTIO: RObust Visual Thermal Inertial Odometry. Master's thesis. NTNU. Trondheim.

[19] FLIR, 2018. Flat field correction control. Technical reference. Accessed online March 23, 2022. URL: `http://softwareservices.flir.com/BFS-U3-120S4/latest/Model/public/FlatFieldCorrectionControl.html`.

[20] Forster, C., Carlone, L., Dellaert, F., Scaramuzza, D., 2015a. Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation, Georgia Institute of Technology. doi:`1853/55417`.

[21] Forster, C., Carlone, L., Dellaert, F., Scaramuzza, D., 2015b. Supplementary material to: Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation, Georgia Institute of Technology. doi:`1853/53653`.

[22] Forster, C., Pizzoli, M., Scaramuzza, D., 2014. SVO: Fast semi-direct monocular visual odometry, in: 2014 IEEE International Conference on Robotics and Automation (ICRA), pp. 15–22. doi:`10.1109/ICRA.2014.6906584`.

[23] Forster, C., Zhang, Z., Gassner, M., Werlberger, M., Scaramuzza, D., 2017. Svo: Semidirect visual odometry for monocular and multicamera systems. IEEE Transactions on Robotics 33, 249–265. doi:`10.1109/TRO.2016.2623335`.

[24] Furgale, P., Maye, J., Rehder, J., Oth, L., et al., 2014. Kalibr. `https://github.com/ethz-asl/kalibr`.

[25] Gonzalez, R.C., Woods, R.E., 2006. Digital Image Processing (3rd Edition). Third ed., Prentice-Hall, Inc.

[26] Grupp, M., 2017. evo: Python package for the evaluation of odometry and slam. `https://github.com/MichaelGrupp/evo`.

[27] Hartley, R., Zisserman, A., 2004. Multiple View Geometry in Computer Vision. 2 ed., Cambridge University Press. doi:`10.1017/CBO9780511811685`.

[28] Jewel, E.B., Ponce, D.A., Morin, R.L., 2000. Principal Facts for Gravity Stations in the Antelope Valley–Bedell Flat area, West-Central Nevada. Technical Report. U.S. Geological Survey. URL: `https://pubs.usgs.gov/of/2000/0506/`.

[29] Kaess, M., Ila, V., Roberts, R., Dellaert, F., 2010. The bayes tree: An algorithmic foundation for probabilistic robot mapping , 157 – 173.

[30] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J.J., Dellaert, F., 2012. isam2: Incremental smoothing and mapping using the bayes tree. The International Journal of Robotics Research 31, 216–235. doi:`10.1177/0278364911430419`.

[31] Kaess, M., Ranganathan, A., Dellaert, F., 2008. isam: Incremental smoothing and mapping. IEEE Transactions on Robotics 24, 1365–1378. doi:`10.1109/TRO.2008.2006706`.

[32] Khattak, S., Mascarich, F., Dang, T., Papachristos, C., Alexis, K., 2019a. Robust thermal-inertial localization for aerial robots: A case for direct methods, in: 2019 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 1061–1068. doi:`10.1109/ICUAS.2019.8798279`.

[33] Khattak, S., Papachristos, C., Alexis, K., 2019b. Keyframe-based direct thermal–inertial odometry. 2019 International Conference on Robotics and Automation (ICRA) URL: `http://dx.doi.org/10.1109/ICRA.2019.8793927`, doi:`10.1109/icra.2019.8793927`.

[34] Lay, D., Lay, S., McDonald, J., 2016. Linear Algebra and Its Applications. Fifth ed., Pearson.

[35] Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R., Furgale, P., 2015. Keyframe-based visual–inertial odometry using nonlinear optimization. The International Journal of Robotics Research 34, 314–334. URL: https://doi.org/10.1177/0278364914554813, doi:10.1177/0278364914554813.

[36] Lucas, B.D., Kanade, T., 1981. An iterative image registration technique with an application to stereo vision, in: Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. p. 674–679.

[37] Lupton, T., Sukkarieh, S., 2012. Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions. IEEE Transactions on Robotics 28, 61–76. doi:10.1109/TRO.2011.2170332.

[38] Ma, Y., Soatto, S., Košecká, J., Sastry, S.S., 2004. An Invitation to 3-D Vision. Interdisciplinary Applied Mathematics, Springer New York. doi:10.1007/978-0-387-21779-6.

[39] Nister, D., 2004. An efficient solution to the five-point relative pose problem. IEEE Transactions on Pattern Analysis and Machine Intelligence 26, 756–770. doi:10.1109/TPAMI.2004.17.

[40] Papachristos, C., Mascarich, F., Alexis, K., 2018. Thermal-inertial localization for autonomous navigation of aerial robots through obscurants, in: 2018 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 394–399. doi:10.1109/ICUAS.2018.8453447.

[41] Pizzoli, M., Forster, C., Scaramuzza, D., 2014. Remode: Probabilistic, monocular dense reconstruction in real time, in: 2014 IEEE International Conference on Robotics and Automation (ICRA), pp. 2609–2616. doi:10.1109/ICRA.2014.6907233.

[42] Qin, T., Li, P., Shen, S., 2018. Vins-mono: A robust and versatile monocular visual-inertial state estimator. IEEE Transactions on Robotics 34, 1004–1020. doi:10.1109/TRO.2018.2853729.

[43] Qin, T., Shen, S., 2017. Robust initialization of monocular visual-inertial estimation on aerial robots, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4225–4232. doi:10.1109/IROS.2017.8206284.

[44] Rosten, E., Drummond, T., 2006. Machine learning for high-speed corner detection, in: Leonardis, A., Bischof, H., Pinz, A. (Eds.), Computer Vision – ECCV 2006, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 430–443.

[45] Solà, J., 2017. Quaternion kinematics for the error-state kalman filter. URL: https://arxiv.org/abs/1711.02508, doi:10.48550/ARXIV.1711.02508.

[46] Solà, J., Deray, J., Atchuthan, D., 2018. A micro lie theory for state estimation in robotics. arXiv:1812.01537.

[47] Strasdat, H., Montiel, J.M.M., Davison, A.J., 2010. Real-time monocular slam: Why filter?, in: 2010 IEEE International Conference on Robotics and Automation, pp. 2657–2664. doi:10.1109/ROBOT.2010.5509636.

[48] Szeliski, R., 2010. Computer Vision: Algorithms and Applications. Springer. URL: https://szeliski.org/Book/.

[49] Teledyne FLIR, . FLIR UAS IR Camera FAQ. Accessed online March 23, 2022. URL: https://www.flir.com/suas/flir-uas-ir-camera-faq/.

[50] Teledyne FLIR, 2021. What are ir camera lenses made of? Accessed online March 23, 2022. URL: https://www.flir.com/discover/rd-science/what-are-ir-camera-lenses-made-of/.

[51] Tomasi, C., Kanade, T., 1991. Detection and Tracking of Point Features. Technical Report. International Journal of Computer Vision.

[52] Usenko, V., Engel, J., Stückler, J., Cremers, D., 2016. Direct visual-inertial odometry with stereo cameras, in: 2016 IEEE International Conference on Robotics and Automation (ICRA), pp. 1885–1892. doi:10.1109/ICRA.2016.7487335.

[53] Vogiatzis, G., Hernández, C., 2011. Video-based, real-time multi-view stereo. Image and Vision Computing 29, 434–441. URL: https://www.sciencedirect.com/science/article/pii/S0262885611000138, doi:https://doi.org/10.1016/j.imavis.2011.01.006.

[54] Weiss, S.M., 2012. Vision based navigation for micro helicopters. Ph.D. thesis. ETH Zurich. Zürich. doi:10.3929/ethz-a-007344020.

[55] Workswell Infrared Cameras and Systems, . Radiometric and non-radiometric thermal camera. Accessed online Marhc 23, 2022. URL: https://workswell-thermal-camera.com/radiometric-and-non-radiometric-thermal-camera/.

[56] Young, H., Freedman, R., 2015. University Physics with Modern Physics. Pearson Education. URL: https://books.google.no/books?id=WsigBwAAQBAJ.

[57] Zhang, J., Singh, S., 2014. LOAM: Lidar odometry and mapping in real-time, in: Proceedings of Robotics: Science and Systems (RSS ’14).

[58] Zhao, S., Zhang, H., Wang, P., Nogueira, L., Scherer, S., 2021. Super odometry: Imu-centric lidar-visual-inertial estimator for challenging environments. URL: https://arxiv.org/abs/2104.14938, doi:10.48550/ARXIV.2104.14938.

Eirik Flemsæter Falck

Implementation and evaluation of 16-bit thermal-inertial odometry using non-linear factor graphs