Observability-Informed Measurement Validation for Visual-Inertial Navigation

by

Matthew Boler

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 7, 2022

Keywords: Computer Vision, SLAM, Kalman Filter, Nonlinear Optimization, Navigation

Approved by

Scott Martin, Chair, Assistant Research Professor of Mechanical Engineering
David Bevly, Professor of Mechanical Engineering
Thaddeus Roppel, Associate Professor of Electrical and Computer Engineering
Stanley Reeves, Professor of Electrical and Computer Engineering

This work is dedicated to my family.

Abstract

This thesis presents a measurement validation method for visual-inertial navigation and an implementation of a visual-inertial estimator which makes use of it. Many autonomous platforms, especially flying ones, rely on accurate and reliable state estimates from a visual-inertial estimator to maintain safe and controlled flight towards a goal. The measurement validation method presented in this thesis makes use of a geometric analysis of the landmark measuremnt model to enable early and reliable validation, safely integrating high-quality measurements into the state estimation process. First, the IMU and camera sensors are detailed along with the sensor processing techniques necessary to make use of them. Next, a detailed description of two standard visual-inertial estimation approaches is presented to develop necessary background knowledge. Following these descriptions, an analysis of the relationships between the geometry of landmark observation and the accuracy and reliability of landmark estimates is performed, concluding with the proposal of a new validation method which delays measurement processing until the landmark is predicted to be observable. Lastly, a visual-inertial estimator is developed which makes use of the proposed method and tested on the EUROC dataset, the most common visual-inertial dataset, against several state-of-the-art estimators. In this comparison, the proposed estimator demonstrates competitive performace, reliably producing positioning errors of less than 0.5 meters over flights up to 120 meters long. Overall, the proposed method is demonstrated to be reliable and accurate in competition with significantly more advanced and complicated estimators.

Acknowledgments

First and foremost, I'd like to thank Dr. Scott Martin for his unending support. The ideas I have presented him with have tested the limits of the word "plausible" and he has responded universally with enthusiasm and suggestion. This thesis would have never been written if not for his mentorship. I owe a massive thanks to Dr. David Bevly for recruiting me to the lab in the first place and for his absolute devotion to his students, both in the lab and on campus in general. Lastly, I'd like to thank Dr. Howard Chen for being a brilliant sounding board and for preventing me from falling in with the Euler-angled masses.

While the mentors who guided me are responsible for the quality and completion of this Masters degree, the coworkers and friends I have made in the lab are responsible for my thorough enjoyment of the process. Will Bryan has been my partner in crime every step of the way and, despite our intimate familiarity with Murphy's Law, our projects have somehow always come together when they needed to. I'd like to thank Brendan Schretter for his sage wisdom in the art of un-breaking filters and for introducing me to SSBM. Jake Ward deserves a truckload of thanks for his unbridled enthusiasm and his willingness to learn everything that anyone has ever found interesting. Anderson Givhan and Amy Strong have been stalwart friends and high benchmarks for academic achievement and musical taste. All of these fellow GAVLab students have helped me through tough times, in research and in life, and have made grad school so enjoyable I couldn't say no to another round. Thank you all.

My family deserves more credit than I know how to give them. My mom and dad have given me everything I have ever needed in the pursuit of my dreams and have been my strongest supporters. You have always made me feel loved. My brother Finn has been an inspiration his whole life and I'm proud to be his brother.

Lastly, my girlfriend Madison. Thank you for sticking with me through this. You deserve the world. I love you.

Oh, and Mr. Bond, Bilbo, Ally, and Brody. You all have been a beautiful emotional support troupe and I'm grateful for all of you. Stop causing so much trouble.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

EKF	Extended Kalman Filter
EsKF	Error State Extended Kalman Filter
GN	Gauss-Newton
GPS	Global Positioning System
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
KF	Kalman Filter
MAV	Micro Aerial Vehicle
PDF	Probability Density Function
SLAM	Simultaneous Localization and Mapping
SO3	Special Orthogonal Group (3)
VINS	Visual-Inertial Navigation System

Chapter 1

Introduction

"The best nations are those most widely related; and navigation, as effecting a world-wide mixture, is the most potent advancer of nations." - Ralph Waldo Emerson

1.1 Background and Motivation

Despite the rapid push for autonomy, reliable autonomous platforms are essentially outliers in the field of robotics as current products and platforms are constrained to tightly controlled environments. The fundamental question of *Where am I?* is of absolute importance to autonomous mobile robots, requiring them to process information about their motion and environment from a collection of sensors to estimate their locations. Systems that perform this task are called *navigation systems* and have been an active area of research preceding computers themselves. The classical navigation system consists of an IMU and a GPS receiver. Systems such as these provide positioning solutions to everything from commercial airplanes to autonomous lawn mowers.

Figure 1.1: An autonomous lawn mower, produced by Husqvarna, utilizing GPS to map out a lawn

Despite being so widely relied upon, these classical systems struggle in challenging environments such as indoors, underground, and inside urban canyons. To address these challenges GPS-denied navigation methods such as simulatneous localization and mapping (SLAM) have become highly active areas of research, driven by newer computers' abilities to process more and more complex data from sensors such as cameras and LIDARs. Of particular interest is the *visual-inertial navigation system*, which fuses data from a camera and an IMU. These systems can provide highly accurate positioning information as well as detailed contextual information about their environments, all using a sensor package already built into every smartphone in the world. Given accurate measurements, visual-inertial estimators regularly produce positioning errors of less than 1% of distance travelled, however faulty measurements will degrade accuracy at best and cause total failure at worst. In this work we propose a method of measurement validation for visual-inertial navigation to ensure that only valid measurements are included in the navigation solution to maximize the accuracy and robustness of the estimator.

1.2 Related Work

There have been a wide range of methods proposed for aiding an IMU with a camera. Cameras have been used to aid terrain-relative navigation systems for missiles, airplanes, and spacecraft

since their introduction. The mars rovers Spirit and Opportunity first made popular visual odometry as a method of navigating on the surface of Mars [48], and now the Mars helicopter employs a more advanced version of the same system to enable short autonomous flights on Mars [3]. In more recent civillian news, visual-inertial estimation has made headlines as the backbone of semi-autonomous consumer drones and have even begun testing for autonomous drone racing [12].

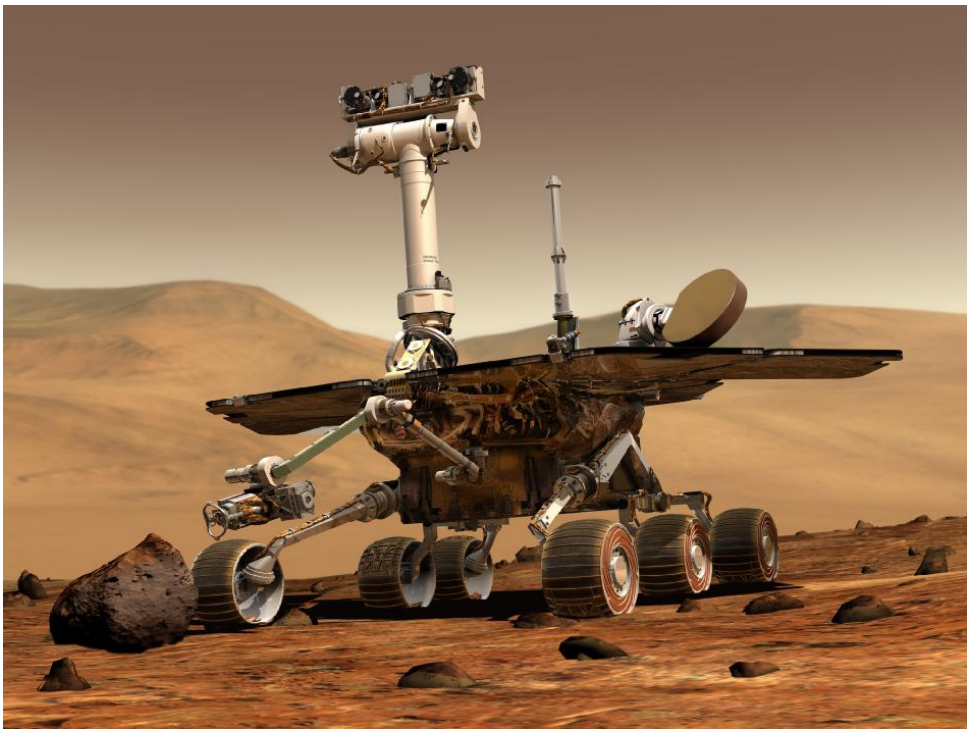

Figure 1.2: NASA's Spirit rover, one of the first major systems to employ visual odometry for navigation

Figure 1.3: The AlphaPilot autonomous racing drone[15]

Visual-inertial estimation can be said to have arrived with Mourikis et al.'s groundbreaking contribution of an extremely efficient vision-aided Extended Kalman Filter [37]. Improvements to this architecture were then made to include SLAM functionality [38] and to optimize the operation of the estimator to maintain performance on resource-constrained systems [33]. Following the exponential growth in computing power of the late 2000's and early 2010's an explosion of nonlinear optimization-based methods arrived [31] [32] [10] [39] following an earlier trend in vision-only navigation. More recent work has expanded visual-inertial fusion to include direct methods which operate directly on the raw images themselves rather than including any preprocessing steps [46]. Visual-inertial estimation has also been a primary testing ground for the development of invariant Kalman filters [8] [23] [49]. For a considerably more in-depth summary of the history and developments of visual-inertial navigation, please read the excellent survey by Chen et al. [11]. A selection of highlights are explained in much greater detail later in this work, so for now it is enough to say that visual-inertial estimation has a long history of implementation at the most cutting-edge areas of robotics and only appears to be solidifying its position as the standard navigation system for small robots.

1.3 Research Contributions

A list of the contributions of this thesis are shown below.

- A tutorial on filtering and optimization methods for visual-inertial navigation
- An analysis of the effects of observation geometry on feature position estimation
- A new measurement validation method informed by geometric observability constraints
- An implementation of a visual-inertial estimator utilizing the proposed method to achieve performance competitive with the state-of-the-art on a standard dataset

1.4 Thesis Outline

This thesis has six remaining chapters. Chapter 2 outlines the sensors used in the field of visual-inertial navigation and presents common sensor models. Chapter 3 develops the methodologies used in filtering and optimization for estimation. Chapter 4 extends the methodologies presented in the previous chapter to develop and explain the families of visual-inertial estimation techniques. Chapter 5 presents an analysis of the effects of observation geometry on landmark estimation in visual-inertial SLAM and proposes an adaptive measurement validation algorithm. Chapter 6 presents results from applying the proposed method on standard datasets and compares them to existing state-of-the-art methods. Chapter 7 summarizes the work presented and outlines future work.

The included appendices build the needed background knowledge to implement the described estimators as well as additional results. Appendix A builds the necessary knowledge of the quaternion representation of attitude. Appendix B demonstrates methods of numerically integrating dynamic equations. Appendix C includes further results plots.

A side goal of this work is to provide an introduction to visual-inertial estimation to anyone who, like myself, comes into the field with very little background knowledge. Chapters 2, 3, and 4 form a thorough tutorial on VINS and should provide the necessary background information a new graduate student, researcher, or engineer needs to pick up a visual-inertial navigation project.

1.5 Notation

- Scalars are lower-case italic letters: a
- Vectors are lower-case bold letters: $\mathbf{a}$
- Matrices are upper-case bold letters: $\mathbf{A}$
- Reference frames are upper case italic: G
- Vectors can be expressed in a frame: ${}^{G}\mathbf{a}$
- Rotation matrices are denoted ${}^{to}_{from}\mathbf{C}$
  - ${}^{G}_{L}\mathbf{C}$ rotates a vector from frame L to frame G
- Time derivatives have dots: $\dot{a}$
- Estimated values have hats: $\hat{a}$
- Continuous equations are denoted: $f(x)$
- Discrete equations are denoted : $f(x_k)$
- Normalized coordinates are denoted: $(x, y)^T$
- Pixel coordinates are denoted $(u, v)^T$
- Distorted pixel coordinates are denoted $(u', v')^T$
- $\boxplus$ denotes the generic composition operator:
  - Vectors: $\boldsymbol{a} \boxplus \boldsymbol{b} = \boldsymbol{a} + \boldsymbol{b}$
  - Rotation matrices: $\boldsymbol{C}_1 \boxplus \boldsymbol{C}_2 = \boldsymbol{C}_1\boldsymbol{C}_2$
  - Quaternions: $\boldsymbol{q}_1 \boxplus \boldsymbol{q}_2 = \boldsymbol{q}_1 \otimes \boldsymbol{q}_2$

Chapter 2

Sensors

The canonical visual-inertial navigation consists of just that: an inertial measurement unit (IMU) and a camera. While it is common for navigation platforms to consist of these sensors in conjunction with several others, the minimal VINS sensor suite is considered here.

2.1 Inertial Measurement Units

IMUs consist of a combination of accelerometers, gyroscopes, and occasionally magnetometers or barometers. Modern microelectromechanical system (MEMS) IMUs are ubiquitous in smaller navigation platforms as they are cheap to produce and have minimal size, weight, and power requirements. While the magnetic and barometric sensing capabilities have found use in navigation systems, we will focus on the acceleromter and gyroscope sensors for use in visual-inertial navigation.

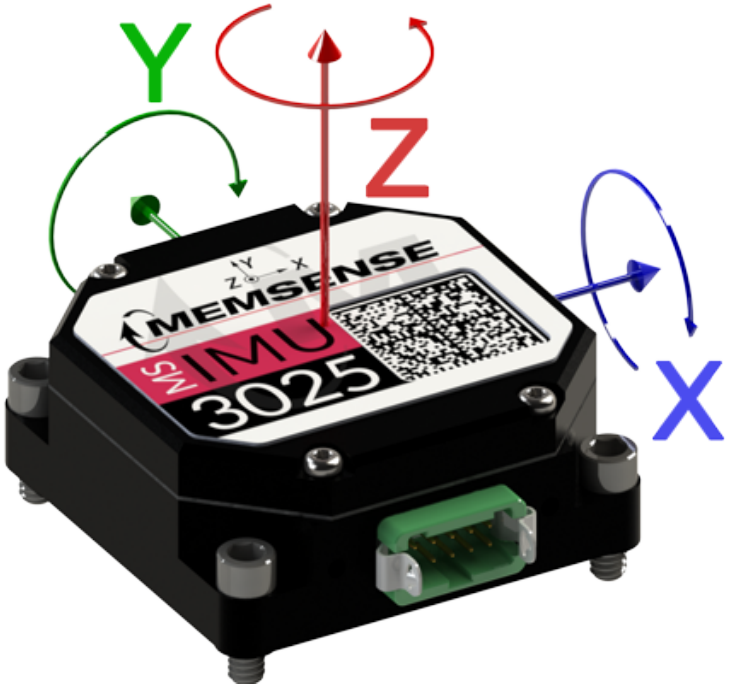

Figure 2.1: A common MEMS IMU

An accelerometer measures the specific force the sensor is exposed to. This measurement is considered relative to a non-accelerating frame, and thus includes measurements of the effects of gravity. The accelerometers we consider are actually groups of three single-axis devices with their measurement axes aligned orthogonally, resulting in measurements $\boldsymbol{a}_m = [f_x, f_y, f_z]^T$. A perfect accelerometer would then have the measurement model shown in Equation 2.1.

$$\boldsymbol{a}_m = {}^B\boldsymbol{a}_b - {}^B\boldsymbol{g} \tag{2.1}$$

Unfortunately, we operate in a world incoherent to spherical chickens and so must account for measurement errors. Inertial sensors are subject to two common errors: random noise and measurement bias. Addressing these errors, we arrive at our canonical accelerometer measurement model as shown in Equation 2.2.

$$\boldsymbol{a}_m = {}^B_G\boldsymbol{C}({}^G\boldsymbol{a}_b + {}^G\boldsymbol{g}) + \boldsymbol{b}_a + \boldsymbol{w}_a \tag{2.2}$$

where ${}^{G}\boldsymbol{a}_b$ is acceleration of the body expressed in the global frame, ${}^{G}\boldsymbol{g}$ is the gravity vector expressed in the global frame, $\boldsymbol{b}_a$ is the accelerometer bias vector, and $\boldsymbol{w}_a$ is random noise.

A gyroscope measures the angular rate of the sensor. Like the accelerometer before, errors must be accounted for. The canonical gyroscope measurement model is then shown in Equation 2.3.

$$\boldsymbol{\omega}_m = {}^{B}\omega_b + \boldsymbol{b}_g + \boldsymbol{w}_g \tag{2.3}$$

where ${}^{B}\omega_b$ is the angular rate of the body expressed in the body frame, $\boldsymbol{b}_g$ is the gyroscope bias vector, and $\boldsymbol{w}_g$ is random noise.

For both accelerometers and gyroscopes, the bias vectors can be modeled as slowly-varying vectors driven by random walk processes $\boldsymbol{w}_{wa} \sim N(0, \boldsymbol{Q}_{wa})$ and $\boldsymbol{w}_{wg} \sim N(0, \boldsymbol{Q}_{wg})$, respectively. Additional accelerometer and gyroscope errors result from scale factor error, cross-coupling effects, and temperature effects. These are considered out of the scope of this work. For more detail, consider chapter 4 in [19].

Despite being the foundation of most navigation platforms, the IMU is almost unusable for navigation on its own. Deriving the desired quantities of position, velocity, and attitude from the measured accleration and angular rate requires integrating the noisy sensor measurments. As a result, we can expect position, velocity, and attitude errors to grow unbounded with time [19]. Because of this, it is effectively a requirement that all inertial navigation systems (INS) have an aiding sensor of some form to correct the drifting state estimate and bound errors. Classically this sensor is a GPS reciever, although barometers, magnetometers, cameras, lidars, radars, and wheelspeed sensors are all commonly used. For a more detailed treatment, consider chapter 5 in [19] and the tutorial on inertial navigation [20] by the same author.

2.2 Cameras

In contrast to the ability of an IMU to measure quantities directly relating to the motion of the body, cameras measure information about the environment relative to the body. Consisting of

a lens and a photosensitive array, modern digital cameras are fundamentally directional light sensors in that they measure the intensity of light arriving across the field of view of the camera. The lens collects the light from this range of directions and focuses it onto the photosensitive array to be sampled. Each discrete sampling location on the array is known as a pixel. At each sample time, every pixel is sampled to form a matrix of intensity values which we call an image.

Figure 2.2: A small form factor camera

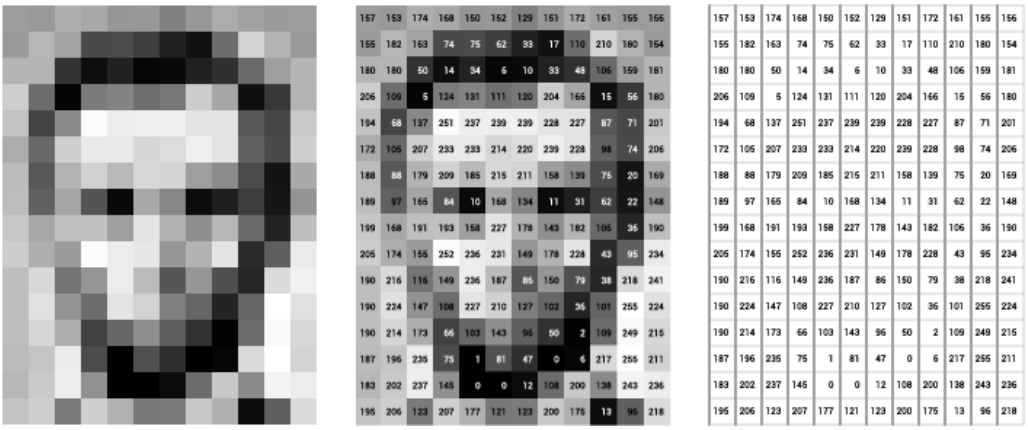

Figure 2.3: An example of an image as a grid of discrete values

2.2.1 Image Formation

The process of forming an image from a view of the world is governed by projective geometry. As the camera performs a mapping of the three-dimensional world onto a two-dimensional grid, one degree of freedom is lost. Specifically, depth information is lost, resulting in scale

ambiguity in images. This is what makes photos of tourists ”leaning against” the Eiffel Tower possible.

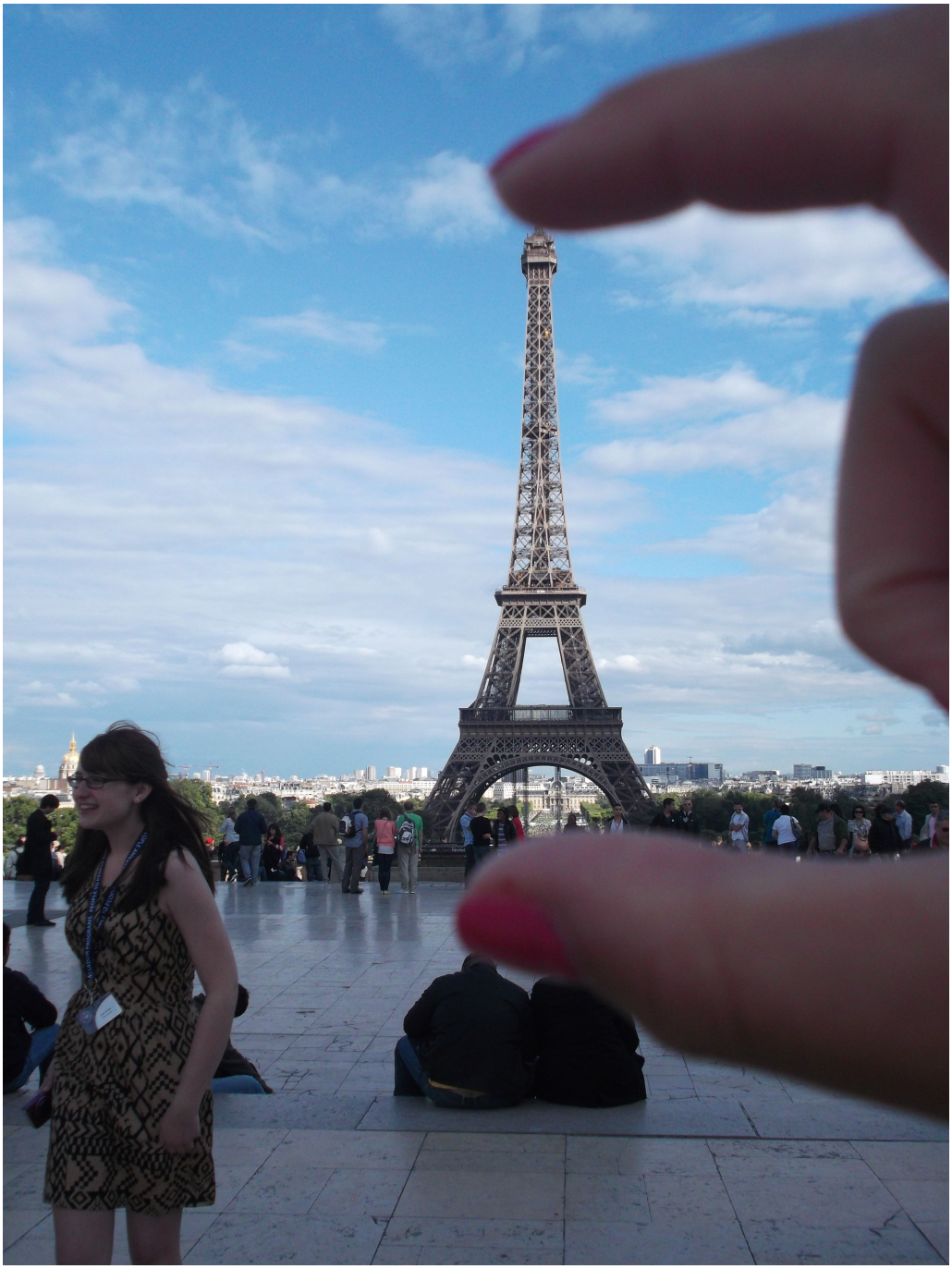

Figure 2.4: An example of scale ambiguity in an image

We use the pinhole camera model to describe image formation. This model describes the camera as a point in space. Light travels from the environment through this point, called the focal point, and projects onto the image plane behind it. The distance between the focal point

and the image plane is the *focal length* of the camera. This projected image is inverted, so for simplicity and intuition's sake we consider the image on the virtual image plane in front of the focal point. This model is illustrated below in figure 2.5 and the simplified model using only the virtual image plane in figure 2.6. It is important to note that most camera models, including our treatment here, use the coordinate frame:

- Positive X: To the right of the camera
- Positive Y: To the bottom of the camera
- Positive Z: To the front of the camera

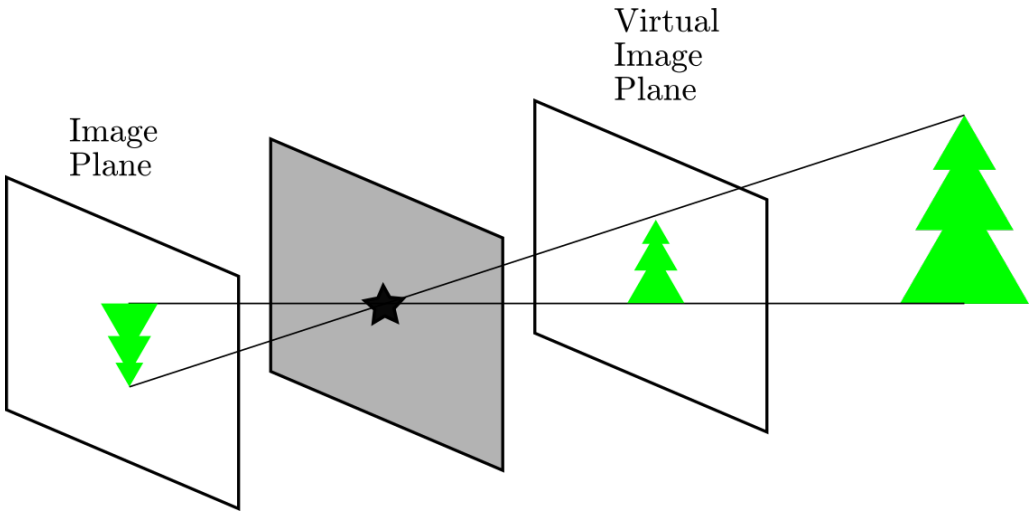

Figure 2.5: Projection of an object onto the image plane

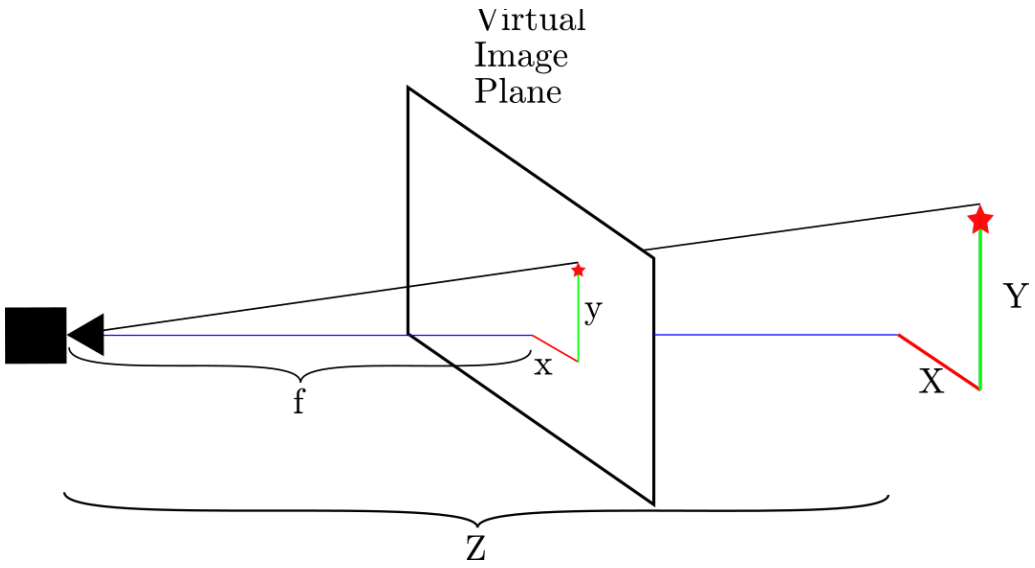

Figure 2.6: The considered perspective projection camera model

From this model we can derive the equation that projects a three-dimensional point onto the image plane at a pixel location.

We begin with the mapping of a point in camera coordinates to an idealized image plane with unity focal length. We denote this function the *normalized projection function*, and the resulting coordinates *normalized coordinates*. Using the quantities shown in figure 2.6, the mapping is given in Equation 2.4.

$$\begin{aligned} h_{norm}({}^{C}\boldsymbol{p}_f) &= \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \\ &= \frac{1}{{}^{C}Z_f} \begin{bmatrix} {}^{C}X_f \\ {}^{C}Y_f \\ {}^{C}Z_f \end{bmatrix} \end{aligned} \tag{2.4}$$

These normalized coordinates are still in units of meters, and another function is required to map them to the pixel coordinates of the camera. We begin by noting that the coordinates of

the point on the camera's true image plane are given by Equation 2.5.

$$\begin{aligned} x &= f\frac{{}^{C}X_f}{{}^{C}Z_f} \\ y &= f\frac{{}^{C}Y_f}{{}^{C}Z_f}. \end{aligned} \tag{2.5}$$

Converting these to pixels requires dividing the focal length f by the width of a pixel in meters, d. This width commonly varies in the horizontal and vertical directions, denoted as the two focal lengths f_x and f_y, given in Equation 2.6.

$$\begin{aligned} f_x &= \frac{f}{d_x} \\ f_y &= \frac{f}{d_y}. \end{aligned} \tag{2.6}$$

Additionally, we must account for the image coordinate system having $(0, 0)$ at the top left of the image by adding the location of the optical center (c_x, c_y) in pixels, resulting in the *image projection function* shown in Equation 2.7.

$$\begin{aligned} h_{pix}(\boldsymbol{p}_{norm}) &= \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \end{aligned} \tag{2.7}$$

This matrix is known as the *intrinsic parameters* matrix, and is referred to by $\boldsymbol{K}$.

Lastly, we account for any distortion present in the lens of the camera. The *distortion function* (Equation 2.8) accounts for this by warping the 'true' pixel locations according to a

set of distortion parameters $(k_1, k_2, k_3, p_1, p_2)$:

$$\begin{aligned} h_{dist}(\boldsymbol{p}_{pix}) &= \begin{bmatrix} u' \\ v' \end{bmatrix} \\ &= (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} 2p_1 uv + p_2(r^2 + 2u^2) \\ p_1(r^2 + 2v^2) + 2p_2 uv \end{bmatrix} \end{aligned} \tag{2.8}$$

Having these parameters is extremely important in getting accurate visual measurements. Even if the manufacturer of a camera provides them, it is generally worth calibrating the camera yourself using a software like OpenCV [7] or Kalibr `https://github.com/ethz-asl/kalibr`. Examples of distortion effects are shown below in figure 2.7.

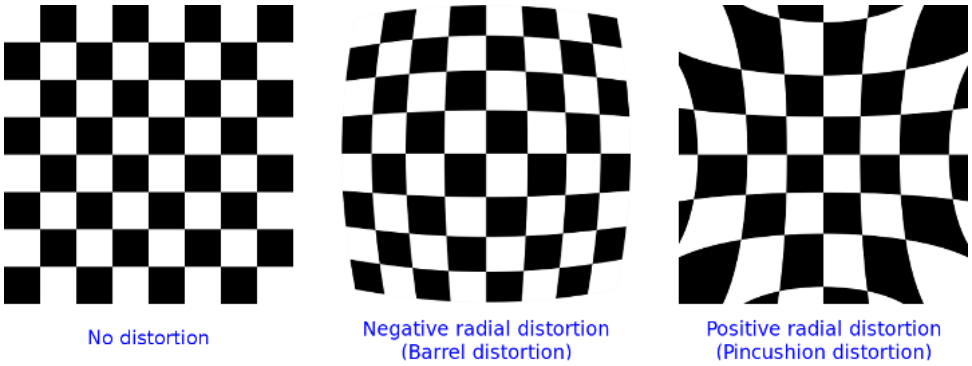

Figure 2.7: Distortion effects in an image, reproduced from `https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html`

The complete measurement model is then given by Equation 2.9.

$$\begin{aligned} \pi({}^C\boldsymbol{p}_f) &= \begin{bmatrix} u' \\ v' \end{bmatrix} \\ &= h_{dist}(h_{pix}(h_{norm}({}^C\boldsymbol{p}_f)) \end{aligned} \tag{2.9}$$

Generally, we work in either undistorted pixel coordinates $(u, v)^T$ or normalized coordinates $(x, y)^T$. All measurements from the camera are in distorted pixel coordinates, so pixel measurements must be undistorted by inverting 2.8 and then optionally normalized by inverting

2.7. This allows for simplicity and efficiency when comparing measured feature locations $\boldsymbol{y}$ to predicted feature locations $h({}^{C}\boldsymbol{p}_f)$.

2.2.2 Triangulation

As established earlier, an observation of a point from a camera is a measurement of the relative bearing between the camera and point. This measurement contains no range data, and thus a single observation doesn't define the position of the point in space, only a vector it must lie on. Multiple observations then provide multiple bearing vectors, one each between the camera and point at every observation. If the position and attitude of the camera during each observation is known, the position of the point can be calculated by estimating the point of intersection for these bearing vectors (shown below in figure 2.8). This process is known as triangulation.

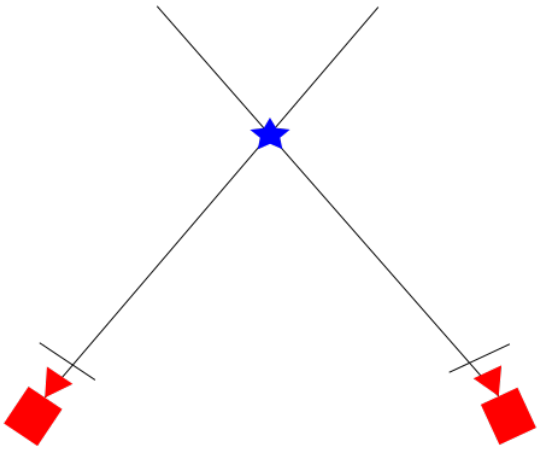

Figure 2.8: A 2D View of Triangulation

The simplest form of triangulation involves two observations from two known cameras. We assume the following information is known:

- $\boldsymbol{y}_1$: The $(x_1, y_1)^T$ observation of the point from the first camera pose

- $\boldsymbol{y}_2$: The $(x_2, y_2)^T$ observation of the point from the second camera pose
- ${}^2_1\boldsymbol{C}$: The rotation from the first camera pose to the second camera pose
- ${}^2\boldsymbol{t}_{2\to1}$: The vector from the second pose to the first expressed in the frame of the second

Using this information, we can form Equation 2.10.

$$ {}^2\boldsymbol{P}_f = {}^2_1\boldsymbol{C}\,{}^1\boldsymbol{P}_f + {}^2\boldsymbol{t}_{2\to1} \tag{2.10} $$

We replace ${}^1\boldsymbol{P}_f$ and ${}^2\boldsymbol{P}_f$ with their homogeneous equivalents:

$$ {}^1\boldsymbol{P}_f = \lambda_1 \boldsymbol{z}_1 $$
$$ {}^2\boldsymbol{P}_f = \lambda_2 \boldsymbol{z}_2 $$

where

$$ \boldsymbol{z}_1 = \begin{bmatrix} \boldsymbol{y}_1 \\ 1 \end{bmatrix} $$
$$ \boldsymbol{z}_2 = \begin{bmatrix} \boldsymbol{y}_2 \\ 1 \end{bmatrix} $$

resulting in the homogenenous form of Equation 2.10.

$$ \lambda_2 \boldsymbol{z}_2 = {}^2_1\boldsymbol{C}\lambda_1 \boldsymbol{z}_1 + {}^2\boldsymbol{t}_{2\to1}. $$

The position of the point in the frame of the first observation is found by arranging this equation into a linear system and solving for λ_1. The linear system is shown in Equation 2.11.

$$ \begin{bmatrix} -{}^2_1\boldsymbol{C}\boldsymbol{z}_1 & \boldsymbol{z_2} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = {}^2\boldsymbol{t}_{2\to1} \tag{2.11} $$
$$ {}^1\boldsymbol{P}_f = \lambda_1 \boldsymbol{z}_1 $$

This problem can be extended to situations in which more than two observations by expanding the linear system to include additional terms or by using the methods described in the appendix of [37].

2.2.3 Feature Detection

Having discussed methods by which an observed landmark can be estimated over sequences of observations, we now describe how exactly a sequence of observations is obtained. Biological eyes recognize and follow semantically-informed environmental features such as "road sign" or "building." Lacking this complex visual processing, visual SLAM system instead detect and track *features* within an image. These features are regions of the image determined to be repeatably detectable such that observations across images can be connected together reliably. Algorithms which find and extract these features are known as *feature detectors*.

A representative example of a feature detector is the *Harris* corner detector[21]. This algorithm stems from the observation that unique and repeatable features are those which are noticably different from their surroundings. The Harris detector searches for these features by finding patches in an image where the difference between the patch and surrounding patches is very large. Given a window W centered at (u, v), the sum of squared differences between the it and a patch of size $(2x + 1, 2y + 1)$ shifted by $(\delta u, \delta v)$ is given in Equation 2.12.

$$E(\delta u, \delta v) = \sum_{u,v \in W} (I(u + \delta u, v + \delta v) - I(u, v))^2 \tag{2.12}$$

This value E, called the *energy* of a feature, is large when the feature at the center of the patch is very different from the rest of the pixels in the patch. Searching for pixel locations $(u, v)^T$ which maximize E will find the locations of good features. This search can be performed efficiently by the approximation of $E(u, v)$ shown in Equation 2.13.

$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} \left(\sum_{x,y \in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.13}$$

The *Harris response* or *Harris score* $R = \det(M) - 0.05 * tr(M)^2$ is then the measure of feature strength. The matrix M is known as the *structure tensor* and is computed as shown in Equation 2.14.

$$\boldsymbol{M} = \sum_{x,y \in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} \tag{2.14}$$

Intuitively, this approximation tells us that good features will have large gradients in both vertical and horizontal directions. This accurately describes the appearance of corners in images, leading to the name Harris *corner* detector. Examples of corners detected by this algorithm are shown below in figure 2.9.

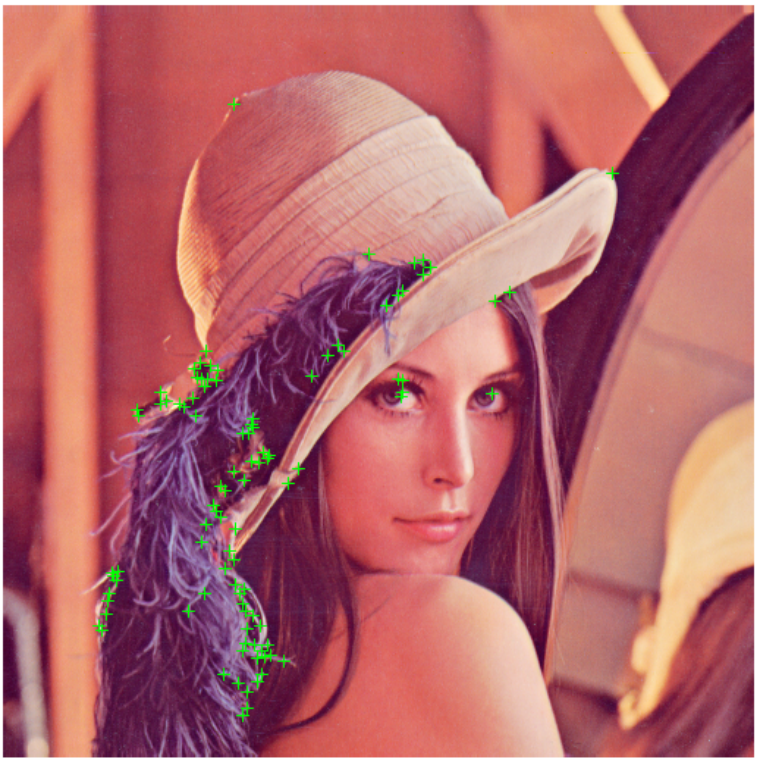

Figure 2.9: Corners detected by the Harris detector

Many feature detectors have been proposed since Harris developed his. Shi-Tomasi [25], SIFT [34], SURF [2], FAST [41], and ORB [42] are the most commonly seen feature detectors today.

2.2.4 Feature Matching

Having detected feature points in a sequence of images, the next task is to connect observations of the same point across this sequence. This task is called the *data association* problem, and is common across all perceptive sensors. There are two common ways of doing this: feature tracking and feature matching.

Feature tracking is the act of taking a keypoint from an image and predicting its location in the next. The standard method for this task is the *Lucas-Kanade* method [35] which performs an iterative optimization process to minimize the difference between the patch surrounding a feature in the first image and a nearby patch in the next image. This is performed via a course-to-fine optimiziation scheme [6], where solutions are calculated at progressively finer and finer scales until convergence. The function to be optimized is given in Equation 2.15

$$\sum_{i=u-x}^{u+x} \sum_{j=v-y}^{v+y} (I(u_+i, v+j) - T(u,v))^2 \tag{2.15}$$

where T is the patch surrounding the feature in the previous image and (i, j) is the displacement that locates the feature in the next image. This optimization problem is solved by either gradient descent or Gauss-Newton. The Lucas-Kanade method is an example of an *optical flow* algorithm, a collection of algorithms which estimate pixel motion through video. This is the method we select for this work. An example of tracking features across images is shown below in figure 2.10, where Harris corners are detected in the first image and tracked into the second using the Lucas-Kanade method.

Figure 2.10: Tracking Shi-Tomasi corners in images from the Kitti dataset [18]

Alternatively, features can be associated by feature matching. Rather than detecting a feature and then tracking it into a new image, matching seeks to find the most likely pairs between features detected in two images. This is accomplished through the extraction of *descriptors*, mathematical objects that uniquely describe the region of the image surrounding the feature. Features are then matched by finding the most similar descriptors across images. In the simplest case, a descriptor may be the pixel values in a square around the feature and the matching process would involve finding the descriptor in the next image with the most similar values. In other cases, such as contemporary descriptors [34], [2], and [42], these descriptors are carefully crafted algorithms which produce a string of numbers. These strings are then matched by minimizing the $L2$-norm between their vector forms. A survey of descriptors and descriptor matching methods is available in [30].

2.2.5 Outlier Rejection

Despite advancements in algorithms for data association in images, outliers and spurious matches are common. The gold standard method of detecting and rejecting outliers in computer vision is Random Sample Consensus (RANSAC) . RANSAC is a model-fitting method which fits the best possible model to a sample of data by repeatedly fitting models to randomly-sampled subsets and saving the model with the most inliers across the entire dataset. An example of using RANSAC to find a line of best fit is shown below in figure 2.11.

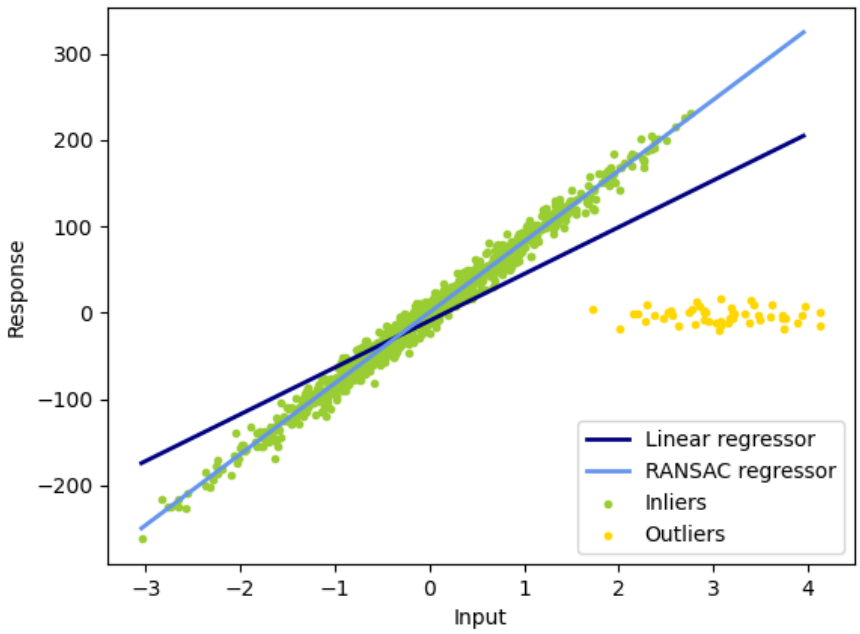

Figure 2.11: RANSAC line fitting in the presence of outliers

In the case of feature matching, the model we use is the *essential matrix*, which describes the mapping of points from one image to another from the camera's rotation and translation between taking the images [22]. A full treatment of the essential matrix is outside the scope of this thesis and interested readers should consider the textbook *Multiple View Geometry* [22] for the necessary background and treatment on the geometry of vision. For now it is enough to state that we use RANSAC to fit an essential matrix to our candidate point pairs, and use this fit to reject outlier pairings. An example of rejecting outliers by fitting an essential matrix is shown below in figure 2.12.

Figure 2.12: Rejecting outliers in Lucas-Kanade tracks with RANSAC

Chapter 3

Parameter Estimation

3.1 Random Variables

A key ability of a navigation system is to not only estimate the states of the system but also the uncertainty in those states. This falls under the umbrella of *probabalistic state estimation*, which requires a basic understanding of probability theory.

A random variable is defined as a variable x whose value falls in a range according to some likelihood function $p(x)$. This function is known as the *probability density function*(pdf). The PDF describes the *probability* of x taking on certain values. The probability $P(x = a)$ is the likelihood that x takes on the value a, ranging from 0 (impossible) to 1 (certain). An example of a PDF is shown below in figure 3.1. The PDF and probability have the following properties:

$$P(a \leq x \leq b) = \int_a^b p(x)dx$$

$$\int_{-\infty}^{\infty} p(x)dx = 1$$

$$p(x) \geq 0 \forall x$$

$$P(x) = 0 \forall x$$

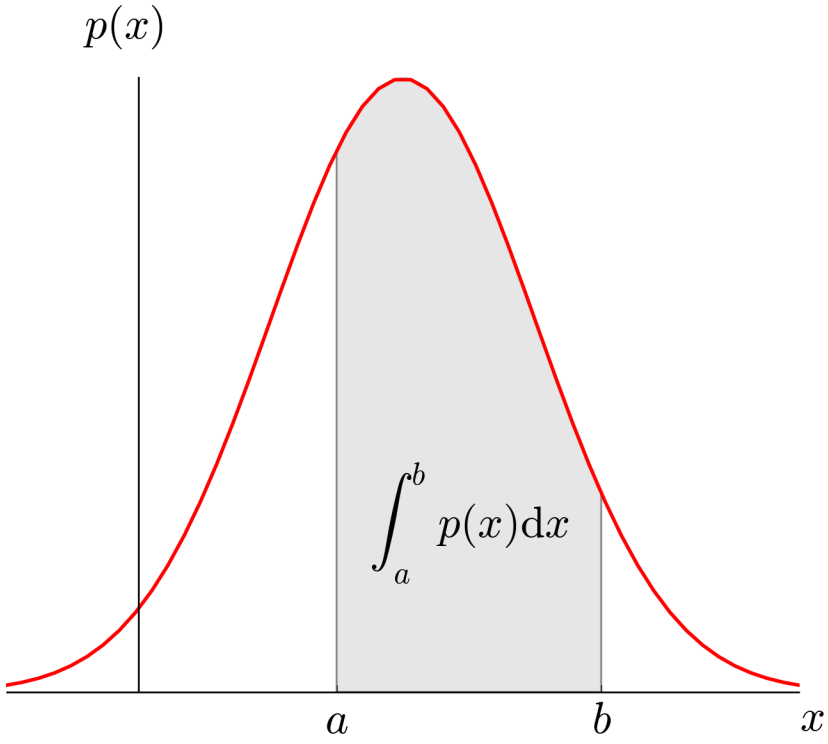

Figure 3.1: A probability density function

The weighted average of all values of x is the *expected value* of x, denoted $E(x)$. The *variance* of x is the weighted average of the squared difference between each value of x and the expected value of x, and intuitively describes how "spread out" x is. These two properties are the most common ways of describing PDFs.

$$E(x) = \int_{-\infty}^{\infty} p(x)dx$$
$$Var(x) = \int_{-\infty}^{\infty} (x - E(x))^2 p(x)dx$$

For our purposes, we can interpret $E(x)$ as the most likely value of x and $Var(x)$ as the uncertainty in that value.

3.1.1 The Gaussian Distribution

The *Gaussian* distribution, also known as the *normal* distribution, is the most common distribution in robotics as sensor noise commonly follows a Gaussian distribution. A Gaussian

one-dimensional random variable is written $x \sim N(\mu, \sigma^2)$. The PDF of a one-dimensional Gaussian distribution is given in Equation 3.1.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{3.1}$$

Examples of Gaussian PDFs are shown below in 3.2.

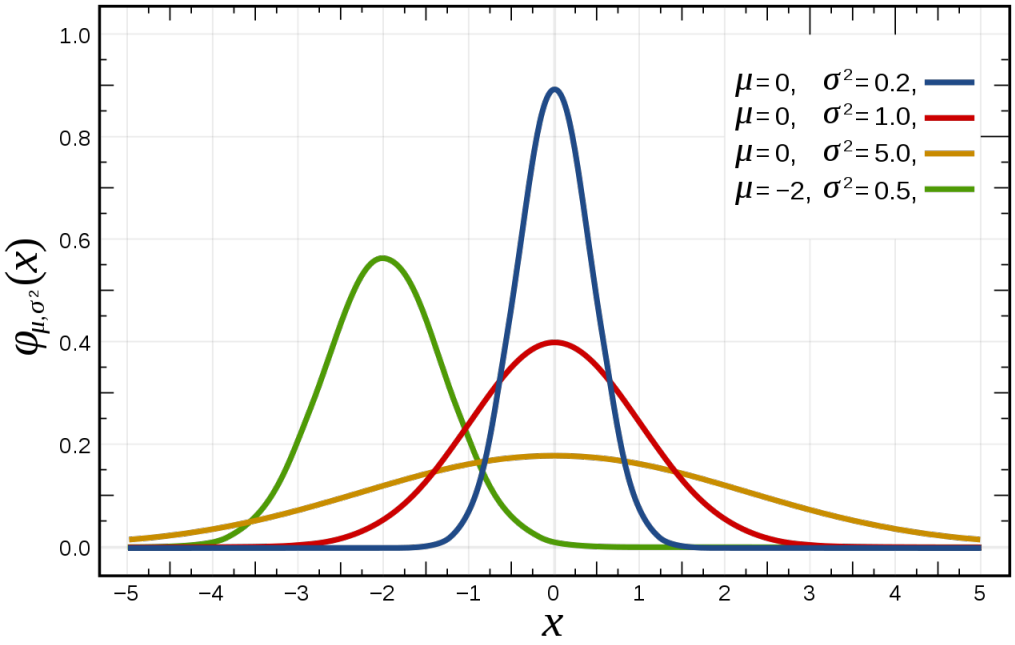

Figure 3.2: A variety of 1D Gaussian distributions, from `https://en.wikipedia.org/wiki/Normal_distribution`

Higher-dimensional Gaussian distributions are a simple extension of the one-dimensional case. In this case, the random variable takes on a vector value rather than a scalar and is written $\boldsymbol{x} \sim N(\boldsymbol{u}, \boldsymbol{\Sigma})$ Here $\boldsymbol{x} = (a, b, c, ...)^T$ is an $n \times 1$ vector of Gaussian variables where $\boldsymbol{\mu}$ is the $n \times 1$ vector of their means and $\boldsymbol{\Sigma}$ is their associated *covariance* matrix. The diagonal entries of this matrix are the variances of each element in $\boldsymbol{x}$: $(\sigma_a^2, \sigma_b^2, \sigma_c^2, ...)$. The off-diagonal elements are the *correlations* between elements: $(\rho_{(a,b)}\sigma_a\sigma_b, \rho_{(a,c)}\sigma_a\sigma_c, ...)$. The structure of this matrix

is given in Equation 3.2.

$$Cov(\boldsymbol{x}) = \begin{bmatrix} \sigma_{x_1}^2 & \rho_{(x_1,x_2)}\sigma_{x_1}\sigma_{x_2} & \cdots & \rho_{(x_1,x_n)}\sigma_{x_1}\sigma_{x_n} \\ \rho_{(x_2,x_1)}\sigma_{x_2}\sigma_{x_1} & \sigma_{x_2}^2 & \cdots & \rho_{(x_2,x_2)}\sigma_{x_2}\sigma_{x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{(x_n,x_1)}\sigma_{x_n}\sigma_{x_1} & \rho_{(x_n,x-2)}\sigma_{x_n}\sigma_{x_2} & \cdots & \sigma_{x_n}^2 \end{bmatrix} \tag{3.2}$$

A useful property of Gaussian distributions is that they remain Gaussian under linear transforms. That is,

- Given $x \sim N(\mu, \sigma^2)$, $y = ax + b \sim N(\mu + b, a^2\sigma^2$
- Given $\boldsymbol{x} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \sim N(\boldsymbol{\mu} + \boldsymbol{b}, \boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^T)$

3.2 Kalman Filtering

3.2.1 The Linear Kalman Filter

The Kalman filter (KF) [28] is the standard approach to recursive estimation for linear Gaussian systems. The KF maintains a state estimate in the form of a Gaussian $N(\hat{\boldsymbol{x}}, \boldsymbol{P})$ which is recursively updated in an alternating *predict - correct* algorithm, where the current $\hat{\boldsymbol{x}}$ and $\boldsymbol{P}$ are propagated forward using the state transition equations and then corrected when a measurement becomes available. These predicted and corrected values, known as *a priori* and *a posteriori* estimates, are denoted $(\hat{\boldsymbol{x}}^-, \boldsymbol{P}^-)$ for predicted and $(\hat{\boldsymbol{x}}^+, \boldsymbol{P}^+)$ for corrected. The KF algorithm is shown in algorithm 1. Linear Gaussian systems are dynamic systems of the form shown in Equation 3.3

$$\begin{aligned} \boldsymbol{x}_k &= \boldsymbol{A}\boldsymbol{x}_{k-1} + \boldsymbol{B}\boldsymbol{u}_{k-1} + \boldsymbol{w}_{k-1} \\ \boldsymbol{z}_k &= \boldsymbol{H}\boldsymbol{x}_k + \boldsymbol{v}_k \end{aligned} \tag{3.3}$$

where $\boldsymbol{x}$ is a vector of state variables, $\boldsymbol{z}$ is a vector of measurements, $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ are matrices describing the system, and $\boldsymbol{w}$ and $\boldsymbol{v}$ are independent noise processes with Gaussian distributions

$$\boldsymbol{w} \sim N(0, \boldsymbol{Q})$$
$$\boldsymbol{v} \sim N(0, \boldsymbol{R}).$$

Algorithm 1: The Kalman Filter

while *measurements are available* **do**

Time Update (Prediction)

Propagate the state mean and covariance forward

$$\hat{\boldsymbol{x}}_k^- = \boldsymbol{A}\hat{\boldsymbol{x}}_{k-1}^+ + \boldsymbol{B}\boldsymbol{u}_{k-1} \tag{3.4}$$
$$\boldsymbol{P}_k^- = \boldsymbol{A}\boldsymbol{P}_{k-1}^+\boldsymbol{A}^T + \boldsymbol{Q} \tag{3.5}$$

Measurement Update (Correction)

Calculate the Kalman gain

$$\boldsymbol{K}_k = \boldsymbol{P}_k^-\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{P}_k^-\boldsymbol{H}^T + \boldsymbol{R})^{-1} \tag{3.6}$$

Update the state mean and covariance

$$\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k(y_k - \boldsymbol{H}\hat{\boldsymbol{x}}_k^-) \tag{3.7}$$
$$\boldsymbol{P}_k^+ = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H})\boldsymbol{P}_k^- \tag{3.8}$$

It is worth noting that Equation 3.8 is usually replaced with the *Joseph form* covariance update shown in Equation 3.9

$$\boldsymbol{P}_k^+ = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H})\boldsymbol{P}_k^-(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H})^T + \boldsymbol{K}_k\boldsymbol{R}\boldsymbol{K}_k^T \tag{3.9}$$

which forces $\boldsymbol{P}^+$ to remain positive-semidefinite in the face of rounding errors.

3.2.2 The Extended Kalman Filter

Linear systems, while simple to work with, are rare in the navigation world. Instead the system to be estimated will often have nonlinear dynamic and measurement models. The standard

form of a driven nonlinear system is given in Equation 3.10

$$\begin{aligned} \boldsymbol{x}_k &= f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, \boldsymbol{w}_{k-1}) \\ \boldsymbol{z}_k &= h(\boldsymbol{x}_k, \boldsymbol{v}_k) \end{aligned} \tag{3.10}$$

where $\boldsymbol{x}$, $\boldsymbol{z}$, $\boldsymbol{w}$, and $\boldsymbol{v}$ are defined as before. The KF can be extended to these nonlinear systems by taking advantage of the Taylor series function approximation shown in Equation 3.11.

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)(x - x_0)^2}{2!} + h.o.t., \tag{3.11}$$

By taking the first two terms a nonlinear function can be approximated as locally linear if the function is continuously differentiable. The KF can then be extended to nonlinear state transition and measurement functions by taking the linear Taylor approximation about the current best state estimate, resulting in the linearized state equations shown in Equation 3.12

$$\begin{aligned} \boldsymbol{x}_k &\approx f(\hat{\boldsymbol{x}}_{k-1}^{+}, \boldsymbol{u}_k, 0) + \boldsymbol{F}_k(\boldsymbol{x}_{k-1} - \hat{\boldsymbol{x}}_{k-1}^{+}) + \boldsymbol{G}_k \boldsymbol{w}_{k-1} \\ \boldsymbol{z}_k &\approx h(\hat{\boldsymbol{x}}_k^{-}, 0) + \boldsymbol{H}_k(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k^{-}) + \boldsymbol{V}_k \boldsymbol{v}_k \end{aligned} \tag{3.12}$$

where the following matrices are defined:

$$\begin{aligned} \boldsymbol{F}_k &= \frac{\partial f}{\partial x}|_{\hat{x}_{k-1}^{+}, \boldsymbol{u}_{k-1}, 0} \\ \boldsymbol{G}_k &= \frac{\partial f}{\partial w}|_{\hat{x}_{k-1}^{+}, \boldsymbol{u}_{k-1}, 0} \\ \boldsymbol{H}_k &= \frac{\partial h}{\partial x}|_{\hat{x}_k^{-}, 0} \\ \boldsymbol{V}_k &= \frac{\partial h}{\partial v}|_{\hat{x}_k^{-}, 0} \end{aligned}$$

as the *system Jacobians*. These Jacobians can either be derived from the discrete difference equations or from numerically integrating the continuous equations. Of note, $\boldsymbol{G}_k$ and especially $\boldsymbol{V}_k$ are often identity matrices of the appropriate size. This new estimator, called the *Extended Kalman Filter*, results in the algorithm shown in algorithm 2.

Algorithm 2: The Extended Kalman Filter

while *measurements are available* **do**

Time Update (Prediction)

Linearize the dynamic equations about the current state estimate

$$\boldsymbol{F}_k = \frac{\partial f}{\partial x}|_{\hat{x}^+_{k-1}, \boldsymbol{u}_{k-1}, 0} \tag{3.13}$$

$$\boldsymbol{G}_k = \frac{\partial f}{\partial w}|_{\hat{x}^+_{k-1}, \boldsymbol{u}_{k-1}, 0} \tag{3.14}$$

Propagate the state mean and covariance forward

$$\hat{\boldsymbol{x}}^-_k = f(\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{u}_{k-1}, 0) \tag{3.15}$$

$$\boldsymbol{P}^-_k = \boldsymbol{F}_k \boldsymbol{P}^+_{k-1} \boldsymbol{F}^T_k + \boldsymbol{G}_k \boldsymbol{Q} \boldsymbol{G}^T_k \tag{3.16}$$

Measurement Update (Correction)

Linearize the measurement equations about the predicted state estimate

$$\boldsymbol{H}_k = \frac{\partial h}{\partial x}|_{\hat{x}^-_k, 0} \tag{3.17}$$

$$\boldsymbol{V}_k = \frac{\partial h}{\partial v}|_{\hat{x}^-_k, 0} \tag{3.18}$$

Calculate the Kalman gain

$$\boldsymbol{K}_k = \boldsymbol{P}^-_k \boldsymbol{H}^T_k (\boldsymbol{H}_k \boldsymbol{P}^-_k \boldsymbol{H}^T_k + \boldsymbol{V}_k \boldsymbol{R} \boldsymbol{V}^T_k)^{-1} \tag{3.19}$$

Update the state mean and covariance

$$\Delta \boldsymbol{x} = \boldsymbol{K}_k (\boldsymbol{z}_k - h(\hat{\boldsymbol{x}}^-_k, 0)) \tag{3.20}$$

$$\hat{\boldsymbol{x}}^+_k = \hat{\boldsymbol{x}}^-_k \boxplus \Delta \boldsymbol{x} \tag{3.21}$$

$$\boldsymbol{P}^+_k = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}^-_k \tag{3.22}$$

The operator $\boxplus$ is used in equation 3.21 to note that, while the KF is meant to operate within a vector space under standard vector addition, many nonlinear states (such as rotations) do not generally compose through vector addition and will often require a more involved operation. Thus $\boxplus$ is used to denote generic composition. It is extremely important that the actual nonlinear equations are used in 3.15 and 3.20. Again, the Joseph form covariance update is generally used instead of 3.22.

3.2.3 The Error State Kalman Filter

A common modification of the EKF is to estimate the INS errors rather than the INS states. This adjustment comes from the idea that the error terms will be smaller in magnitude and closer to the operating point than the states themselves, resulting in a more accurate linearization. A nominal INS state $\hat{\boldsymbol{x}}$ is maintained by propagating the state transition equations and is occasionally corrected by injecting the estimated error state $\delta\boldsymbol{x}$ defined by:

$$\boldsymbol{x}_{true} = \hat{\boldsymbol{x}} + \delta\boldsymbol{x}$$

To this end, it is intuitive to reframe the system equations to include the error state. The new form of the nonlinear system is given in Equation 3.23.

$$\begin{aligned} \boldsymbol{x}_k &= f(\hat{\boldsymbol{x}}_{k-1} + \delta\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, \boldsymbol{w}_{k-1}) \\ \boldsymbol{z}_k &= h(\hat{\boldsymbol{x}}_k + \delta\boldsymbol{x}_k, \boldsymbol{v}_k) \end{aligned} \tag{3.23}$$

This formulation then requires that Jacobians are calculated with respect to this error state, and are given as:

$$\begin{aligned} \boldsymbol{F}_k &= \frac{\partial f}{\partial \delta x}|_{\hat{x}^+_{k-1}, \boldsymbol{u}_{k-1}, 0} \\ \boldsymbol{G}_k &= \frac{\partial f}{\partial w}|_{\hat{x}^+_{k-1}, \boldsymbol{u}_{k-1}, 0} \\ \boldsymbol{H}_k &= \frac{\partial h}{\partial \delta x}|_{\hat{x}^-_k, 0} \\ \boldsymbol{V}_k &= \frac{\partial h}{\partial v}|_{\hat{x}^-_k, 0} \end{aligned}$$

These Jacobians are then used as in the standard EKF. This modification results in the *Error State Extended Kalman Filter*, shown in algorithm 3.

An important distinction between the EKF and EsKF is the need to inject the estimated error-state into the nominal state to get the current best estimate. This injection has two components: Correcting the nominal state and resetting the error state. This process is given in Equation 3.24.

$$\begin{aligned} \hat{\boldsymbol{x}} &= \hat{\boldsymbol{x}} \boxplus \delta \boldsymbol{x} \\ \delta \boldsymbol{x} &= 0 \end{aligned} \tag{3.24}$$

No correction of the covariance is needed as the error state simply represents a shift and rotation from the nominal state, neither of which transform the covariance. Intuitively, a variance of 1 meter in the estimated error in a state is identical to a variance of 1 meter in the estimate of the state.

3.3 Nonlinear Optimization

Optimization problems are problems which seek parameters $\boldsymbol{\theta}$ to minimize or maximize an objective function. These problems are posed as $\underset{\boldsymbol{\theta}}{\operatorname{argmin}} f(\boldsymbol{\theta})$ or $\underset{\boldsymbol{\theta}}{\operatorname{argmax}} f(\boldsymbol{\theta})$, respectively. Nonlinear optimization problems are the subset of optimization problems for which $f(\boldsymbol{\theta})$ is nonlinear. Specifically, we address nonlinear *least squares* (NLS) problems, which are of the form $\underset{\boldsymbol{\theta}}{\operatorname{argmin}} ||f(\boldsymbol{\theta})||^2$ and seek to minimize the square of the cost function f. This cost function is generally an error function such as the measurement residual function $e(\boldsymbol{\theta}) = \boldsymbol{y} - h(\boldsymbol{\theta})$, and can be written as a sum of individual squared errors as shown in Equation 3.38.

$$\begin{aligned} &\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_i (\boldsymbol{y}_i - h_i(\boldsymbol{\theta}))^2 \\ &\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_i \boldsymbol{e}_i^T(\boldsymbol{\theta}) \boldsymbol{e}_i(\boldsymbol{\theta}) \end{aligned} \tag{3.38}$$

These individual errors can be weighted by their certainty, represented by their covariance $\boldsymbol{\Sigma}_i$, producing *weighted* nonlinear least-squares problems. The classic NLS problem is given in

Algorithm 3: The Error State Extended Kalman Filter

while *measurements are available* **do**

Time Update (Prediction)

Linearize the dynamic equations about the current state estimate

$$\boldsymbol{F}_k = \frac{\partial f}{\partial \delta x}|_{\hat{x}_{k-1}^+, \boldsymbol{u}_{k-1}, 0} \tag{3.25}$$

$$\boldsymbol{G}_k = \frac{\partial f}{\partial w}|_{\hat{x}_{k-1}^+, \boldsymbol{u}_{k-1}, 0} \tag{3.26}$$

Propagate the nominal state, error state, and error covariance forward

$$\hat{\boldsymbol{x}}_k^- = f(\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{u}_{k-1}, 0) \tag{3.27}$$

$$\delta\hat{\boldsymbol{x}}_k^- = \boldsymbol{F}_k \delta \boldsymbol{x}_{k-1}^+ \tag{3.28}$$

$$\boldsymbol{P}_k^- = \boldsymbol{F}_k \boldsymbol{P}_{k-1}^+ \boldsymbol{F}_k^T + \boldsymbol{G}_k \boldsymbol{Q} \boldsymbol{G}_k^T \tag{3.29}$$

Measurement Update (Correction)

Linearize the measurement equations about the predicted state estimate

$$\boldsymbol{H}_k = \frac{\partial h}{\partial \delta x}|_{\hat{x}_k^-, 0} \tag{3.30}$$

$$\boldsymbol{V}_k = \frac{\partial h}{\partial v}|_{\hat{x}_k^-, 0} \tag{3.31}$$

Calculate the Kalman gain

$$\boldsymbol{K}_k = \boldsymbol{P}_k^- \boldsymbol{H}_k^T (\boldsymbol{H}_k \boldsymbol{P}_k^- \boldsymbol{H}_k^T + \boldsymbol{V}_k \boldsymbol{R} \boldsymbol{V}_k^T)^{-1} \tag{3.32}$$

Update the error-state mean and covariance

$$\Delta\delta\boldsymbol{x} = \boldsymbol{K}_k(\boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_k^-, 0)) \tag{3.33}$$

$$\delta\hat{\boldsymbol{x}}_k^+ = \delta\hat{\boldsymbol{x}}_k^- + \Delta\delta\boldsymbol{x} \tag{3.34}$$

$$\boldsymbol{P}_k^+ = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k)\boldsymbol{P}_k^- \tag{3.35}$$

Inject error correction into nominal state and reset error state

$$\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^- \boxplus \delta\hat{\boldsymbol{x}}_k^+ \tag{3.36}$$

$$\delta\hat{\boldsymbol{x}}_k^+ = 0 \tag{3.37}$$

Equation 3.39.

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_i \boldsymbol{e}_i^T(\boldsymbol{\theta})\boldsymbol{\Sigma}^{-1}\boldsymbol{e}_i(\boldsymbol{\theta}) \tag{3.39}$$

These problems are typically solved by iteratively calculating the optimal *change* $\delta\boldsymbol{\theta}$ to the current estimate of $\boldsymbol{\theta}$ to minimize the cost function until the calculated change is suitably small. Here we present the Gauss-Newton method, the most common method of solving such problems.

3.3.1 Gauss-Newton

Gauss-Newton is an efficient iterative nonlinear solver derived from the 2^{nd}-order Taylor series approximation of the cost function. This approximation is shown in Equation 3.40

$$f(\boldsymbol{\theta} + \delta\boldsymbol{\theta}) \approx f(\boldsymbol{\theta}) + \boldsymbol{J}\delta\boldsymbol{\theta} + \frac{1}{2}\delta\boldsymbol{\theta}^T\boldsymbol{H}\delta\boldsymbol{\theta} \tag{3.40}$$

where

$$\boldsymbol{J} = \frac{\partial f(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}|_{\boldsymbol{\theta}}$$
$$\boldsymbol{H} = \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}\partial \boldsymbol{\theta}^T}|_{\boldsymbol{\theta}}$$

are the Jacobian and Hessian of f calculated at $\boldsymbol{\theta}$. Noticing that in the case of a quadratic objective function ($f(\boldsymbol{\theta}) \approx \frac{1}{2}\boldsymbol{e}(\boldsymbol{\theta})^T\boldsymbol{\Sigma}^{-1}\boldsymbol{e}(\boldsymbol{\theta})$ for some vector-valued function $\boldsymbol{e}(\boldsymbol{\theta})$, e.g. a least-squares formulation), near $\boldsymbol{\theta}_0$ the Jacobian can be approximated as $\boldsymbol{J} \approx \boldsymbol{e}(\boldsymbol{\theta}_0)^T\boldsymbol{\Sigma}^{-1}\boldsymbol{J}_e$ where

$$\boldsymbol{J}_e = \left(\frac{\partial e(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}|_{\boldsymbol{\theta}_0}\right)$$

and the Hessian can be approximated as $\boldsymbol{H} \approx \boldsymbol{J}_e^T\boldsymbol{\Sigma}^{-1}\boldsymbol{J}_e$. These approximations can then be substituted into the Taylor approximation. The value of $\delta\boldsymbol{\theta}$ that minimizes f can be found by

setting the $\frac{\partial f}{\partial \delta \boldsymbol{\theta}} = 0$ and solving:

$$\begin{aligned}
\frac{\partial f}{\partial \delta \boldsymbol{\theta}} &= 0 \\
&= \boldsymbol{J} + \delta \boldsymbol{\theta}^T \boldsymbol{H} \\
\delta \boldsymbol{\theta}^T \boldsymbol{H} &= -\boldsymbol{J} \\
\boldsymbol{H}^T \delta \boldsymbol{\theta} &= -\boldsymbol{J}^T \\
(\boldsymbol{H}^T)^{-1} \boldsymbol{H}^T \delta \boldsymbol{\theta} &= -(\boldsymbol{H}^T)^{-1} \boldsymbol{J}^T \\
\delta \boldsymbol{\theta} &= -(\boldsymbol{H}^T)^{-1} \boldsymbol{J}^T
\end{aligned}$$

which simplifies[1] to the Gauss-Newton correction given in Equation 3.41.

$$\delta \boldsymbol{\theta} = -\boldsymbol{H}^{-1} \boldsymbol{J}^T \tag{3.41}$$

The new best estimate for $\boldsymbol{\theta}$ is updated to be $\boldsymbol{\theta} + \delta \boldsymbol{\theta}$ and the process is repeated until convergence. The combination of these steps is shown in Algorithm 4 below.

Algorithm 4: Gauss-Newton Method for Optimization

while *not converged* **do**

Calculate Jacobian and Hessian

$$\boldsymbol{J} \approx \boldsymbol{e}(\boldsymbol{\theta}_k)^T \boldsymbol{\Sigma}^{-1} \left(\frac{\partial e(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} |_{\boldsymbol{\theta}_k} \right) \tag{3.42}$$

$$\boldsymbol{H} \approx \left(\frac{\partial e(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} |_{\boldsymbol{\theta}_k} \right)^T \boldsymbol{\Sigma}^{-1} \left(\frac{\partial e(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} |_{\boldsymbol{\theta}_k} \right) \tag{3.43}$$

Solve for Update

$$\delta \boldsymbol{\theta} = -\boldsymbol{H}^{-1} \boldsymbol{J}^T \tag{3.44}$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \delta \boldsymbol{\theta} \tag{3.45}$$

[1] As the Hessian is symmetric, we can ignore transpositions of it

Chapter 4

Visual-Inertial Navigation

This chapter develops the dynamic and measurement models used in visual-inertial estimation and presents two VINS formulations: the MSCKF and a batch NLS VINS. While this is intended to be a reasonably comprehensive introduction to the world of visual-inertial estimation, there are a number of methods we ignore here (loosely-coupled, deep learning, etc). Readers interested in these less common methods should look to surveys such as [11].

4.1 Fundamentals

We begin by developing a common treatment of the IMU and camera sensors shared by all visual-inertial estimators. While their pre- and post-processing of the information received from these sensors is different, all of them will use the IMU to propagate the system forward using a dynamic model and the camera to provide measurements of the system's motion and environment using a measurement model.

4.1.1 Dynamic Model

The evolving INS state is described by the tuple $({}^{G}\boldsymbol{p}_I, {}^{G}\boldsymbol{v}_I, {}^{G}_{I}\boldsymbol{q}, \boldsymbol{b}_a, \boldsymbol{b}_g)$ where ${}^{G}\boldsymbol{p}_I$ is the position of the INS in the global frame, ${}^{G}\boldsymbol{v}_I$ is the velocity of the INS in the global frame, ${}^{G}_{I}\boldsymbol{q}$ is the Hamilton quaternion[1] representing the rotation from the INS frame to the global frame, $\boldsymbol{b}_a$ is the accelerometer bias, and $\boldsymbol{b}_g$ is the gyroscope bias. The dynamic equations for the system are

[1]The quaternion attitude representation is explained in more detail in appendix A. It is of utmost importance that quaternion conventions are kept track of as they can be rather confusing.

given in Equation 4.1.

$$
\begin{aligned}
{}^{G}\dot{\boldsymbol{p}}_I &= {}^{G}\boldsymbol{v}_I \\
{}^{G}\dot{\boldsymbol{v}}_I &= \boldsymbol{C}({}^{G}_{I}\boldsymbol{q})\,{}^{B}\boldsymbol{a} + {}^{G}\boldsymbol{g} \\
{}^{G}_{I}\dot{\boldsymbol{q}} &= \frac{1}{2}\,{}^{G}_{I}\boldsymbol{q} \otimes {}^{B}\boldsymbol{\omega} \\
&= \frac{1}{2}\Omega({}^{B}\boldsymbol{\omega})\,{}^{G}_{I}\boldsymbol{q} \\
\dot{\boldsymbol{b}}_a &= \boldsymbol{w}_{wa} \\
\dot{\boldsymbol{b}}_g &= \boldsymbol{w}_{wg}
\end{aligned}
\tag{4.1}
$$

where

$$
\boldsymbol{\Omega}(\boldsymbol{\omega}) = \begin{bmatrix} 0 & -\boldsymbol{\omega}^T \\ \boldsymbol{\omega} & -[\boldsymbol{\omega}\times] \end{bmatrix}
$$

$$
[\boldsymbol{\omega}\times] = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}.
$$

Using equations 2.2 and 2.3, we can reformulate 4.1 into a nonlinear system driven by IMU measurements. This formulation is as follows:

$$
\begin{aligned}
\dot{\boldsymbol{x}} &= f_I(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{w}) \\
&= f\Bigg(\begin{bmatrix} {}^{G}\boldsymbol{p}_I \\ {}^{G}\boldsymbol{v}_I \\ {}^{G}_{I}\boldsymbol{q} \\ \boldsymbol{b}_a \\ \boldsymbol{b}_g \end{bmatrix}, \begin{bmatrix} \boldsymbol{a}_m \\ \boldsymbol{\omega}_m \end{bmatrix}, \begin{bmatrix} \boldsymbol{w}_a \\ \boldsymbol{w}_{wa} \\ \boldsymbol{w}_g \\ \boldsymbol{w}_{wg} \end{bmatrix} \Bigg)
\end{aligned}
$$

which, after applying the expectation operator and noticing that the modeled errors in the system are zero-mean Gaussians, results in the INS state propagation function shown in Equation

4.2.

$$\dot{\hat{\boldsymbol{x}}} = f_I(\hat{\boldsymbol{x}}, \boldsymbol{u}, 0)$$

$$\begin{bmatrix} {}^G\dot{\hat{\boldsymbol{p}}}_I \\ {}^G\dot{\hat{\boldsymbol{v}}}_I \\ {}^G_I\dot{\hat{\boldsymbol{q}}} \\ \dot{\hat{\boldsymbol{b}}}_a \\ \dot{\hat{\boldsymbol{b}}}_g \end{bmatrix} = \begin{bmatrix} {}^G\hat{\boldsymbol{v}}_I \\ \boldsymbol{C}({}^G_I\hat{\boldsymbol{q}})(\boldsymbol{a}_m - \hat{\boldsymbol{b}}_a) + {}^G\boldsymbol{g} \\ \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}_m - \hat{\boldsymbol{b}}_g)\,{}^G_I\hat{\boldsymbol{q}} \\ \boldsymbol{0}_{3\times 1} \\ \boldsymbol{0}_{3\times 1} \end{bmatrix}. \tag{4.2}$$

See appendix B for details on numerical integration of dynamic equations. In the future, we will refer to the discrete-time version of this function as $f_k(\boldsymbol{x}_k, \boldsymbol{u}_k, 0)$.

4.1.2 Measurement Model

The measurement function used is the perspective projection function defined in section 2.2, which maps a point in the world onto the idealized image plane. Here we extend this function to a camera-IMU system for the purpose of measuring features relative to the estimated IMU state.

We begin with the perspective projection function, equation 2.4:

$$\boldsymbol{z} = h({}^C\boldsymbol{p}_f) + \boldsymbol{n}$$

where

$${}^C\boldsymbol{p}_f = {}^C_G\boldsymbol{C}({}^G\boldsymbol{p}_f - {}^G\boldsymbol{p}_C).$$

Using this function for a camera attached to an IMU requires the *extrinsic calibration* $({}^{I}\boldsymbol{p}_{C}, {}^{C}_{I}\boldsymbol{q})$ between the camera and IMU. Using this calibration, the camera position and orientation can be computed as shown in Equation 4.3.

$$\begin{aligned} {}^{C}_{G}\boldsymbol{C} &= \boldsymbol{C}({}^{C}_{I}\boldsymbol{q})\boldsymbol{C}({}^{G}_{I}\boldsymbol{q})^{T} \\ {}^{G}\boldsymbol{p}_{C} &= {}^{G}\boldsymbol{p}_{I} + \boldsymbol{C}({}^{G}_{I}\boldsymbol{q})\,{}^{I}\boldsymbol{p}_{C}. \end{aligned} \tag{4.3}$$

This now allows the computation of a predicted measurement $\hat{\boldsymbol{z}}$ from the estimated IMU state and global feature position as shown in Equation 4.4

$$\begin{aligned} \boldsymbol{z} &= h({}^{G}\boldsymbol{p}_{I}, {}^{G}_{I}\boldsymbol{q}, {}^{G}\boldsymbol{p}_{f}, {}^{C}_{I}\boldsymbol{C}, {}^{I}\boldsymbol{p}_{C}) \\ &= \frac{1}{{}^{C}z_{l}} \begin{bmatrix} {}^{C}x_{l} \\ {}^{C}y_{l} \end{bmatrix} \end{aligned} \tag{4.4}$$

where ${}^{C}_{I}\boldsymbol{C}$ is the rotation matrix which rotates vectors from the IMU frame to the camera frame and ${}^{I}\boldsymbol{p}_{C}$ is the position of the camera in the IMU frame. These two elements together are referred to as the *extrinsic calibration* of the camera-IMU system and are considered to be known perfectly in this work. In this case, this function can be referred to as $h(\boldsymbol{x}_{I}, {}^{G}\boldsymbol{p}_{f})$.

4.2 A Filtering Approach to VINS: The Multi-State Constraint Kalman Filter

To demonstrate the filtering approach to VINS we present the most representative filtering VINS work: the Multi-State Constraint Kalman Filter (MSCKF). Mourikis and Roumeliotis [37] address the flaws of previous EKF-based VINS by moving away from the standard EKF-SLAM formulation of VINS to a stochastic-cloning based structureless VIO approach. In contrast to the EKF-SLAM VINS formulation, which maintains estimates of the current pose and any landmarks currently in sight, the MSCKF instead maintains a sliding window of past INS poses. This estimator structure is shown in4.1, with INS poses at previous time steps shown in red and observed landmarks shown in blue. Gray ellipses enclose objects which are maintained in the EKF state vector.

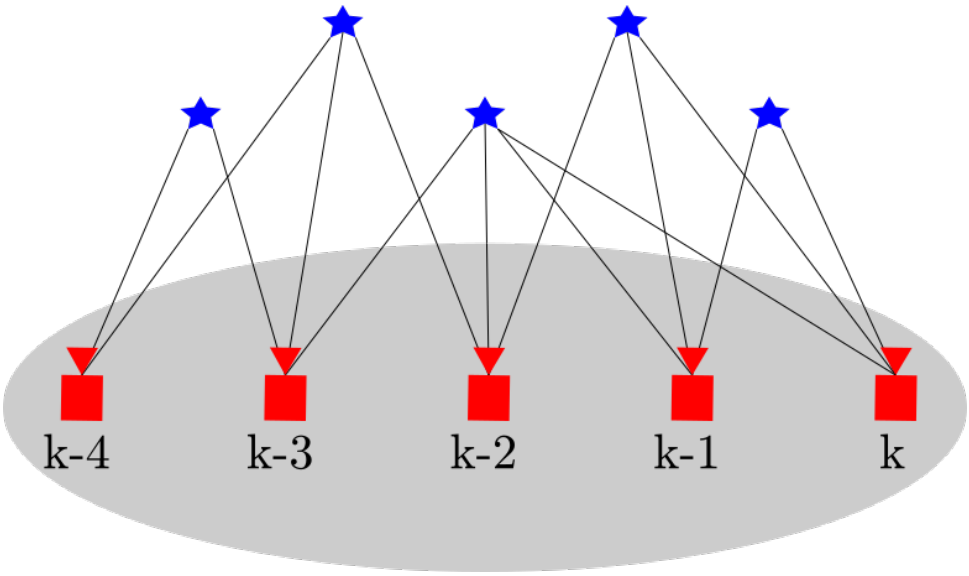

Figure 4.1: Structure of the MSCKF

The MSCKF then does not maintain estimates of observed landmarks at all. Instead, the processing of observations is delayed. As features fall out of view or reach a maximum number of observations, they are triangulated using the window states they are observed from. This triangulated position is then used to form geometric constraints between the feature and the window states from which it is observed which are used for EKF updates. By waiting to process features until the last possible moment, the MSCKF only linearizes the measurement function once per feature. This results in a more accurate feature position estimate and reduces the information lost through the continued linearization process of EKF-SLAM.

4.2.1 Filter Definition

The MSCKF nominal state vector is defined as:

$$\boldsymbol{x} = (\boldsymbol{x}_I, \boldsymbol{\xi}_{k-w}, \boldsymbol{\xi}_{k-(w-1)}, ..., \boldsymbol{\xi}_{k-1})$$

where $\boldsymbol{x}_I$ is the INS state defined in 4.1.1 and $\boldsymbol{\xi}_{k-n}$ is a copy of the INS pose n time steps ago, a modification used in [33]. It is **extremely** important to note that their original implementation uses JPL quaternions. Here we rederive it with Hamilton quaternions to match the convention of the rest of this work.

We define a pose $\boldsymbol{\xi} \in SO(3)$ to be the following vector representing the position and orientation of an INS:

$$\boldsymbol{\xi} = \begin{bmatrix} {}^{G}\boldsymbol{p}_I \\ {}^{G}_{I}\boldsymbol{q} \end{bmatrix}$$

These past poses are then of the form:

$$\boldsymbol{\xi}_{k-n} = \begin{bmatrix} {}^{G}\boldsymbol{p}_{I_{k-n}} \\ {}^{G}_{I_{k-n}}\boldsymbol{q} \end{bmatrix}.$$

We select an error-state Kalman filter implementation with the error state $\delta\boldsymbol{x} = (\delta\boldsymbol{x}_I, \delta\boldsymbol{\xi}_{k-1}, \delta\boldsymbol{\xi}_{k-2}, ...)$. The INS error state $\delta\boldsymbol{x}_I$ is defined as:

$$\delta\boldsymbol{x}_I = \begin{bmatrix} \delta\boldsymbol{p} \\ \delta\boldsymbol{v} \\ \delta\boldsymbol{\theta} \\ \delta\boldsymbol{b}_a \\ \delta\boldsymbol{b}_g \end{bmatrix} \in \mathbb{R}^{15}$$

where $\delta\boldsymbol{x}$, $\delta\boldsymbol{v}$, $\delta\boldsymbol{b}_a$, and $\delta\boldsymbol{b}_g$ are standard additive vector errors and $\delta\boldsymbol{\theta}$ is the local attitude error defined as:

$${}^{G}_{I}\boldsymbol{q}_{true} = {}^{G}_{I}\hat{\boldsymbol{q}} \otimes \delta\boldsymbol{q}$$

where $\delta\boldsymbol{q}$ is the small-angle quaternion:

$$\delta\boldsymbol{q} = \begin{bmatrix} 1 \\ \frac{1}{2}\delta\boldsymbol{\theta} \end{bmatrix}$$

Likewise, the pose error $\delta\boldsymbol{\xi}_{k-n}$ is defined as:

$$\delta\boldsymbol{\xi}_{k-n} = \begin{bmatrix} \delta\boldsymbol{p}_{k-n} \\ \delta\boldsymbol{\theta}_{k-n} \end{bmatrix} \in \mathbb{R}^6$$

For simplicity, we partition the MSCKF state as:

$$\boldsymbol{x} = (\boldsymbol{x}_I, \boldsymbol{x}_W)$$
$$\delta\boldsymbol{x} = (\delta\boldsymbol{x}_I, \delta\boldsymbol{x}_W)$$

where $\boldsymbol{x}_W$ is the list of past INS poses. The state covariance $\boldsymbol{P} \in \mathbb{R}^{15+6n \times 15+6n}$ is partitioned similarly:

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{P}_I & \boldsymbol{P}_{IW} \\ \boldsymbol{P}_{IW}^T & \boldsymbol{P}_W \end{bmatrix}$$

where $\boldsymbol{P}_I$ is the 15×15 matrix corresponding to the INS state, $\boldsymbol{P}_W$ is the $6n \times 6n$ matrix corresponding to the window of poses, and $\boldsymbol{P}_{IW}$ is the $15 \times 6n$ matrix of cross terms.

We also define the noise vector $\boldsymbol{w}$ as:

$$\boldsymbol{w} = \begin{bmatrix} \boldsymbol{w}_a \\ \boldsymbol{w}_g \\ \boldsymbol{w}_{ba} \\ \boldsymbol{w}_{bg} \end{bmatrix}.$$

4.2.2 Dynamic Model

The filter state is propagated according to Equation 4.5.

$$\begin{aligned} \dot{\boldsymbol{x}} &= f(\boldsymbol{x}, \boldsymbol{u}, 0) \\ \begin{bmatrix} \dot{\boldsymbol{x}}_I \\ \dot{\boldsymbol{x}}_W \end{bmatrix} &= \begin{bmatrix} f_I(\boldsymbol{x}, \boldsymbol{u}, 0) \\ \mathbf{0}_{6n\times 1} \end{bmatrix} \end{aligned} \tag{4.5}$$

where f_I is the INS state propagation function shown in equation4.2. The linearized error state dynamics are given in Equation 4.6

$$\begin{aligned} \delta\dot{\boldsymbol{x}} &= \boldsymbol{F}\delta\boldsymbol{x} + \boldsymbol{G}\boldsymbol{w} \\ \begin{bmatrix} \delta\dot{\boldsymbol{x}}_I \\ \delta\dot{\boldsymbol{x}}_W \end{bmatrix} &= \begin{bmatrix} \boldsymbol{F}_I & \mathbf{0}_{15\times 6n} \\ \mathbf{0}_{6n\times 15} & \mathbf{0}_{6n\times 6n} \end{bmatrix} \begin{bmatrix} \delta\dot{\boldsymbol{x}}_I \\ \delta\dot{\boldsymbol{x}}_W \end{bmatrix} + \begin{bmatrix} \boldsymbol{G}_I \\ \mathbf{0}_{6n\times 12} \end{bmatrix} \boldsymbol{w} \end{aligned} \tag{4.6}$$

where the INS error dynamics are derived from equation 4.1 as normal in the EsKF formulation:

$$\boldsymbol{F}_I = \frac{\partial f_I}{\partial \delta \boldsymbol{x_I}} \tag{4.7}$$

$$\boldsymbol{G}_I = \frac{\partial f_I}{\partial \boldsymbol{w}} \tag{4.8}$$

The discrete state transition matrix $\boldsymbol{\Phi}$ is computed as:

$$\boldsymbol{\Phi} = \exp(\boldsymbol{F}_I \Delta t) \tag{4.9}$$

and the discrete noise matrix $\boldsymbol{Q}_d$ as:

$$\boldsymbol{Q}_d = \boldsymbol{\Phi}\boldsymbol{G}_I\boldsymbol{Q}\boldsymbol{G}_I^T\boldsymbol{\Phi}^T\Delta t \tag{4.10}$$

where $\boldsymbol{Q}$ is the 12×12 covariance matrix associated with $\boldsymbol{w}$. The state covariance is then propagated according to Equation 4.11.

$$\boldsymbol{P}_{k+1} = \begin{bmatrix} \boldsymbol{\Phi}\boldsymbol{P}_{I,k}\boldsymbol{\Phi}^T + \boldsymbol{Q} & \boldsymbol{\Phi}\boldsymbol{P}_{IW,k} \\ \boldsymbol{P}_{IW,k}^T\boldsymbol{\Phi}^T & \boldsymbol{P}_W \end{bmatrix} \tag{4.11}$$

When a new image is recorded, the state is propagated forward in time up to the time the image was received at. At this point, a new IMU pose is inserted into the filter. This new pose is created as:

$$\boldsymbol{\xi}_k = \begin{bmatrix} {}^G\boldsymbol{p}_{I,k} \\ {}^G_I\boldsymbol{q}_k \end{bmatrix} \tag{4.12}$$

and inserted at the back of the window. Additionally, the covariance is augmented by:

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{I}_{6n+15} \\ \boldsymbol{J} \end{bmatrix} \boldsymbol{P} \begin{bmatrix} \boldsymbol{I}_{6n+15} \\ \boldsymbol{J} \end{bmatrix}^T \tag{4.13}$$

where

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{I}_3 & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} \\ \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} & \boldsymbol{I}_3 & \boldsymbol{0}_{3\times3} & \boldsymbol{0}_{3\times3} \end{bmatrix}$$

4.2.3 Measurement Model

The MSCKF uses the measurement function h defined in section 2.2:

$$h({}^G\boldsymbol{p}_I, {}^G_I\boldsymbol{q}, {}^G\boldsymbol{p}_f) = h(\boldsymbol{\xi}, {}^G\boldsymbol{p}_{f_i}) \tag{4.14}$$

$$= \frac{1}{{}^Cz_f} \begin{bmatrix} {}^Cx_f \\ {}^Cy_f \end{bmatrix} + \boldsymbol{w}_c \tag{4.15}$$

which describes the projection of a point in the camera frame onto the idealized image plane. While here we work in normalized coordinates, it is just as valid to work in pixel coordinates. In pixel coordinates $\boldsymbol{w}_c$ would be the image noise in units of pixels with standard deviation σ_{pix}. Working in normalized coordinates requires converting the following conversion:

$$\sigma_c = \frac{\sigma_{pix}}{f_{pix}} \tag{4.16}$$

where f_{pix} is the focal length of the camera in pixels, often taken as the average of f_x and f_y from the intrisic parameters matrix defined in section 2.2.

As the MSCKF does not maintain an estimate of ${}^G\boldsymbol{p}_f$, the position of the point in the global frame, it is estimated through the triangulation process outlined in section 2.2.2. After estimation, the following measurement jacobians are computed:

$$\boldsymbol{H}_\xi = \frac{\partial h}{\partial \boldsymbol{\xi}} \tag{4.17}$$

$$\boldsymbol{H}_f = \frac{\partial h}{\partial\, {}^G\boldsymbol{p}_f} \tag{4.18}$$

We denote $\boldsymbol{H}_\xi(\boldsymbol{\xi}, {}^G\boldsymbol{p}_f)$ and $\boldsymbol{H}_f(\boldsymbol{\xi}, {}^G\boldsymbol{p}_f)$ to be $\boldsymbol{H}_\xi$ and $\boldsymbol{H}_f$ evaluated at the given pose and feature point and $\boldsymbol{H}_{\xi_{ij}}$ and $\boldsymbol{H}_{f_{ij}}$ as the Jacobians evaluated for pose $\boldsymbol{\xi}_j$ and feature ${}^G\boldsymbol{p}_{f_i}$. To form the measurement model, we consider the case of a feature ${}^G\boldsymbol{p}_{f_i}$ observed from IMU poses $\boldsymbol{\xi}_{k-1}$ through $\boldsymbol{\xi}_{k-m}$. First, the residuals are computed for each viewing of the feature from IMU poses $j = k-1, ..., k-m$:

$$\boldsymbol{r}_{ij} = \boldsymbol{z}_{ij} - h(\boldsymbol{\xi}_j, {}^G\boldsymbol{p}_{f_i}) \tag{4.19}$$

$$\approx \boldsymbol{H}_{\xi_{ij}}\boldsymbol{\xi}_j + \boldsymbol{H}_{f_{ij}}\, {}^G\boldsymbol{p}_{f_i} + \boldsymbol{n}_{ij} \tag{4.20}$$

We collect all of the data from observations of ${}^G\boldsymbol{p}_{f_i}$ into the necessary structures following [33]:

$$\boldsymbol{z}_i \approx \boldsymbol{H}_{\xi_i}\delta\boldsymbol{x} + \boldsymbol{H}_{f_i}\delta\, {}^G\boldsymbol{p}_{f_i} + \boldsymbol{n}_i \tag{4.21}$$

where

$$\boldsymbol{z}_i = \begin{bmatrix} \boldsymbol{z}_{ik-1} \\ \boldsymbol{z}_{ik-2} \\ \dots \\ \boldsymbol{z}_{ik-m} \end{bmatrix} \tag{4.22}$$

$$\boldsymbol{n}_i = \begin{bmatrix} \boldsymbol{n}_{ik-1} \\ \boldsymbol{n}_{ik-2} \\ \dots \\ \boldsymbol{n}_{ik-m} \end{bmatrix} \tag{4.23}$$

$$\boldsymbol{H}_{f_i} = \begin{bmatrix} \boldsymbol{H}_{f_{ik-1}} \\ \boldsymbol{H}_{f_{ik-2}} \\ \dots \\ \boldsymbol{H}_{f_{ik-m}} \end{bmatrix} \tag{4.24}$$

$$\boldsymbol{H}_{\xi_i} = \begin{bmatrix} \mathbf{0}_{2\times15} & \boldsymbol{H}_{\xi_{ik-1}} & \mathbf{0}_{2\times6} & \dots & \mathbf{0}_{2\times6} \\ \mathbf{0}_{2\times15} & \mathbf{0}_{2\times6} & \boldsymbol{H}_{\xi_{ik-2}} & \dots & \mathbf{0}_{2\times6} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_{2\times15} & \mathbf{0}_{2\times6} & \mathbf{0}_{2\times6} & \dots & \boldsymbol{H}_{\xi_{ik-m}} \end{bmatrix} \tag{4.25}$$

This model is not usable for the EKF update in its current form as the current state estimate was used to compute the position of the feature and thus $\delta\boldsymbol{x}$ and $\delta\boldsymbol{p}_{f_i}$ are correlated. To account for this, *nullspace marginalization*[37] is employed to decorrelate the two. By multiplying through by the matrix spanning the left nullspace of the feature measurement Jacobian $\boldsymbol{H}_{f_i}$, the residuals $\boldsymbol{r}_i$ are mapped onto its nullspace, making them independent from errors in the feature

estimate. We term this new residual $\boldsymbol{r}_i^0$ and calculate it as follows:

$$\boldsymbol{A} = \texttt{null}(\boldsymbol{H}_{f_i}^T) \tag{4.26}$$

$$\boldsymbol{r}_i^0 = \boldsymbol{A}^T(\boldsymbol{z}_i - \hat{\boldsymbol{z}}_i) \tag{4.27}$$

$$\approx \boldsymbol{A}^T\boldsymbol{H}_{\xi_i}\delta\boldsymbol{x} + \boldsymbol{A}^T\boldsymbol{H}_{f_i}\delta\,{}^G\boldsymbol{p}_{f_i} + \boldsymbol{A}^T\boldsymbol{n}_i \tag{4.28}$$

$$= \boldsymbol{A}^T\boldsymbol{H}_{\xi_i}\delta\boldsymbol{x} + \boldsymbol{0} + \boldsymbol{n}_i^0 \tag{4.29}$$

$$= \boldsymbol{A}^T\boldsymbol{H}_{\xi_i}\delta\boldsymbol{x} + \boldsymbol{n}_i^0 \tag{4.30}$$

Here the second round of outlier rejection is performed. These residuals r_i^0 are validated with a Mahalanobis gating test to remove any remaining outliers. We stack the residuals and matrices from all features which pass the gating test, resulting in the following form:

$$\boldsymbol{r}^0 = \boldsymbol{H}_\xi^0\delta\boldsymbol{x} + \boldsymbol{n}^0 \tag{4.31}$$

which is valid to use in the EKF update along with their transformed measurement covariance:

$$\boldsymbol{R}^0 = \boldsymbol{H}_\xi^0\boldsymbol{R}\boldsymbol{H}_\xi^{0,T} \tag{4.32}$$

The filter correction then proceeds according to the EsKF update process. At this point, all feature observations used in the update are removed. We then check to see if any pose states have no more observations associated with them and, if so, delete the corresponding columns/rows of the state and covariance matrices.

4.3 An Optimization Approach to VINS: Batch Visual-Inertial SLAM

An alternative approach to visual-inertial estimation is to formulate it as a batch NLS problem. This approach, made popular by OKVIS [31], [32], estimates the IMU poses and landmarks to minimize a cost function relating them, weighted by sensor noise. This batch optimization can be performed over either a window of past poses and measurements, called *fixed-lag smoothing*, or the full trajectory and all measurements, called *full smoothing*. While it operates over a

collection of past poses like the MSCKF, the inclusion of landmarks into the estimation problem and the application of iterative cost minimization results in a significantly different estimator.

4.3.1 Estimator Structure

We formulate the batch least-squares VINS as maintaining a state vector consisting of N IMU poses and I landmarks. The state vector is then:

$$\boldsymbol{x} = (\boldsymbol{x}_I, \boldsymbol{x}_L)$$

where $\boldsymbol{x}_I = (\boldsymbol{x}_{I,k}, \boldsymbol{x}_{I,k-1}, \boldsymbol{x}_{I,k-2}, ..., \boldsymbol{x}_{I,k-N-1})$ is the list of N past IMU poses and $\boldsymbol{x}_L = (\boldsymbol{l}_i, \boldsymbol{l}_{i+1}, \boldsymbol{l}_{i+2}, ..., \boldsymbol{l}_I)$ is the list of all landmarks visible to the IMU poses. Like before, we define these IMU poses $\boldsymbol{x}_I$ as:

$$\boldsymbol{x}_{I,k} = \begin{bmatrix} {}^G\boldsymbol{p}_{I,k} \\ {}^G\boldsymbol{v}_{I,k} \\ {}^G_I\boldsymbol{q}_k \\ \boldsymbol{b}_{a,k} \\ \boldsymbol{b}_{g,k} \end{bmatrix}.$$

We define landmark states to be:

$$\boldsymbol{l}_i = {}^G\boldsymbol{p}_{L,i}$$

A visualization of this state vector is shown below in figure 4.2.

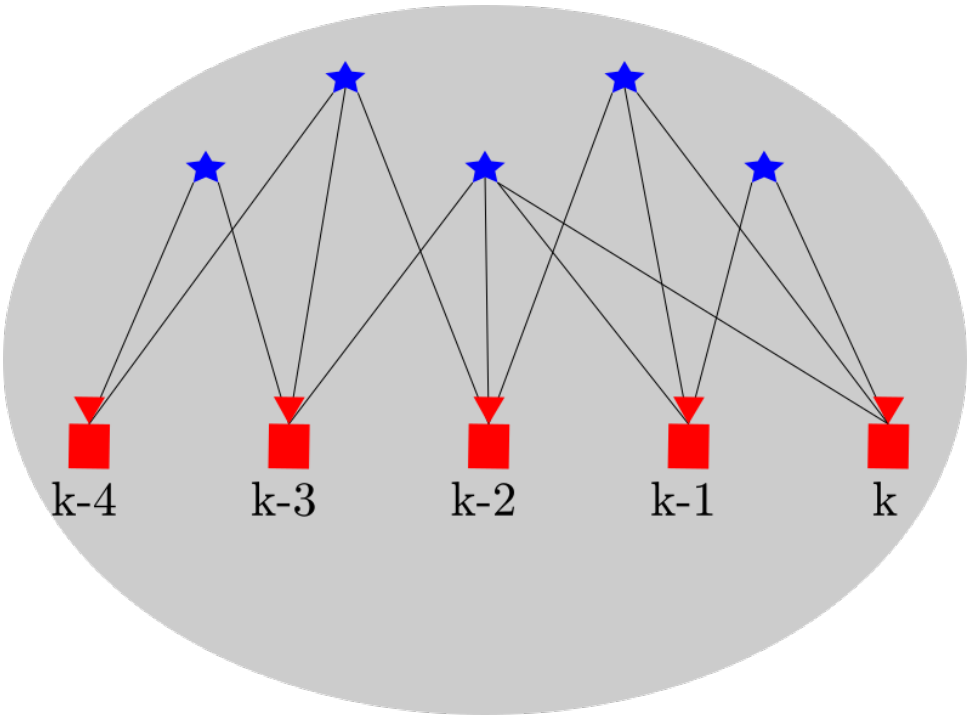

Figure 4.2: Structure of a batch optimization VINS

4.3.2 Dynamic Model

Like before, we consider a system moving through space 'driven' by inertial measurements. The state of this system is thus the same IMU propagation function define in equation 4.2:

$$\dot{\hat{\boldsymbol{x}}} = f_I(\hat{\boldsymbol{x}}, \boldsymbol{u}, 0)$$

$$\begin{bmatrix} {}^G\dot{\hat{\boldsymbol{p}}}_I \\ {}^G\dot{\hat{\boldsymbol{v}}}_I \\ {}^G_I\dot{\hat{\boldsymbol{q}}} \\ \dot{\hat{\boldsymbol{b}}}_a \\ \dot{\hat{\boldsymbol{b}}}_g \end{bmatrix} = \begin{bmatrix} {}^G\hat{\boldsymbol{v}}_I \\ \boldsymbol{C}({}^G_I\hat{\boldsymbol{q}})(\boldsymbol{a}_m - \hat{\boldsymbol{b}}_a) + {}^G\boldsymbol{g} \\ \frac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega}_m - \hat{\boldsymbol{b}}_g)\,{}^G_I\hat{\boldsymbol{q}} \\ \boldsymbol{0}_{3\times 1} \\ \boldsymbol{0}_{3\times 1} \end{bmatrix}.$$

This model is used to propagate the state of the system forward in time according to incoming measurements from the IMU. When a new image is received, the IMU state is propagated up to the time the image was received at and a new IMU pose is created in the estimator [2].

[2]For efficiency, often we decline to add a new pose until the system has moved significantly from the last one as closely-spaced poses provided little increase in accuracy. This process is known as *keyframing*.

4.3.3 Measurement Model

As before, we use the normalized projection function equation 2.4:

$$\boldsymbol{z}_{k,i} = h_{norm}(\boldsymbol{x}_{I,k}, \boldsymbol{l}_i)$$

where $\boldsymbol{z}_{k,i}$ is the observation of the i^{th} landmark from the k^{th} IMU pose. When an estimate of a new landmark becomes available, it is added to the state vector and the IMU poses it is observed from are recorded.

4.3.4 Cost Function Formulation

The batch least-squares approach seeks to minimize a cost function consisting of IMU odometry terms and landmark measurement terms. To this end, the IMU and measurement error functions are defined as follows:

$$\boldsymbol{e}_I(\boldsymbol{x}_{I,k}, \boldsymbol{x}_{I,k-1}, \boldsymbol{u}_{k-1}) = \boldsymbol{x}_{I,k} \boxminus f_k(\boldsymbol{x}_{I,k-1}, \boldsymbol{u}_{k-1}, 0) \tag{4.33}$$

$$\boldsymbol{e}_F(\boldsymbol{x}_{I,k}, \boldsymbol{l}_i) = \boldsymbol{z}_{k,i} - h_{norm}(\boldsymbol{x}_{I,k}, \boldsymbol{l}_i) \tag{4.34}$$

where f_k is the IMU dynamic model from equation 4.2 and h is the perspective projection equation 2.4. The optimization problem is then:

$$\underset{\boldsymbol{x}}{\operatorname{argmin}} \sum_i \boldsymbol{e}_i^T(\boldsymbol{x}) \boldsymbol{\Sigma}^{-1} \boldsymbol{e}_i(\boldsymbol{x}) \tag{4.35}$$

$$= \underset{\boldsymbol{x}_I, \boldsymbol{x}_L}{\operatorname{argmin}} \sum_k ||\boldsymbol{x}_{I,k} \boxminus f(\boldsymbol{x}_{I,k-1}, \boldsymbol{u}_{k-1}, 0)||^2_{\boldsymbol{\Sigma}_I} + \sum_k \sum_i ||\boldsymbol{z}_{k,i} - h(\boldsymbol{x}_{I,k}, \boldsymbol{l}_i)||^2_{\boldsymbol{\Sigma}_c} \tag{4.36}$$

where the first term represents the difference between predicted and estimated IMU poses and the second term represent the difference between measured and estimated camera measurements. We use the notation $||\boldsymbol{e}||^2_{\boldsymbol{\Sigma}} = \boldsymbol{e}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{e}$ to denote that terms are weighted according to their uncertainties.

Like other VINS, pixel measurements are undistorted before processing so there is no need to include distortion in h.

4.3.5 Optimization

This formulation results in a somewhat obvious Jacobian structure, which we illustrate with an example. Say we have IMU poses at $k = 1, 2, 3$ and landmarks $\boldsymbol{l}_1, \boldsymbol{l}_2$. Landmark $\boldsymbol{l}_1$ is observed from $\boldsymbol{x}_{I,1}$ and $\boldsymbol{x}_{I,2}$ while landmark $\boldsymbol{l}_2$ is observed from $\boldsymbol{x}_{I,1}$, $\boldsymbol{x}_{I,2}$, and $\boldsymbol{x}_{I,3}$. The estimated update vector is then:

$$\boldsymbol{x} = \begin{bmatrix} \delta\boldsymbol{x}_{I,1} \\ \delta\boldsymbol{x}_{I,2} \\ \delta\boldsymbol{x}_{I,3} \\ \delta\boldsymbol{l}_1 \\ \delta\boldsymbol{l}_2 \end{bmatrix}$$

where the δ states represent the updates to the current state estimate. The Jacobian of the error function with resptect to the state is:

$$\boldsymbol{J}_e = \begin{bmatrix} -\boldsymbol{I} & 0 & 0 & 0 & 0 \\ \boldsymbol{F}_2 & -\boldsymbol{I} & 0 & 0 & 0 \\ 0 & \boldsymbol{F}_3 & -\boldsymbol{I} & 0 & 0 \\ \boldsymbol{H}_{x_{1,1}} & 0 & 0 & \boldsymbol{H}_{f_{1,1}} & 0 \\ 0 & \boldsymbol{H}_{x_{2,1}} & 0 & 0 & \boldsymbol{H}_{f_{2,1}} \\ \boldsymbol{H}_{x_{1,2}} & 0 & 0 & \boldsymbol{H}_{f_{1,2}} & 0 \\ 0 & \boldsymbol{H}_{x_{2,2}} & 0 & 0 & \boldsymbol{H}_{f_{2,2}} \\ 0 & 0 & \boldsymbol{H}_{x_{2,3}} & 0 & \boldsymbol{H}_{f_{2,3}} \end{bmatrix}$$

where

$$\begin{aligned} \boldsymbol{F}_j &= \frac{\partial f}{\partial x_I}|_{x_I = x_{I,j-1}, u = u_{j-1}} \\ \boldsymbol{H}_{x_{i,j}} &= \frac{\partial h}{\partial x_I}|_{x_I = x_{I,j}, l = l_i} \\ \boldsymbol{H}_{f_{i,j}} &= \frac{\partial h}{\partial l}|_{x_I = x_{I,j}, l = l_i} \end{aligned}$$

The error vector is:

$$\boldsymbol{e}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{x}_{I,1} \\ \boldsymbol{x}_{I,2} - f(\boldsymbol{x}_{I,1}, \boldsymbol{u}_1, 0) \\ \boldsymbol{x}_{I,3} - f(\boldsymbol{x}_{I,2}, \boldsymbol{u}_2, 0) \\ \boldsymbol{z}_{1,1} - h(\boldsymbol{x}_{I,1}, \boldsymbol{l}_1) \\ \boldsymbol{z}_{1,2} - h(\boldsymbol{x}_{I,1}, \boldsymbol{l}_2) \\ \boldsymbol{z}_{2,1} - h(\boldsymbol{x}_{I,2}, \boldsymbol{l}_1) \\ \boldsymbol{z}_{2,2} - h(\boldsymbol{x}_{I,2}, \boldsymbol{l}_2) \\ \boldsymbol{z}_{3,2} - h(\boldsymbol{x}_{I,3}, \boldsymbol{l}_2) \end{bmatrix}$$

From these structures we form the Gauss-Newton update equations and solve for $\delta\boldsymbol{x}$ until convergence:

$$\boldsymbol{J} = \boldsymbol{e}(\boldsymbol{x})^T \boldsymbol{\Sigma}^{-1} \boldsymbol{J}_e$$
$$\boldsymbol{H} = \boldsymbol{J}_e^T \boldsymbol{\Sigma}^{-1} \boldsymbol{J}_e$$
$$\delta\boldsymbol{x} = -\boldsymbol{H}^{-1} \boldsymbol{J}^T$$

It is generally easiest to construct these equations on a per-errror-term basis. If our system has m error terms (e.g. $\boldsymbol{e}(\boldsymbol{x}) = (\boldsymbol{e}_1, \boldsymbol{e}_2, ..., \boldsymbol{e}_m)$), then the system can be formed as:

$$\boldsymbol{J} = \sum_m \boldsymbol{e}_m^T \boldsymbol{\Sigma}_m^{-1} \boldsymbol{J}_{e,m}$$
$$\boldsymbol{H} = \sum_m \boldsymbol{J}_{e,m}^T \boldsymbol{\Sigma}_m^{-1} \boldsymbol{J}_{e,m}$$

where $\boldsymbol{J}_{e,m}$ is the m^{th} row of $\boldsymbol{J}_e$ and $\boldsymbol{\Sigma}_m$ is the covariance associated with $\boldsymbol{e}_m$.

Like the MSCKF, batch least-squares VINS operates over a bounded window of past states and observations. To keep the window bounded it is necessary to marginalize (remove) states intermitently. This marginalization process is non-trivial and is explained in detail in [31].

4.3.6 Discussion

Unlike the filtering approach which will have to be almost completely manually implemented, there exist several frameworks to simplify the implementation of optimization-based estimators. These include Ceres [1], g2o [29], and GTSAM [14].

A difficult choice in the design of an optimization-based VINS is whether to perform fixed-lag smoothing or full smoothing. Historically, the computational load of full-smoothing has made it infeasible for real-time systems, resulting in most early optimization-based VINS performing fixed-lag smoothing over a window of keyframes as in OKVIS [31]. Immediately these methods demonstrated improved performance over existing filtering approaches and have come to dominate the VINS landscape [45].

Recent methods have bridged the gap between fixed-lag and full smoothing by augmenting a fixed-lag VINS with a loop-closing backend. This backend solves another optimization problem, one including only the estimated keyframe poses and relative constraints provided by visual place recognition to detect when the system revisits a location. The optimized poses are then fed back into the fixed-lag VINS frontend. This approach, taken by VINS-MONO [40], provides an efficient approximation of a full smoothing approach and is generally considered the state of the art in VINS.

Even more recently, incremental smoothing methods such as ISAM [26] and ISAM2 [27] have enabled true full-smoothing visual-inertial estimation [16] by modifying the optimization problem to marginalize out states unaffected by incoming measurements. These full-smoothing methods demonstrate superior accuracy at the cost of numerical instability and sensitivity to outliers [13], and thus require advanced outlier rejection schemes for use in real-world systems. An active area of work in this project is the development

Chapter 5

Measurement Validation in Visual-Inertial Navigation

This chapter presents an analysis of the methods used to validate measurements in visual-inertial estimators. This analysis then informs the development of an observability-aware validation algorithm to delay the validation of measurements corresponding to a landmark until the landmark becomes observable, but no longer. An experimental VINS is then developed to compare this measurement validation method against other existing visual-inertial estimators with the specific goal of improving upon the robustness properties of the system proposed by Forster et al. in [16].

5.1 Measurement Validation in Visual-Inertial Navigation

Despite increasingly feature detection and data association methods, it is extremely common for measurements detrimental to the overall navigation solution to pass through outlier rejection and into the estimator. As a result, it is standard practice to perform a measurement validation step before including tracked features in any estimation process. In this work we focus on *geometric* validation, which validates measurements by determining if they are consistent with the existing state estimates. This consistency is observed by attempting to estimate the position of the landmark from the current state estimates and collected measurements. If the estimation fails for any of a variety of reasons the measurements are treated as invalid and discarded.

The standard approach to this validation is that taken by VINS-MONO [40] and the MSCKF [37]. These estimators attempt to triangulate the landmark using all measurements and camera positions currently in their window of estimation. In both cases these are extremely

robust methods of validation as they perform this estimation with the maximum information possible, bounded by the maximum length of their windows. These two systems rely on as accurate an initial estimation step as possible as they then directly compute navigation solutions using these landmark initializations.

We notice two immediate areas for improvement with these estimators. VINS-MONO performs motion-only optimization for its navigation estimates. That is, it does not optimize landmarks after they are initialized and must then re-initialize them at every optimization. This approach, while efficient, prevents measurements corresponding to poses that have passed out of the window from being considered in newer triangulations, and therefore the accuracy of the estimated landmark positions is suboptimal. Additionally, this results in a full re-estimation of the feature position every time after the first initialization, In contrast, the MSCKF must process all observations at once and only once, and thus must wait until all observations of the feature have been received to validate it. We refer to this problem as having *maximally-delayed* validation, as the MSCKF does not validate features (and thus does not process filter corrections) until the feature is lost. To correct these issues, we propose that an ideal measurement validation scheme would have the following two properties:

- Maximize the use of prior information used when validating
- Minimize the delay between receiving measurements and including them in the navigation solution

5.2 Maximizing Prior Information in Measurement Validation

We begin by addressing the need to include prior information in the validation process. In the case of a newly-seen landmark, there is not much room for improvement as the landmark must still be initialized. However, we can significantly improve the cost of validating newer measurements after the feature is initialized. The key problem with the approach taken by VINS-MONO is that it re-processes measurements already used to estimate the feature position. Instead, we decide to maintain estimated landmarks in the state of the estimator as in EKF-SLAM and other full VISLAM systems. By maintaining this estimate, the validation process

for new measurements of already-initialized landmarks reduces from a full re-initialization to a residual check, validating that the received measurement falls within an expected range of the estimated landmark.

This simple change describes moving from visual-inertial odometry to visual-inertial SLAM, as now a map is being actively maintained rather than as a by-product of pose estimation. This comes with a significant increase in computational complexity, as the state vector now contains To accomplish this task, we propose to follow the methodology used in [16], where the incremental smoothing algorithm ISAM2 [27] allows for efficient updating of large state vectors by temporarily marginalizing out states that are not affected in the current update.

5.3 Minimizing the Delay Caused by Validation

This decision to validate once and update after then motivates the second problem of minimizing the delay between receiving measurements of a landmark and using it in the estimated state. This is equivalent to minimizing the number of measurements of a landmark needed before validating those observations. To understand why the delay occurs in the validation process, we discuss the observability properties of landmarks in visual SLAM systems.

5.3.1 Geometry of Triangulation

The triangulation problem presented in section 2.2.2, like many estimation problems, suffers from the necessary condition of *observability*. Observability, the mathematical property of an estimation problem being uniquely solvable, is of great importance in all navigation systems but is of particular importance in the case of visual SLAM as projection of landmarks onto images results in a measurement of lower dimensionality than the quantity to estimate. This loss of information prevents a single measurement from providing an estimate of the landmark's location in space, and thus multiple measurements are required to solve for the landmark's position and for its measurements to aid the navigation solution. The observability of the point location is dependent on the geometric relationships between the measured bearing vectors. The geometry of an example two-view point estimation problem in two dimensions is shown in figure 5.1, with the angle θ being known as the *parallax* between the observation vectors.

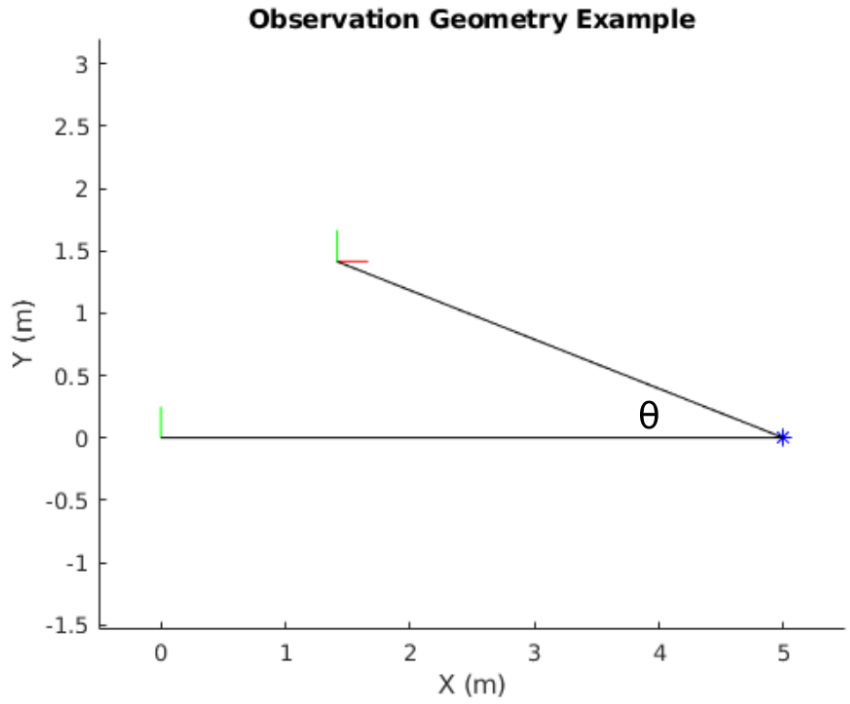

Figure 5.1: Feature observation geometry

Individual bearing vector observations define a vector beginning at the camera and pointing out into space on which the landmark must lie, but the location of the landmark along the vector is unknown. Continued measurements of the landmark from different camera locations provide additional vectors on which the feature must lie, the intersection of which is then the location of the landmark in space. The uncertainty in these bearing measurements causes the measured bearing vectors to define cones reaching out from the camera rather than true vectors. The intersection of these cones then defines a region containing the estimated landmark location. Figures 5.2 and 5.3 demonstrate regions of intersection for two difference observation geometries. It is clear that perpendicular bearing vectors produce the most observable landmark position while nearly-colinear bearing vectors provide very little observability. This is verified in simulation as shown below.

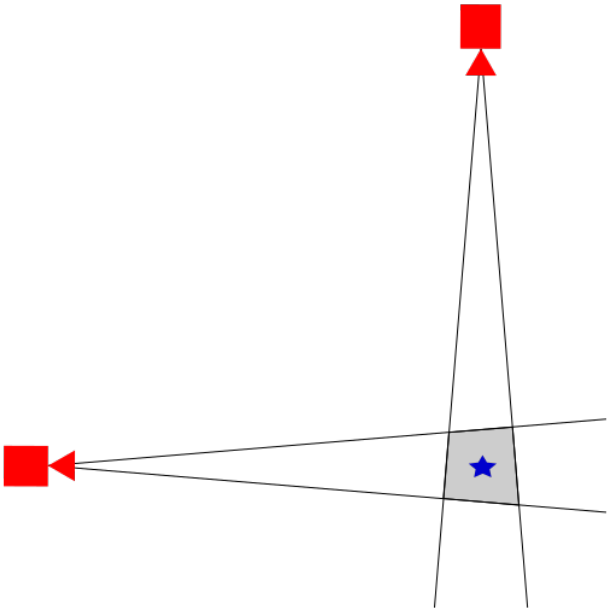

Figure 5.2: Good observation geometry

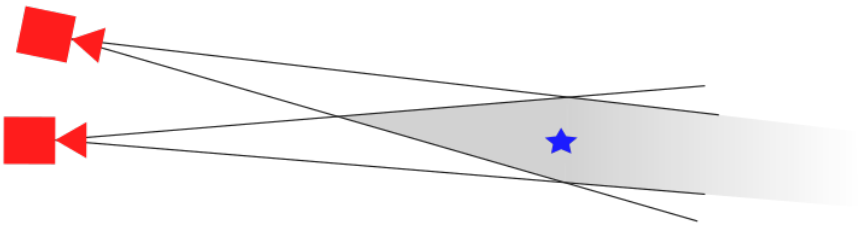

Figure 5.3: Bad observation geometry

To better understand the relationship between the colinearity of the bearing vectors and the landmark estimation problem, a Monte-Carlo simulation was performed with a camera fixed at $(0, 0)$, a landmark fixed five meters in front of the camera, and a second camera moving in a circular path around the first camera, pointing towards the landmark to simulate a variety of parallaxes. Each pose was sampled 100 times with measurement error $w_{pix} \sim N(0, 1pix)$. The sample poses are shown below in figure 5.4 and the resulting triangulation errors are shown in figure 5.5.

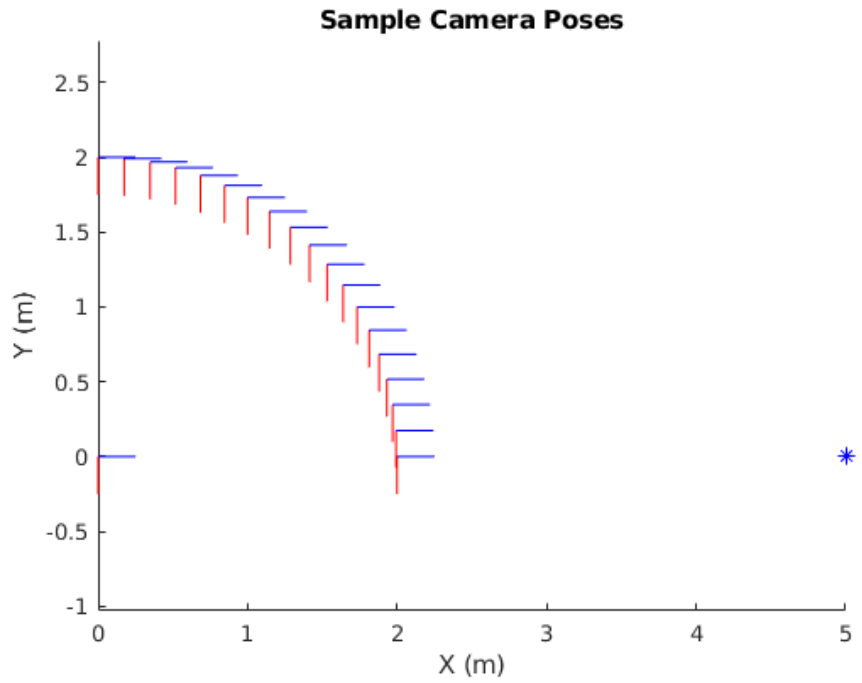

Figure 5.4: Triangulation results with well-conditioned observation geometry

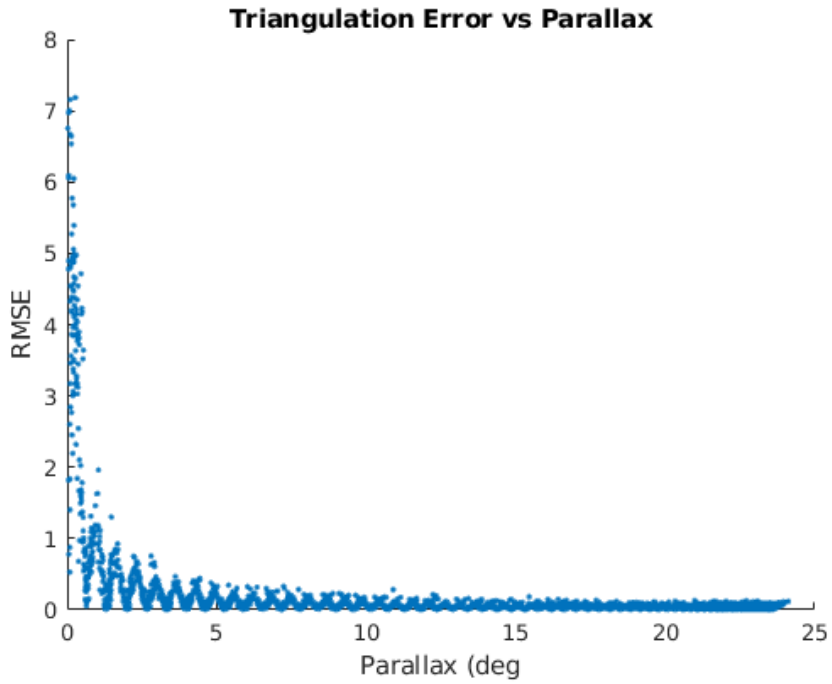

Figure 5.5: Triangulation errors resulting from a range of parallaxes

This simulation demonstrates the aforementioned observability effects and provides some insight into the relationship between observation geometry and the ability to accurately estimate landmarks: in the presence of noisy camera measurements, there is a threshold below which reliable landmark estimation is unreliable and above which it is reliable. Additionally, we can observe that valid measurments can result in large errors if the geometry of observation is ill-posed. Errors this large in a validation attempt will generally result in a failed valiation despite the measurements being valid.

5.3.2 Minimally-Delayed Validation

Drawing on the conclusions of the previous section, we can make the following inferences:

1. Geometric validation of poorly-observable measurments is untrustworthy
2. Parallax is a reliable metric for the observability of landmarks
3. Above a minimum parallax, observability is constant

From these observations we propose that to delay geometric validation until the collected measurements reach a threshold of observability is to maximize the likelihood of accurate measurement validation and minimize the number of observations required to perform a trustworthy validation. To this end we propose an adaptive measurement validation method which we name *Minimally-Delayed Validation* in which we delay validation of a new landmark until the moment it becomes observable, then perform geometric validation. To do so, each time a measurement of the landmark is recieved we record the measurement $\boldsymbol{z}$ and the camera pose $({}^{W}\boldsymbol{p}_{C}, {}^{W}_{C}\boldsymbol{q})$ it was observed from. For each new measurement $\boldsymbol{z}_k$, we compute the parallax angle θ between the first and newest measurements using Equation 5.1

$$\theta = \arccos\left(\frac{({}^{W}\boldsymbol{z}_1{}^{T} \cdot {}^{W}\boldsymbol{z}_k)}{||{}^{W}\boldsymbol{z}_1|| \cdot ||{}^{W}\boldsymbol{z}_k||}\right) \tag{5.1}$$

where

$${}^{W}\boldsymbol{z}_1 = \boldsymbol{C}({}^{W}_{C}\boldsymbol{q}_1) \begin{bmatrix} \boldsymbol{z}_1 \\ 1 \end{bmatrix}$$

$${}^{W}\boldsymbol{z}_k = \boldsymbol{C}({}^{W}_{C}\boldsymbol{q}_k) \begin{bmatrix} \boldsymbol{z}_k \\ 1 \end{bmatrix}$$

If θ is above a threshold (0.12 radians in our experiments), then the landmark is considered observable and geometric validation is performed. Otherwise, the landmark is saved for attempts with future measurements. Upon successful geometric validation, the full list of measurements is added to the optimization problem and a new solution is computed, resulting in updated state estimates and the adding of the landmark to the SLAM map. From this point on validation of measurements of this landmark consists of a simple residual check, where if measurement residuals are too large the measurement is considered invalid. Alternatively, a failed geometric validation results in the stored measurements of the landmark being eliminated.

To validate this method, we first demonstrate it in simulation. Simple scenarios of cameras moving relative to landmarks at different locations were run as Monte Carlo trials with noisy odometry and camera measurements. These landmark locations are a constant distance from the

camera's starting point, offset by angles from 0 to 45 degrees. Landmarks are validated in two ways: with all of the measurements collected (denoted "First-Last" as the initial triangulation is performed with the first and last observations) and with the observations collected up to and including the observability threshold (denoted "First-Mid"). After this validation, features are optimized using a Gauss-Newton process using all measurements collected over the trajectory. A sample of the generated landmark locations and the trajectory taken by the camera is shown below in figure 5.6.

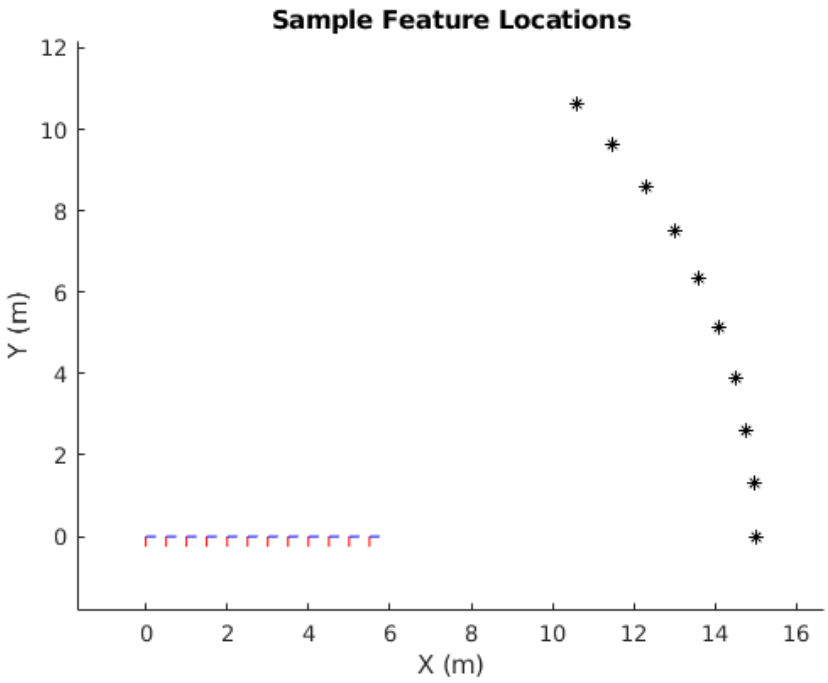

Figure 5.6: Sample landmark locations used in validation simulation

Figures 5.7 through 5.10 demonstrate two illustrative examples, one with well-posed geometry and one with ill-posed geometry. The first example, shown in figures 5.7 and 5.8, shows validation and estimation of a highly-observable feature where the observability condition is reached well before the last measurement is received. These plots show that, given good observability, estimates of the landmark's position from the point of observability are extremely similar to those using all measurements. Additionally, the iterative updates performed using Gauss-Newton converge to the same values as the First-Last batch solution, validating that there is no loss of final accuracy from this earlier initialization.

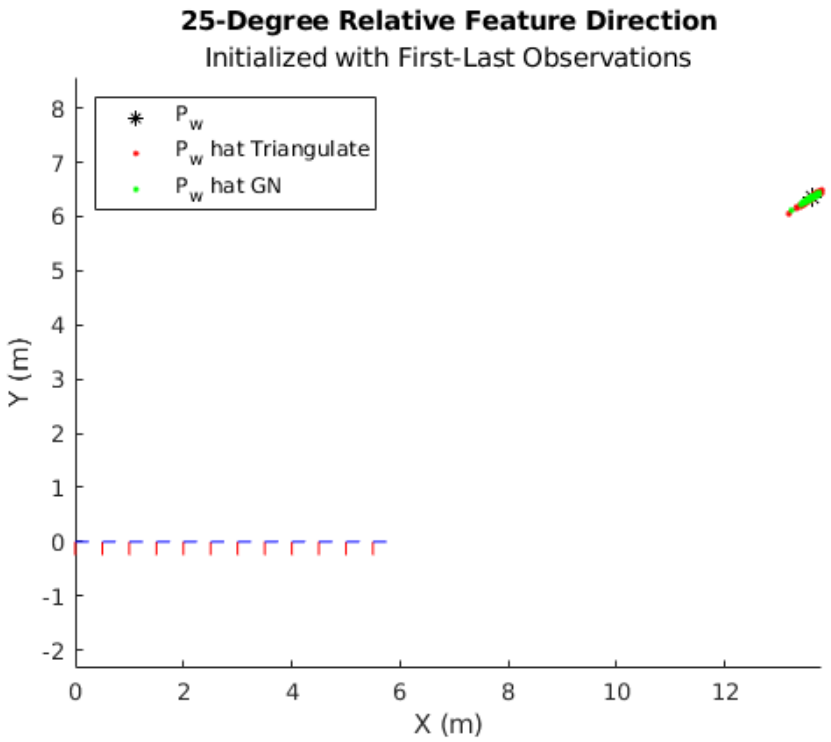

Figure 5.7: Feature estimation after First-Last initialization with good observation geometry

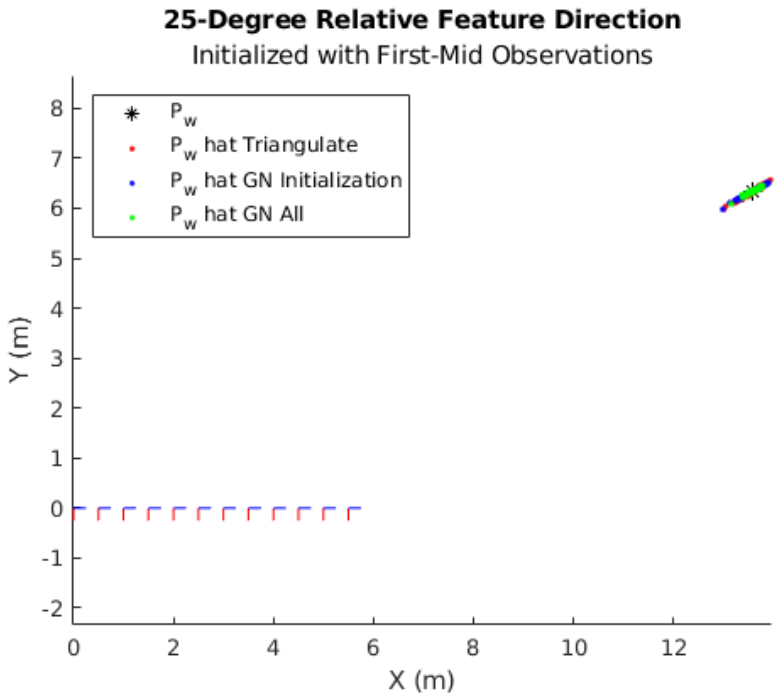

Figure 5.8: Feature estimation after First-Mid initialization with good observation geometry

The second example, shown in figures 5.9 and 5.10, demonstrates the same process as before. This time, as all samples are ill-posed, the observability condition is never met and the MDV estimator refuses to initialize the landmark. On the other hand, the First-Last estimator proceeds to initialize the landmark to a wide range of inaccurate locations. These locations all pass the geometric validation checks but would be detrimental to a navigation solution.

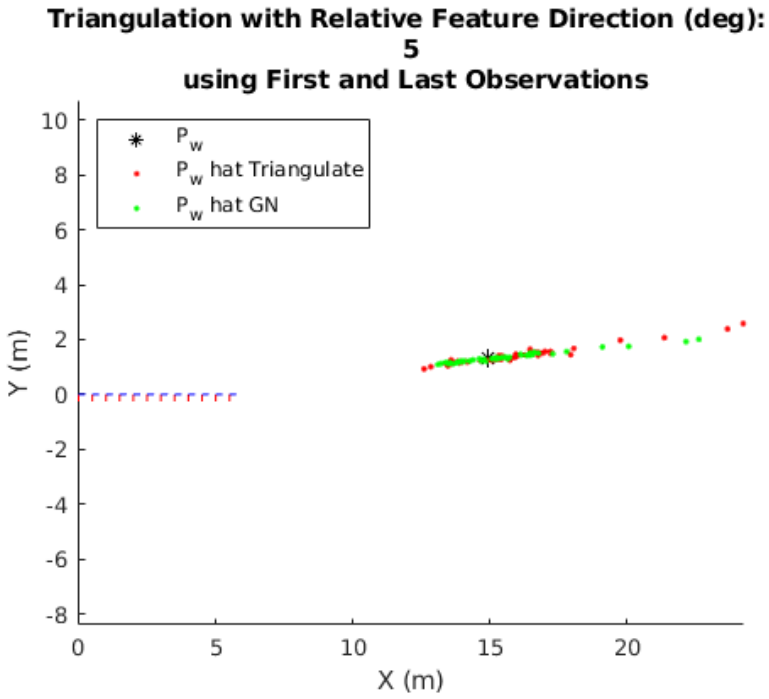

Figure 5.9: Feature estimation after First-Last initialization with poor observation geometry

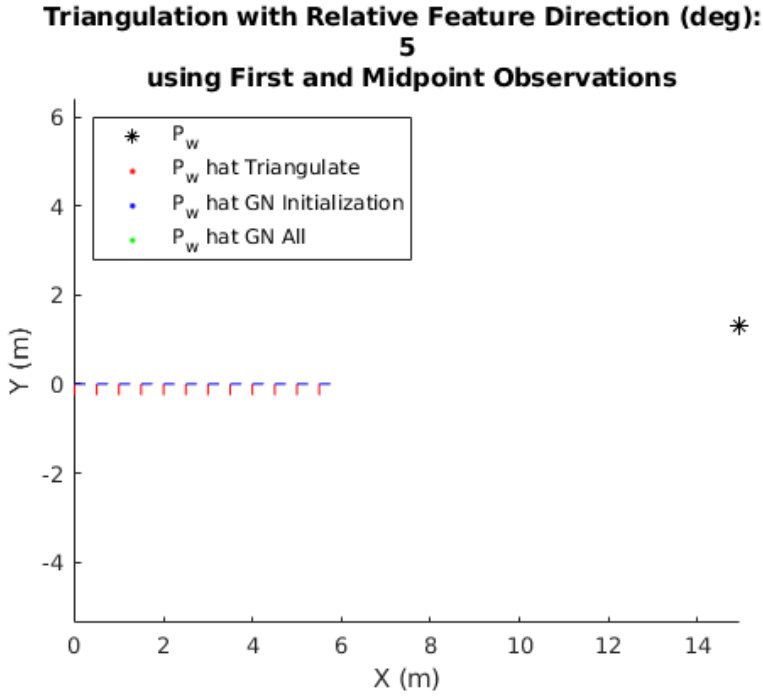

Figure 5.10: Feature estimation after First-Mid initialization with poor observation geometry

The mean error for each of these trials is shown below in figure 5.11, where we compare the RMSE for 100 trials for the landmark positions estimated by First-Last, First-Mid at validation, and First-Mid after all measurements have been processed. This simulation affirms that there is minimal difference in the estimated landmark position after observability has been reached. Additionally, it confirms the prior assumption that parallax is a primary factor in landmark estimation accuracy as the error at initialization for First-Mid is approximately constant for a constant initialization parallax.

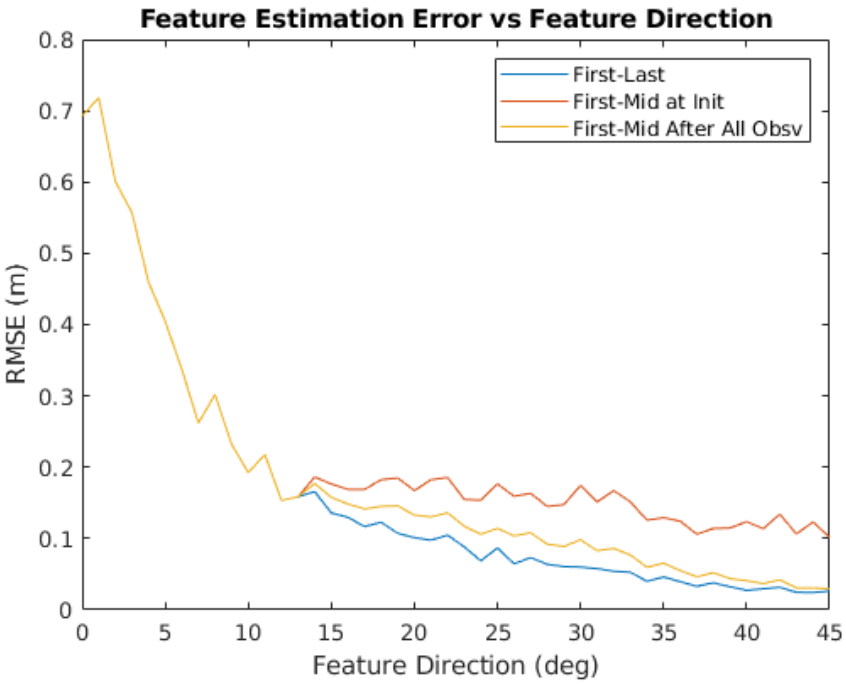

Figure 5.11: Error in estimated feature position using First-Mid algorithm

Figure 5.12 below shows the number of observations required before validation is performed for the First-Last and First-Mid estimators. It is clear that only a small displacement from the camera's optical axis is required before features become observable after very few measurments. This effect is clearly visible despite the simulated environments being close to the worst case for parallax generation, as longitudinal motion generates very small parallax for a given magnitude of motion for landmarks in front of the camera.

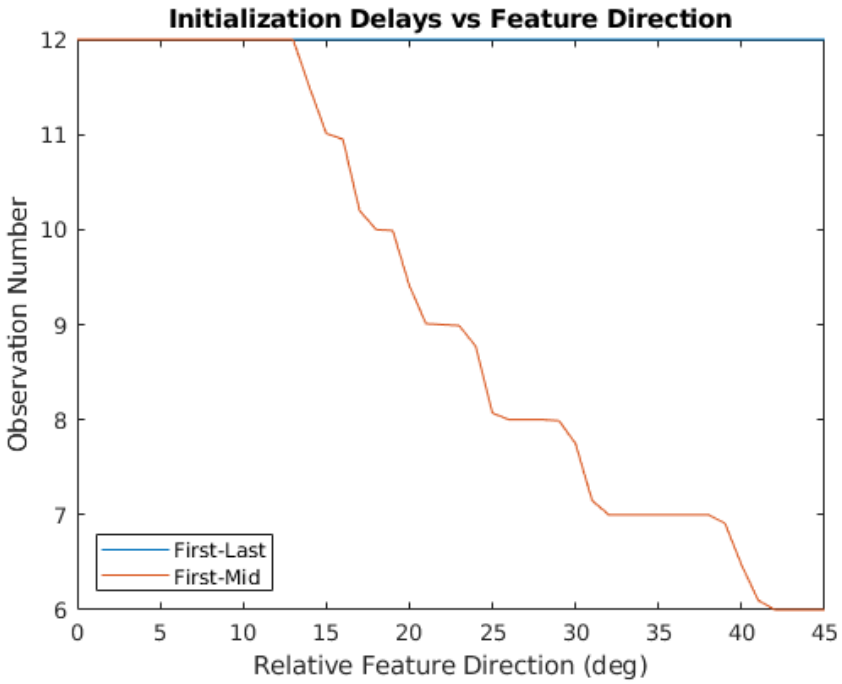

Figure 5.12: Observations used before initializing feature

Chapter 6

Results

This chapter presents the results of experiments using the proposed method in a functioning VINS. Following in the footsteps of [13], a collection of visual-inertial estimators are organized to compare the proposed system against. This collection is chosen so as to include a range of strategies from both filtering and optimization categories to provide a accurate representation of the state of the art in VINS. Summaries of these systems and their key features are presented below.

6.1 Proposed System

Following the simulated results shown in the previous chapter, we implement the proposed method in a functional VINS for comparison against the state-of-the-art. The feature detector and tracker are modelled after that implemented in VINS-MONO [40], and uses the Lucas-Kanade optical flow tracker [35], [6] to track Shi-Tomasi corners [25] across sequences of images. The visual-inertial fusion is performed following the method proposed in Forster et al. [16], where a full smoothing backend using the implementation of ISAM2 found in GTSAM [14] is leveraged to allow for arbitrarily long tracking of features. This system is implemented in ROS Noetic [44] using GTSAM [14] and OpenCV [7].

6.2 Comparison VINS Implementations

6.2.1 MSCKF-MONO

We include the Multi-State Constraint Kalman Filter (MSCKF)[37] in our evaluation as it is the most commonly used and best known filtering VINS. Developed to address the computational cost and linearization problems of previous EKF-based visual-inertial estimators, the MSCKF introduced a novel filter structure where features are not actually maintained and estimated. By maintaining a sliding window of past poses instead and employing a measurement model based on the geometric constraints between poses, the MSCKF demonstrates improved accuracy and computational efficiency over the classical EKF-SLAM system. MSCKF-MONO[51] is a public implementation of the MSCKF which includes the observability-constrained modifications proposed in [24] for improved consistency. This implementation is available at `https://github.com/daniilidis-group/msckf_mono`. The MSCKF is explained in detail in section 4.2. Additionally, the MSCKF technical report by Mourikis and Roumeliotis[36] is a comprehensive description.

6.2.2 OKVIS

We include OKVIS [31], [32] as the first major optimization-based VINS. Originally developed for use with stereo cameras, it was later extended to the monocular case and continued to show strong performance. OKVIS would still be considered the standard batch-optimization visual-inertial SLAM system as, unlike VINS-MONO, it includes active features in the optimization problem to both improve estiamtion accuracy and generate an accurate map. A simplified version of OKVIS is described in section 4.3. A key contribution of OKVIS that is not addressed here is its marginalization of past states to maintain their information in the optimization problem, a feature not addressed in other windowed optimization schemes. OKVIS is publicly available at `https://github.com/ethz-asl/okvis`.

6.2.3 ROVIO

First presented by Bloesch et al. in 2015, Robust Visual Inertial Odometry (ROVIO) [5], [4] is a VINS using the EKF-SLAM estimator design. ROVIO proposed an augmented measurement scheme which tracks the image patches surrounding tracked features. These patches are warped based on the predicted feature location and IMU motion and then the photometric error between the predicted and measured patch is used to form the EKF residual vector. Additionally, ROVIO parameterizes features into a combination of a depth factor and bearing vector, improving the Gaussian error assumption on feature positions. The public implementation of ROVIO is available at `https://github.com/ethz-asl/rovio`.

6.2.4 VINS-MONO

VINS-MONO [40] was a groundbreaking development in the visual-inertial literature. Despite being conceptually similar to OKVIS, VINS-MONO introduced several new features such as:

- A robust bootstrapping system to initialize the system from arbitrary initial states
- Online calibration of IMU-camera extrinsic parameters (seen before in EKF-based estimators, but not in optimization-based estimators)
- 4 degree-of-freedom pose-graph optimization to account for scale drift

and others. Upon release, VINS-MONO immediately became the gold standard for visual-inertial estimators and has maintained that status in the years since. VINS-MONO is publicly available at `https://github.com/HKUST-Aerial-Robotics/VINS-Mono`.

While VINS-MONO does include loop closure and pose-graph optimization, the other systems being compared do not and so we present results with those functionalities disabled. See [13] for a comparison which includes VINS-MONO's more advanced capabilities.

6.2.5 SVO + GTSAM

SVO [17] is a lightweight visual-odometry system developed for use on MAVs with limited computational capabilities. It is a semi-direct method like ROVIO, and thus is generally robust

to lighting changes and low texture. In Forster et al. [16], SVO was paired with a full smoothing backend implemented using GTSAM's [14] implementation of ISAM2 [27] This is the first publicly-proposed VINS to use the ISAM2 algorithm for estimation, While the implementation itself is not public, SVO is available at `https://github.com/uzh-rpg/rpg_svo` and GTSAM is available at `https://github.com/borglab/gtsam`.

6.3 Experiments

The EUROC MAV datasets [9] are a group of visual-inertial datasets collected from a manually-flown micro aerial vehicle (MAV). These datasets contain synchronized visual-inertial data along with ground truth trajectories and have become the standard dataset for visual-inertial estimation as they cover a wide range of scenarios. Each flight is approximately 2.5 minutes long and includes an initialization period, standstill period, and then flight around the environment. The data available in the datasets consists of synchronized grayscale WVGA stereo images at 20Hz and IMU data at 200Hz, along with ground truth provided by a motion-capture system. In these experiments we consider only monocular (single-camera) estimators, and as such use only the images from the left camera. An example image taken from flight MH03 is shown below in figure 6.1.

Figure 6.1: An example image taken from the EUROC MAV dataset

The EUROC dataset consists of three sub-datasets, each consisting of several flights performed in a different area of the test facility. These sub-datasets are structured so that each sub-dataset gets progressively more challenging. For example, the Machine Hall sub-dataset (MH) is subdivided into five flights: MH01-easy, MH02-easy, MH03-medium, MH04-difficult, and MH05-difficult. The Vicon 1 (V1) and Vicon 2 (V2) datasets, while consisting of three instead of five flights each, follow the same easy-medium-difficult system. In the Machine Hall flights, difficulty is increased through a combination of more aggressive flights and through decreased lighting. The Vicon flights increase difficulty through more aggresive flights, lighting changes, and decreased scene texture. Examples of these decreased lighting and texture scenarios are shown below in figures

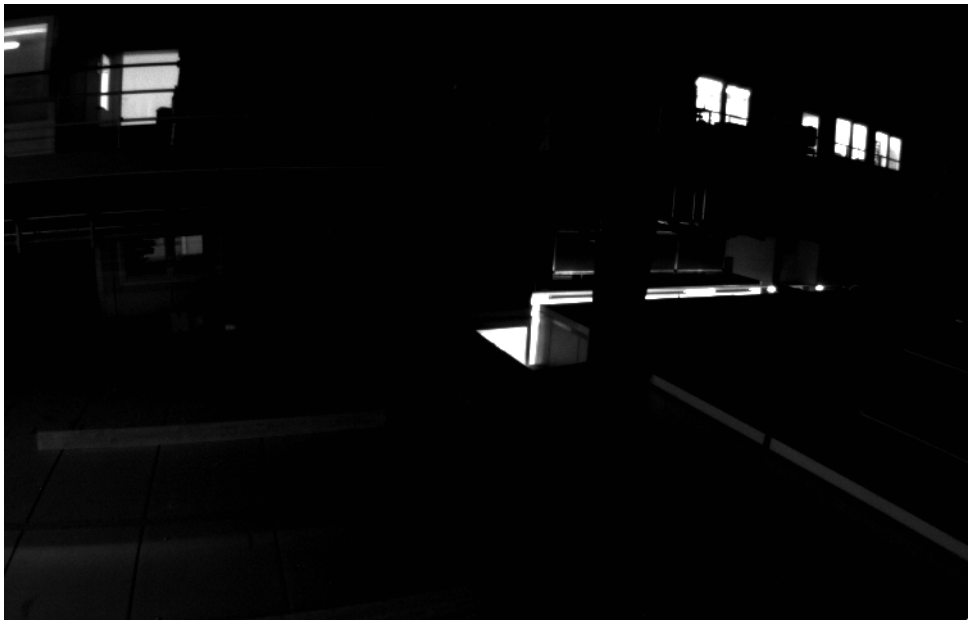

Figure 6.2: A challenging area of MH05-difficult with low lighting

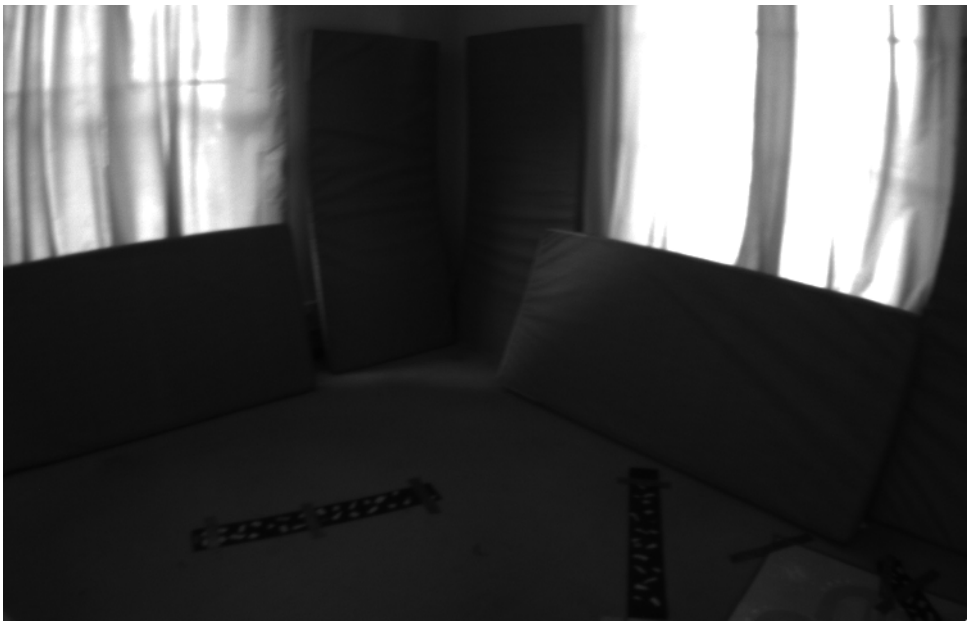

Figure 6.3: A challenging area of V103-difficult with low lighting and low scene texture

6.4 Experimental Setup

Following the methodology of [13], the proposed system is evaluated against the state of the art to compare the accuracy of their trajectory estimates. Each estimated trajectory is aligned to the ground truth according to [50] to account for differing initialization results and then the root mean squared error (RMSE) across the entire trajectory is computed. For each dataset, the system was initialized at a standstill with all states set to 0.

The configuration of the proposed system was chosen so as to be as fair a comparison as possible. As the feature-tracking module is designed after the one implemented in VINS-MONO it was configured as closely as possible to the VINS-MONO EUROC configuration. The ISAM2 backend was configured with the default parameters.

6.5 Evaluation of Proposed System

The following plots (figures 6.5 - 6.15) display the estimation performance of the proposed system in terms of position error on the EUROC dataset flights. For plots of the estimated trajectories and ground truth trajectories, see those in appendix C. An example trajectory is shown below in figure 6.4. In this plot, gray lines connect estiamted and truth poses at the same

timestamp. The performance of the proposed system is summarized and compared to other important visual-inertial estimators in table 6.1 at the end of this section.

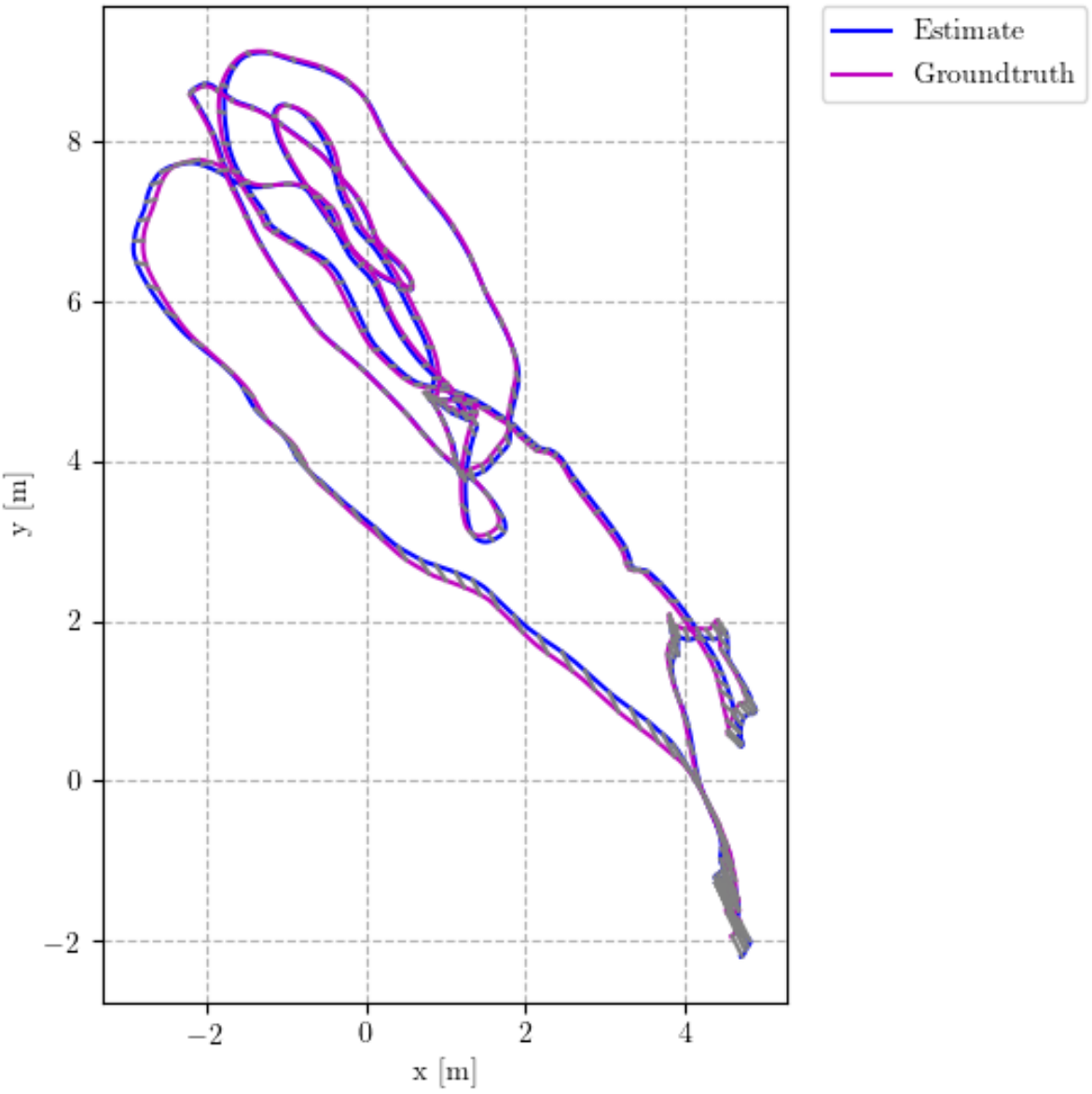

Figure 6.4: Estimated Trajectory for EUROC MH01

Figures 6.5 through 6.8 below display the translational error in the estimated INS pose for the MH01-easy, MH02-easy, V101-easy, and V201-easy datasets. These datasets consist of gentle flights with wide sweeping motions and gentle acceleration and rotation. They have constant illumination and have little to no motion blur. The proposed system generally

demonstrates error of under 20cm in each axis on these easier flights as shown and is generally unchallenged. Importantly, the lack of dramatic changes in error implies few to no invalid measurements entered the system.

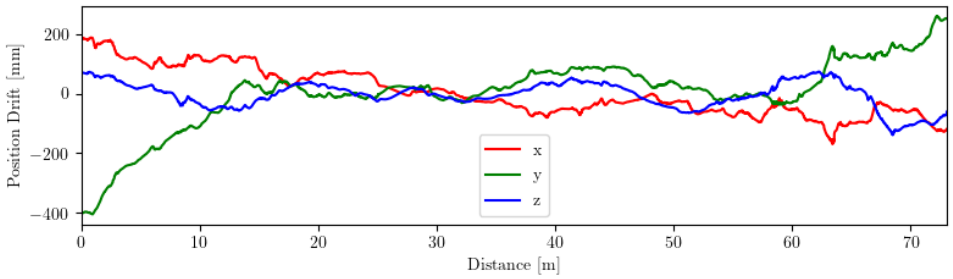

Figure 6.5: Trajectory Error: EUROC MH01

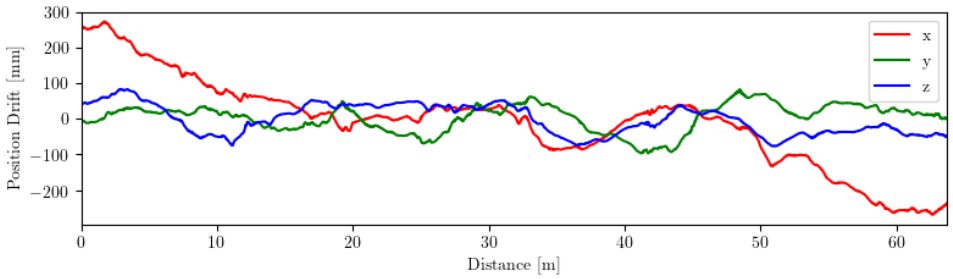

Figure 6.6: Trajectory Error: EUROC MH02

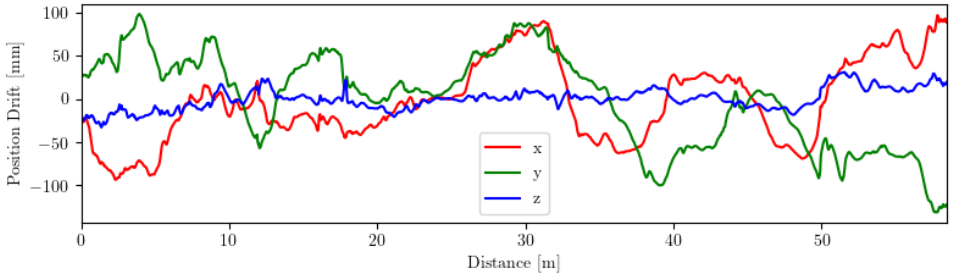

Figure 6.7: Trajectory Error: EUROC V101

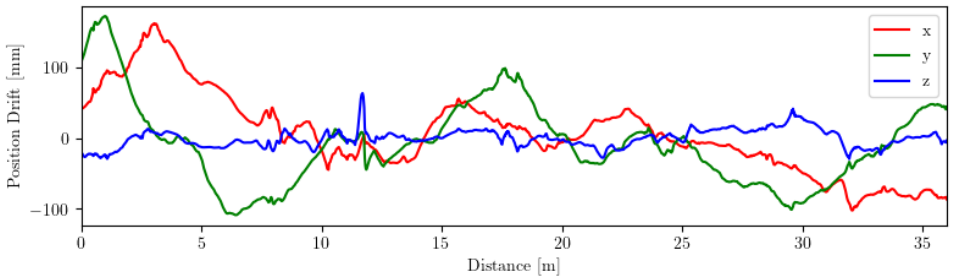

Figure 6.8: Trajectory Error: EUROC V201

Figures 6.9, 6.10, and 6.11 below display the translational error in the INS pose for the MH03-medium, V102-medium, and V202-medium datasets. These datasets include more aggressive flight than the earlier datasets but do not generally include lighting or texture challenges. Minor motion blur is visible in all three of these flights, but not enought to disturb the image processing module. In these flights, especially MH03-medium, we start to see more dramatic changes in error as the system must deal with less reliable measurements. Still, error generally remains under 30cm on all axes.

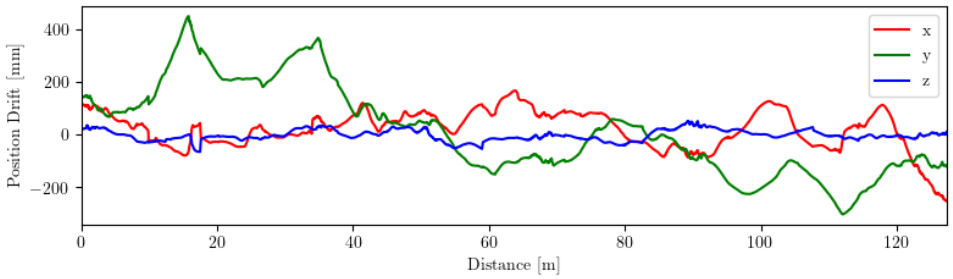

Figure 6.9: Trajectory Error: EUROC MH03

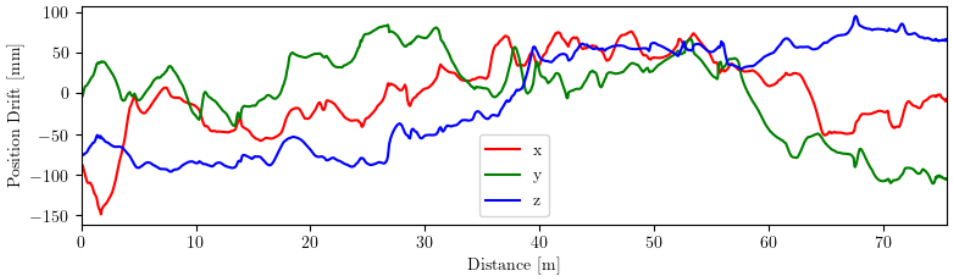

Figure 6.10: Trajectory Error: EUROC V102

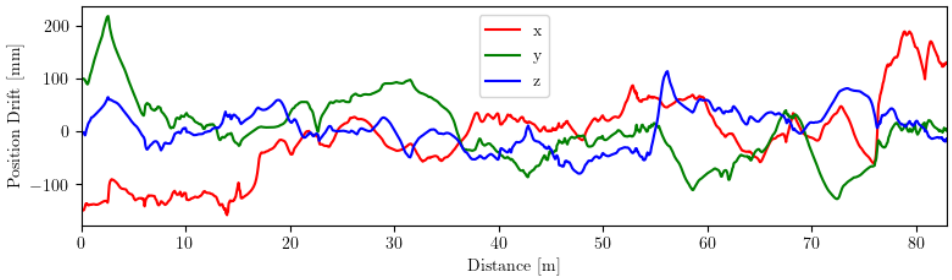

Figure 6.11: Trajectory Error: EUROC V202

Figures 6.12, 6.13, 6.14, and 6.15 below display the translational error in the INS pose for the MH04-difficult, MH05-difficult, V103-difficult, and V203-difficult datasets. These datasets are severely challenging as they include aggressive flight and significant challenges to image processing module in the form of rapid lighting changes, decreased scene texture, and significant motion blur. The aggressive motion and rapid lighting changes present in these flights clearly challenges the system, with the more aggressive flights (MH04-difficult, MH05-difficult) presenting more of a challenge than the flights focused on decreased scene texture (V103-difficult, V203-difficult). Overall the system maintains stable estimation with errors generally below 40cm on all axes.

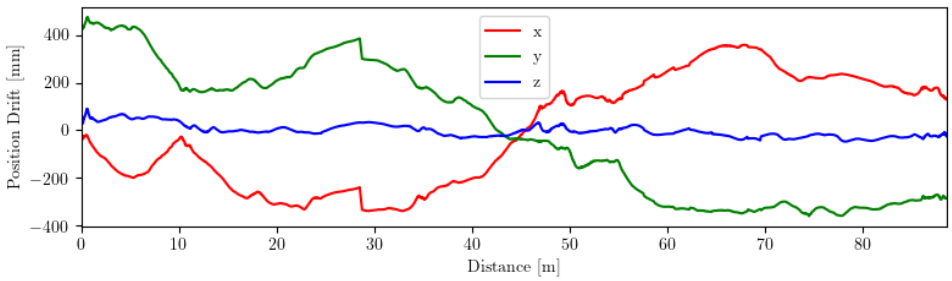

Figure 6.12: Trajectory Error: EUROC MH04

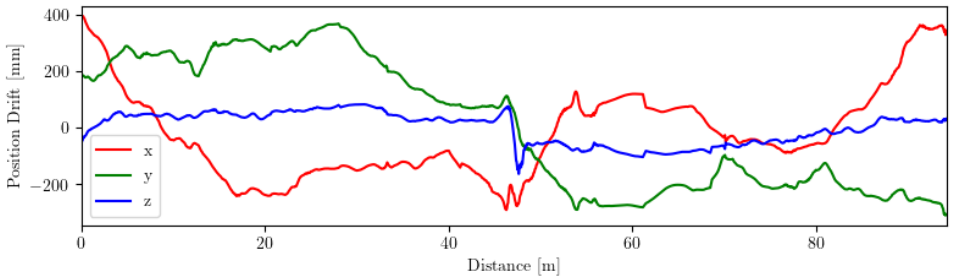

Figure 6.13: Trajectory Error: EUROC MH05

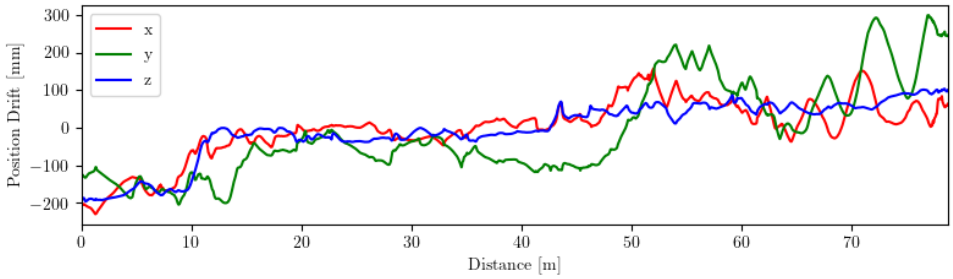

Figure 6.14: Trajectory Error: EUROC V103

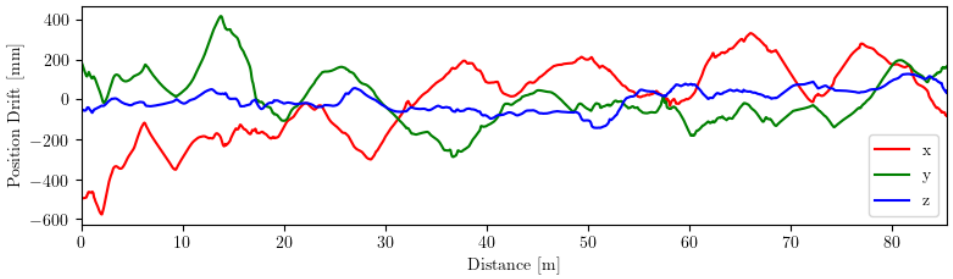

Figure 6.15: Trajectory Error: EUROC V203

6.5.1 Summary and Comparisons to State-of-the-Art

The final RMSE the proposed system achieved on each flight is summarized in table 6.1 below along with the performances reported in Delmerico et al. [13]. We have highlighted the best and second best performances on each flight in green and blue, respectively. Flights that were not able to be completed are marked with a $\times$. With the exception of SVO+GTSAM, all systems successfully completed each flight.

	MSCKF	OKVIS	ROVIO	VINS-MONO	SVO+GTSAM	**Proposed**
MH01	0.42	0.16	0.21	0.27	0.05	0.16
MH02	0.45	0.22	0.25	0.12	0.03	0.14
MH03	0.23	0.24	0.25	0.13	0.12	0.18
MH04	0.37	0.34	0.49	0.23	0.13	0.33
MH05	0.48	0.47	0.52	0.35	0.16	0.29
V101	0.34	0.09	0.10	0.07	0.07	0.07
V102	0.20	0.20	0.10	0.10	0.11	0.09
V103	0.67	0.24	0.14	0.13	×	0.17
V201	0.10	0.13	0.12	0.08	0.07	0.08
V202	0.16	0.16	0.14	0.08	×	0.10
V203	1.13	0.29	0.14	0.21	×	0.25

Table 6.1: EUROC Performance Results: Total Trajectory RMSE, partially reproduced from [13]. The best performance on each dataset is marked in green, second place in blue. Failure to complete the dataset is marked with an $\times$.

It is clear to see that SVO+GTSAM, when possible, outperforms the field. However, it is the only VINS to have any failed flights, reinforcing the motivation for the work in this thesis. The proposed system performs on-par with the current state-of-the-art VINS-MONO and places in the top three for each flight.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, we developed a new method for intelligently validating visual measurements so as to both maximize the reliability of the validation process while minimizing the delay caused by the need to validate measurements. The sensor models and requisite sensor processing techniques were described in 2. Chapter 3 developed the mathematical background necessary to understand and implement common visual-inertial systems. Two common visual-inertial systems were then presented in detail in chapter 4 along with the dynamic and measurement models used. Chapter 5 then provided an in-depth analysis of the relationships between the geometric properties of landmark observations and their relationships to the resulting quality of landmark estimation. This chapter then concluded by proposing a measurement validation strategy informed by these relationships. Chapter 6 then developed a visual-inertial estimator utilizing the proposed strategy and compared its performance against several benchmark visual-inertial systems. Based on these results, the proposed strategy was successful in mitigating the fragility of the selected optimization framework and in achieving competitive results with the state of the art.

7.2 Future Work

Although this thesis achieved its stated goals, several questions remain. First, the developed estimator is lacking several funcitonalities that other state-of-the-art systems include, such as intialization processes, zero-velocity detection, online sensor calibration, and support for loop

closuers or visual relocalization. Before the system can truly be said to belong in conversations with these existing systems at least some of these functionalities must be developed. Additionally, these functionalities will allow the system to be tested on more datasets in more difficult environments where the possibility of tracking loss is to be expected regardless of attempts to improve robustness.

The expansion of the system to support stereo or RGB-D cameras would be a significant step towards applying it to areas more relevant to the GAVLab's research, as monocular visual-inertial estimation performs poorly in the constant-velocity motion of our beloved autonomous ground vehicles. Adding functionality tailored towards specific platforms also motivates the integration of platform-specific constraints into further measurement validation schemes. Support for other sensors common on these vehicles would also move the system closer to being a true workhorse estimator.

References

[1] Sameer Agarwal, Keir Mierle, and Others. Ceres Solver.

[2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, June 2008.

[3] David S. Bayard, Dylan T. Conway, Roland Brockers, Jeff H. Delaune, Larry H. Matthies, Håvard F. Grip, Gene B. Merewether, Travis L. Brown, and Alejandro M. San Martin. Vision-Based Navigation for the NASA Mars Helicopter. In *AIAA Scitech 2019 Forum*, San Diego, California, January 2019. American Institute of Aeronautics and Astronautics.

[4] Michael Bloesch, Michael Burri, Sammy Omari, Marco Hutter, and Roland Siegwart. Iterated extended Kalman filter based visual-inertial odometry using direct photometric feedback. *The International Journal of Robotics Research*, 36(10):1053–1072, September 2017.

[5] Michael Bloesch, Sammy Omari, Marco Hutter, and Roland Siegwart. Robust visual inertial odometry using a direct EKF-based approach. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 298–304, Hamburg, Germany, September 2015. IEEE. 00349.

[6] Jean-yves Bouguet. Pyramidal implementation of the Lucas Kanade feature tracker. *Intel Corporation, Microprocessor Research Labs*, 2000.

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

[8] Martin Brossard, Silvere Bonnabel, and Axel Barrau. Invariant Kalman Filtering for Visual Inertial SLAM. In *2018 21st International Conference on Information Fusion (FUSION)*, pages 2021–2028, Cambridge, July 2018. IEEE.

[9] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The EuRoC micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, September 2016.

[10] Carlos Campos, Richard Elvira, Juan J. Gomez Rodriguez, Jose M. M. Montiel, and Juan D. Tardos. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM. *IEEE Transactions on Robotics*, pages 1–17, 2021.

[11] Chang Chen, Hua Zhu, Menggang Li, and Shaoze You. A Review of Visual-Inertial Simultaneous Localization and Mapping from Filtering-Based and Optimization-Based Perspectives. *Robotics*, 7(3):45, August 2018. 00003.

[12] Jeffrey Delmerico, Titus Cieslewski, Henri Rebecq, Matthias Faessler, and Davide Scaramuzza. Are We Ready for Autonomous Drone Racing? The UZH-FPV Drone Racing Dataset. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6713–6719, Montreal, QC, Canada, May 2019. IEEE.

[13] Jeffrey Delmerico and Davide Scaramuzza. A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2502–2509, Brisbane, QLD, May 2018. IEEE.

[14] Georgia Tech BORG Lab et al. GTSAM, July 2020.

[15] Philipp Foehn, Dario Brescianini, Elia Kaufmann, Titus Cieslewski, Mathias Gehrig, Manasi Muglikar, and Davide Scaramuzza. AlphaPilot: Autonomous Drone Racing. In *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, July 2020.

[16] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. 2015. Publisher: Unknown.

[17] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22, Hong Kong, China, May 2014. IEEE.

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

[19] Paul D. Groves. *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems*. GNSS technology and applications series. Artech House, Boston, 2008. OCLC: ocn172980486.

[20] Paul D. Groves. Navigation Using Inertial Sensors. *IEEE Aerospace and Electronic Systems Magazine*, 30(2):42–69, February 2015. 00048.

[21] C. Harris and M. Stephens. A Combined Corner and Edge Detector. In *Procedings of the Alvey Vision Conference 1988*, pages 23.1–23.6, Manchester, 1988. Alvey Vision Club. 17140.

[22] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, UK ; New York, 2nd ed edition, 2003. 26226.

[23] Sejong Heo and Chan Gook Park. Consistent EKF-Based Visual-Inertial Odometry on Matrix Lie Group. *IEEE Sensors Journal*, 18(9):3780–3788, May 2018.

[24] Joel A. Hesch, Dimitrios G. Kottas, Sean L. Bowman, and Stergios I. Roumeliotis. Consistency Analysis and Improvement of Vision-aided Inertial Navigation. *IEEE Transactions on Robotics*, 30(1):158–176, February 2014.

[25] Jianbo Shi and Tomasi. Good features to track. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94*, pages 593–600, Seattle, WA, USA, 1994. IEEE Comput. Soc. Press. 09648.

[26] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental Smoothing and Mapping. *IEEE Transactions on Robotics*, 24(6):1365–1378, December 2008.

[27] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, February 2012.

[28] Rudolph Emil Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[29] Rainer Kummerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. G^2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613, Shanghai, China, May 2011. IEEE.

[30] Chengcai Leng, Hai Zhang, Bo Li, Guorong Cai, Zhao Pei, and Li He. Local Feature Descriptor for Image Matching: A Survey. *IEEE Access*, 7:6424–6434, 2019.

[31] Stefan Leutenegger, Paul Furgale, Vincent Rabaud, Margarita Chli, Kurt Konolige, and Roland Siegwart. Keyframe-Based Visual-Inertial SLAM using Nonlinear Optimization. In *Robotics: Science and Systems IX*. Robotics: Science and Systems Foundation, June 2013.

[32] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, March 2015. 00696.

[33] Mingyang Li. *Visual-Inertial Odometry on Resource-Constrained Systems*. PhD Dissertation, UC Riverside, 2014.

[34] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004. 55127.

[35] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *In IJCAI81*, pages 674–679, 1981.

[36] Anastasios I. Mourikis and Stergios I. Roumeliotis. A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation. Technical Report, September 2006.

[37] Anastasios I. Mourikis and Stergios I. Roumeliotis. A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3565–3572, Rome, Italy, April 2007. IEEE. 00769.

[38] Anastasios I. Mourikis and Stergios I. Roumeliotis. A dual-layer estimator architecture for long-term localization. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8, Anchorage, AK, USA, June 2008. IEEE.

[39] Tong Qin, Shaozu Cao, Jie Pan, and Shaojie Shen. *A General Optimization-based Framework for Global Pose Estimation with Multiple Sensors*. 2019. _eprint: arXiv:1901.03642.

[40] Tong Qin, Peiliang Li, and Shaojie Shen. VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, August 2018. 00373.

[41] E. Rosten, R. Porter, and T. Drummond. Faster and Better: A Machine Learning Approach to Corner Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):105–119, January 2010. 01728.

[42] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*, pages 2564–2571, Barcelona, Spain, November 2011. IEEE. 05937.

[43] Joan Solà. Quaternion kinematics for the error-state Kalman filter. *arXiv:1711.02508 [cs]*, November 2017. arXiv: 1711.02508.

[44] Stanford Artificial Intelligence Laboratory et al. Robotic Operating System, May 2020.

[45] Hauke Strasdat, J. M. M. Montiel, and Andrew J. Davison. Real-time monocular SLAM: Why filter? In *2010 IEEE International Conference on Robotics and Automation*, pages 2657–2664, May 2010. ISSN: 1050-4729.

[46] L. von Stumberg, V. Usenko, and D. Cremers. Direct Sparse Visual-Inertial Odometry using Dynamic Marginalization. In *International Conference on Robotics and Automation (ICRA)*, May 2018.

[47] Nikollas Trawny and Stergios Roumeliotis. Indirect Kalman Filter for 3D Attitude Estimation, 2005.

[48] Yang Cheng, M. Maimone, and L. Matthies. Visual Odometry on the Mars Exploration Rovers. In *2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 1, pages 903–910, Waikoloa, HI, USA, 2005. IEEE.

[49] Teng Zhang, Kanzhi Wu, Daobilige Su, Shoudong Huang, and Gamini Dissanayake. An Invariant-EKF VINS Algorithm for Improving Consistency. *arXiv:1702.07920 [cs]*, March 2017. arXiv: 1702.07920.

[50] Zichao Zhang and Davide Scaramuzza. A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry. In *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2018.

[51] Alex Zihao Zhu, Nikolay Atanasov, and Kostas Daniilidis. Event-Based Visual Inertial Odometry. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5816–5824, Honolulu, HI, July 2017. IEEE.

Appendices

Appendix A

Quaternion Details

A quaternion is a mathematical object of the form $\boldsymbol{q} = q_4 + q_1 i + q_2 j + q_3 k$ where i, j, k are imaginary numbers such that $i^2 = j^2 = k^2 = ijk = -1$. There are a variety of choices in notation and convention [43]. This work selects the Hamilton convention in line with the existing robotics literature, mostly stemming from [47], [37]. This covention defines the quaternion as

$$\boldsymbol{q} = \begin{bmatrix} q_4 \\ \boldsymbol{q}_v \end{bmatrix} = [q_4, q_1, q_2, q_3]^T$$

where $\boldsymbol{q}_v = [q_1, q_2, q_3]^T$. A rotation quaternion is a quaternion of unit length, e.g. $||\boldsymbol{q}||_2 = 1$. A rotation quaternion can be understood to be an axis-angle rotation parameterization, where a rotation of angle θ about a unit vector $\boldsymbol{k}$ results in a quaternion

$$\begin{aligned} \boldsymbol{q} &= \begin{bmatrix} \cos(\theta/2) \\ k_x \sin(\theta/2) \\ k_y \sin(\theta/2) \\ k_z \sin(\theta/2) \end{bmatrix} \\ &= \begin{bmatrix} \cos(\theta/2) \\ \boldsymbol{k} \sin(\theta/2) \end{bmatrix} \end{aligned}$$

The multiplication of quaternions is defined as:

$$\begin{aligned}\boldsymbol{q} \otimes \boldsymbol{p} &= \boldsymbol{L}(\boldsymbol{q})\boldsymbol{p} \\ &= \begin{bmatrix} q_4\boldsymbol{I}_{3\times 3} - [q\times] & \boldsymbol{q}_v \\ -\boldsymbol{q}_v^T & q_4 \end{bmatrix} \boldsymbol{p} \\ &= q_4\boldsymbol{I}_{4\times 4} + \begin{bmatrix} 0 & -\boldsymbol{q}_v^T \\ \boldsymbol{q}_v & [\boldsymbol{q}_v\times] \end{bmatrix} \boldsymbol{p}\end{aligned}$$

where

$$[\boldsymbol{\omega}\times] = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}.$$

The inverse of a quaternion is defined as:

$$\boldsymbol{q}^{-1} = \begin{bmatrix} q_4 \\ -\boldsymbol{q}_v \end{bmatrix}.$$

The integration of a quaternion using a rotation is defined as:

$$\begin{aligned}\dot{\boldsymbol{q}} &= \frac{1}{2}\boldsymbol{\Omega}(\omega)\boldsymbol{q} \\ &= \begin{bmatrix} 0 & -\boldsymbol{\omega}^T \\ \boldsymbol{\omega} & -[\boldsymbol{\omega}\times] \end{bmatrix} \boldsymbol{q}\end{aligned}$$

The zero-th order discrete integration is then:

$$\begin{aligned}\boldsymbol{q}(t_{k+1}) &= \boldsymbol{\Theta}(t_{k+1}, t_k)\boldsymbol{q} \\ &= \exp\left(\tfrac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega})\Delta t\right)\boldsymbol{q} \\ &\approx \left(\boldsymbol{I}_{4\times 4} + \tfrac{1}{2}\boldsymbol{\Omega}(\boldsymbol{\omega})\Delta t\right)\boldsymbol{q}.\end{aligned}$$

The rotation matrix $\boldsymbol{C}_q$ or $\boldsymbol{C}(\boldsymbol{q})$ associated with the quaternion $\boldsymbol{q}$ is defined as:

$$\boldsymbol{C}_q = \left(q_4^2 - \boldsymbol{q}_v^T \boldsymbol{q}_v \right) \boldsymbol{I}_{3\times 3} + 2\boldsymbol{q}_v \boldsymbol{q}_v^T + 2q_4[\boldsymbol{q}_v \times]$$

The small angle approximation can also be applied to quaternions. Given an incremental rotation $\delta\theta$, the corresponding quaternion is then:

$$\begin{aligned} \delta\boldsymbol{q} &= \begin{bmatrix} \delta q_4 \\ \delta \boldsymbol{q}_v \end{bmatrix} \\ &= \begin{bmatrix} \cos(\delta\theta/2) \\ \boldsymbol{k}\sin(\delta\theta/2) \end{bmatrix} \\ &\approx \begin{bmatrix} 1 \\ \delta\theta/2 \end{bmatrix} \end{aligned}$$

The details of the JPL quaternion implementation are specified in more detail in [47]. An overview of quaternions in general is specified in [43].

Appendix B

Numerical Integration

We consider systems of the form $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$. An obvious and often good-enough method of integrating this function is *Euler's method*:

$$\boldsymbol{x}_{k+1} \approx \boldsymbol{x}_k + f(\boldsymbol{x})\Delta t$$

which is the easiest to implement but least accurate method of numerical integration. Linearizing f at $\boldsymbol{x}_k$ gives:

$$\begin{aligned}\boldsymbol{x}_{k+1} &\approx \boldsymbol{x}_k + \boldsymbol{F}\Delta t\boldsymbol{x}_k \\ &\approx (\boldsymbol{I} + \boldsymbol{F}\Delta t)\boldsymbol{x}_k\end{aligned}$$

which is a first-order approximation of the exact solution

$$\boldsymbol{x}_{k+1} = \exp(\boldsymbol{F}\Delta t)\boldsymbol{x}_k.$$

This first-order approximation also gives the state transition matrix $\boldsymbol{\Phi} \approx \boldsymbol{I} + \boldsymbol{F}\Delta t$.

A more involved but more accurate method is the *Fourth-order Runge-Kutta* method, also known as RK4:

$$\boldsymbol{x}_{k+1} = x_k + \frac{\Delta t}{6}(\alpha_1 + 2\alpha_2 + 2\alpha_3 + \alpha_4)$$

where

$$\alpha_1 = f(\boldsymbol{x})$$
$$\alpha_2 = f(\boldsymbol{x} + \frac{\Delta t}{2}\alpha_1)$$
$$\alpha_3 = f(\boldsymbol{x} + \frac{\Delta t}{2}\alpha_2)$$
$$\alpha_4 = f(\boldsymbol{x} + \Delta t\alpha_3)$$

Appendix C

Additional Trajectory Plots

This appendix contains full trajectory plots for all of the flights used in chapter 6.

C.1 Machine Hall

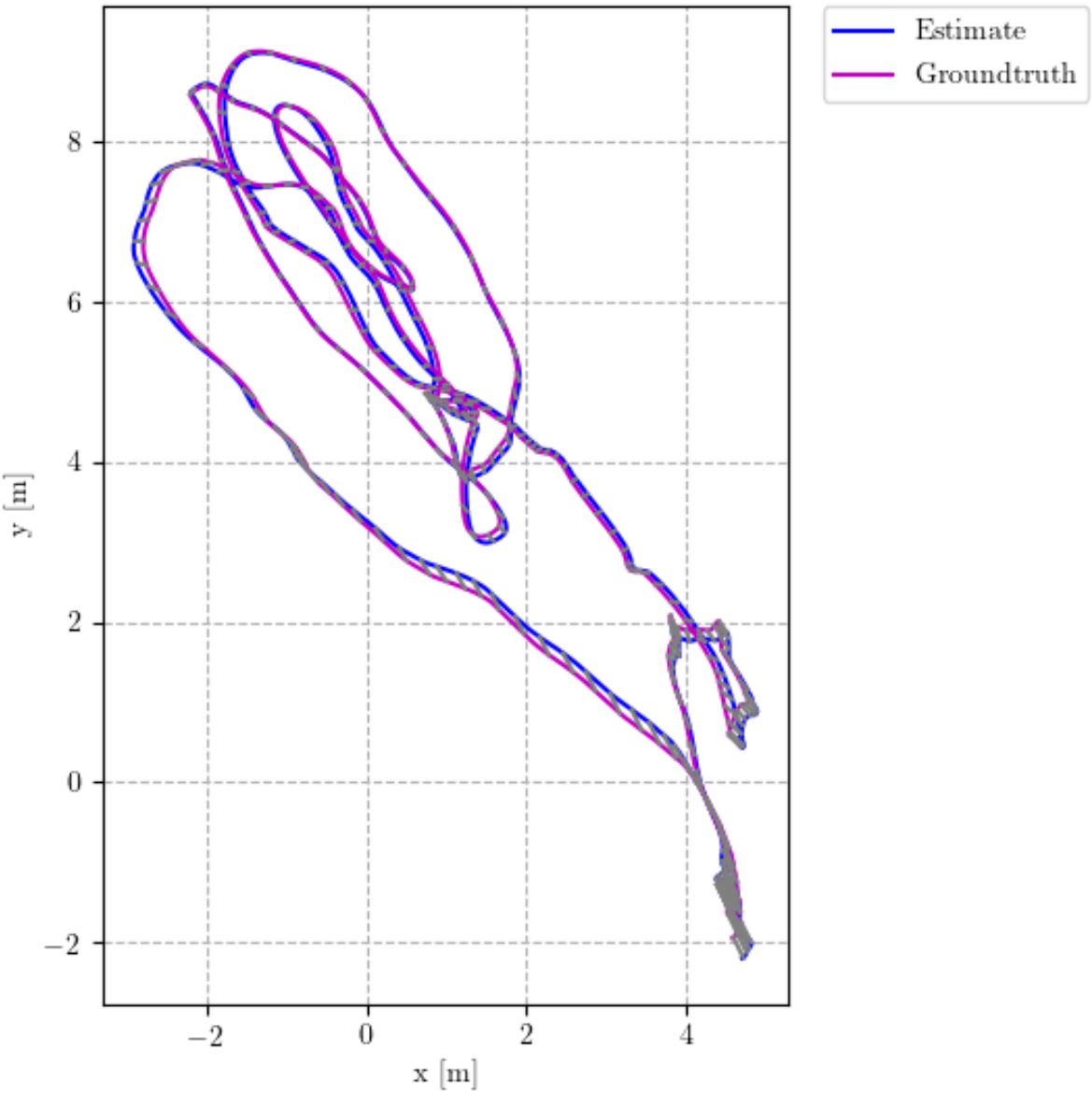

Figure C.1: Estimated Trajectory for EUROC MH01

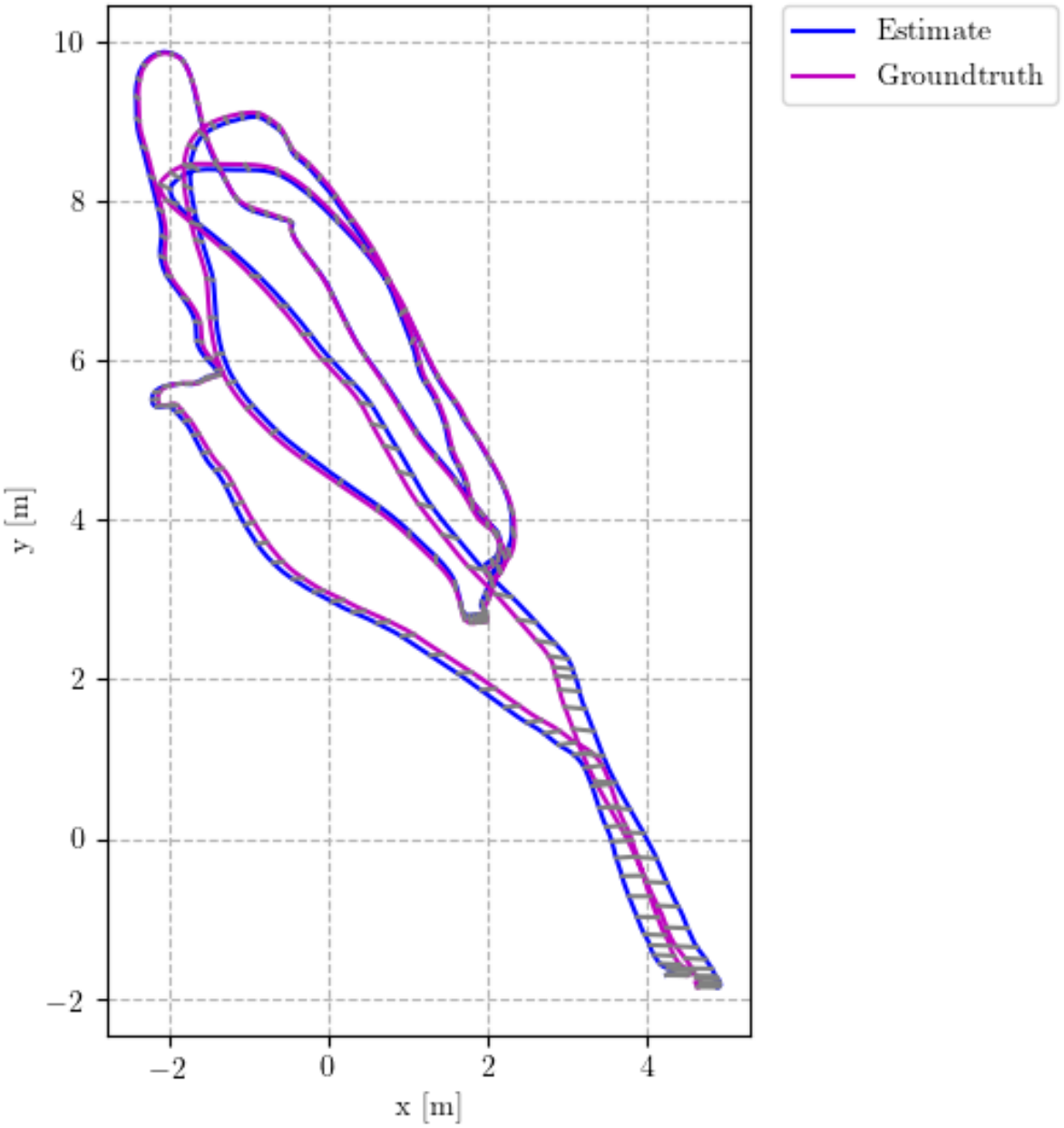

Figure C.2: Estimated Trajectory for EUROC MH02

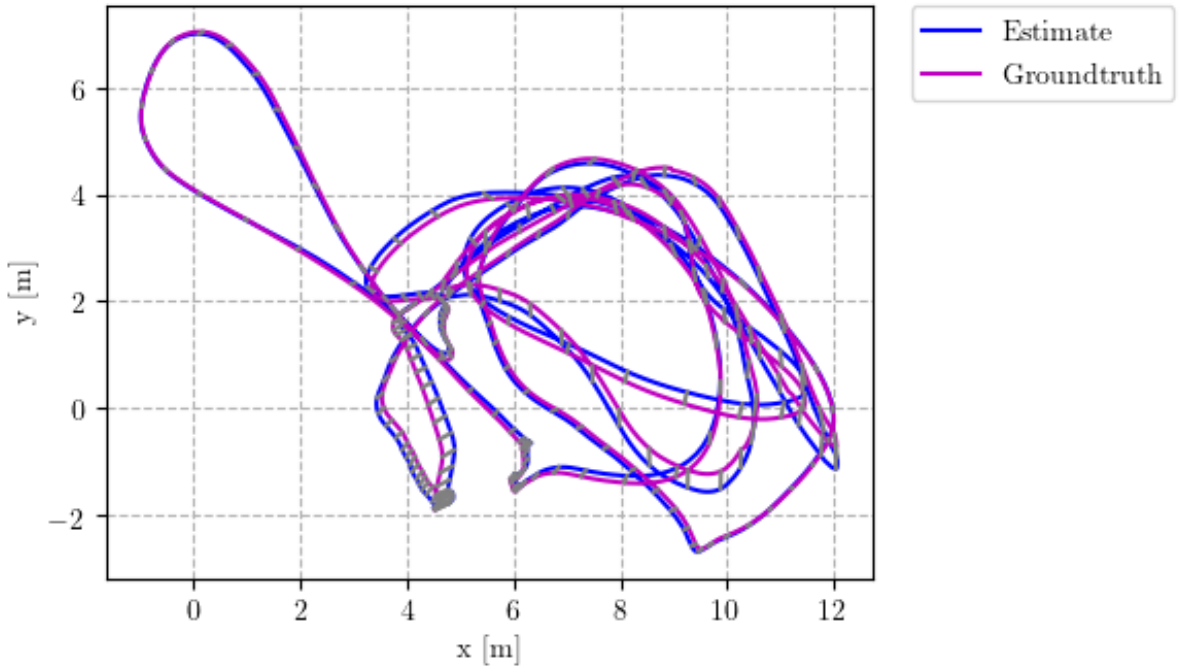

Figure C.3: Estimated Trajectory for EUROC MH03

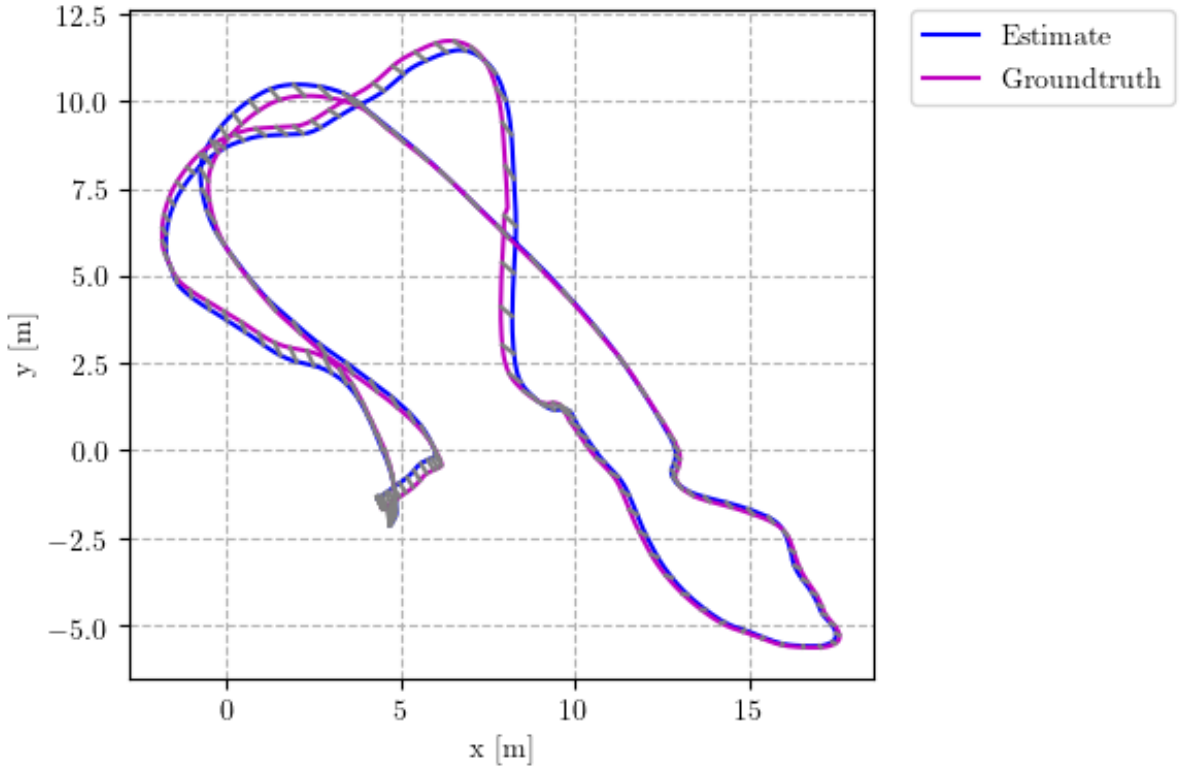

Figure C.4: Estimated Trajectory for EUROC MH04

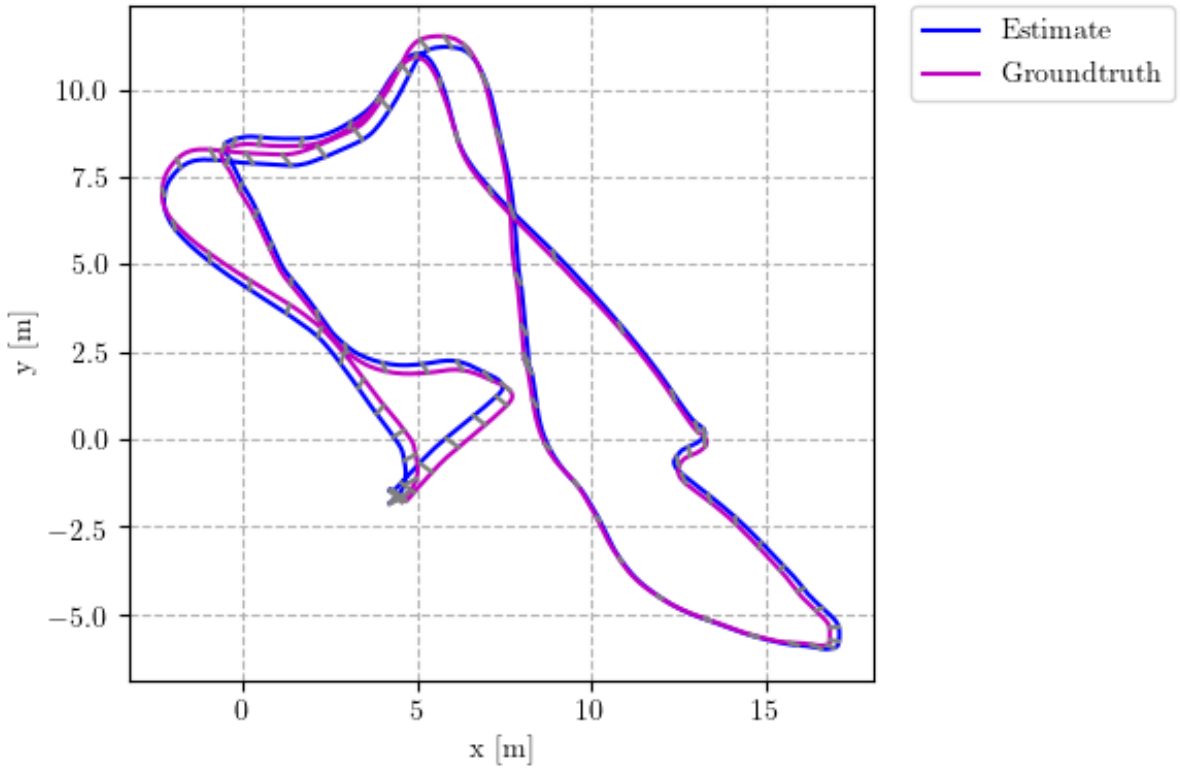

Figure C.5: Estimated Trajectory for EUROC MH05

C.2 Vicon 1

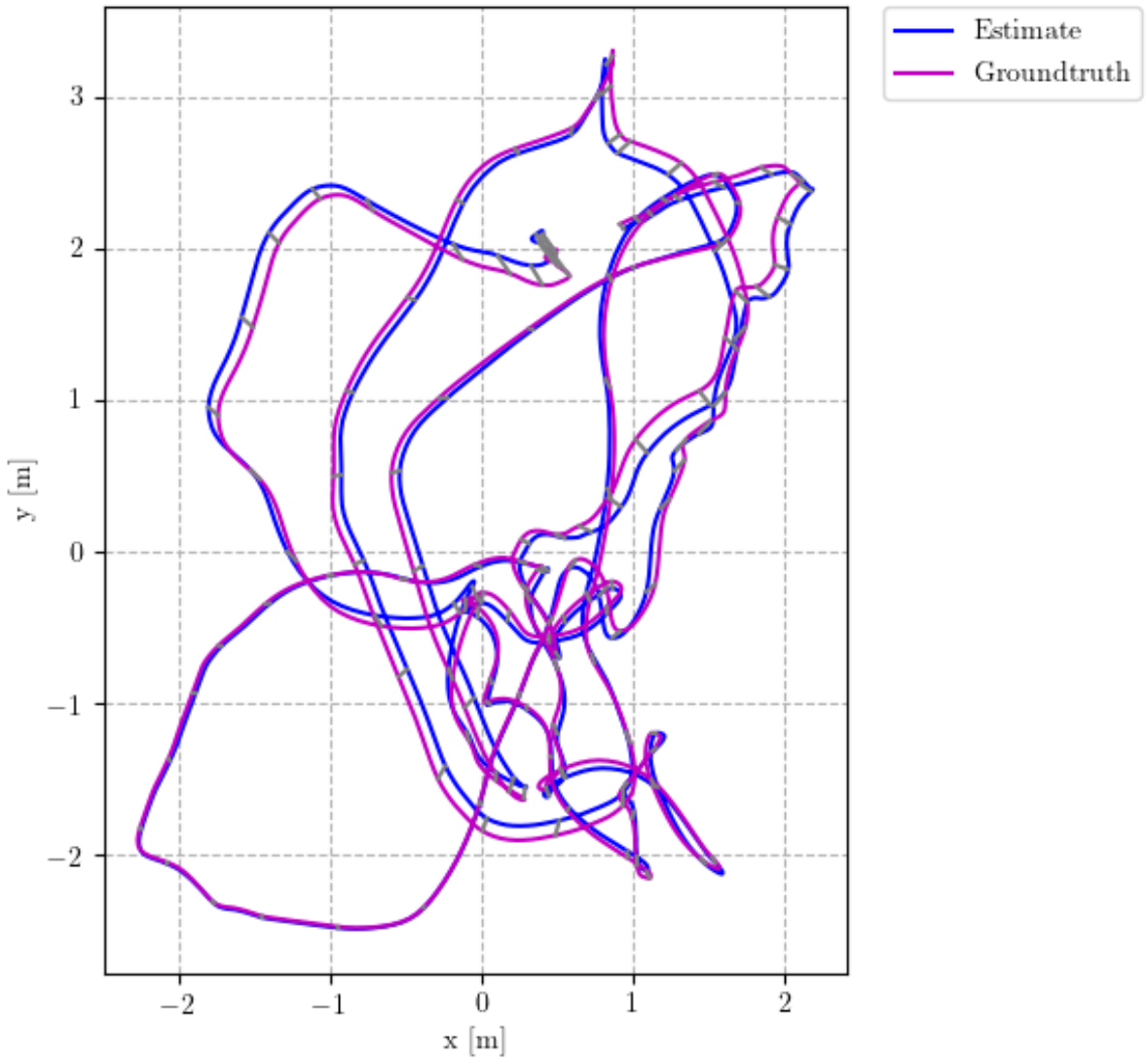

Figure C.6: Estimated Trajectory for EUROC V101

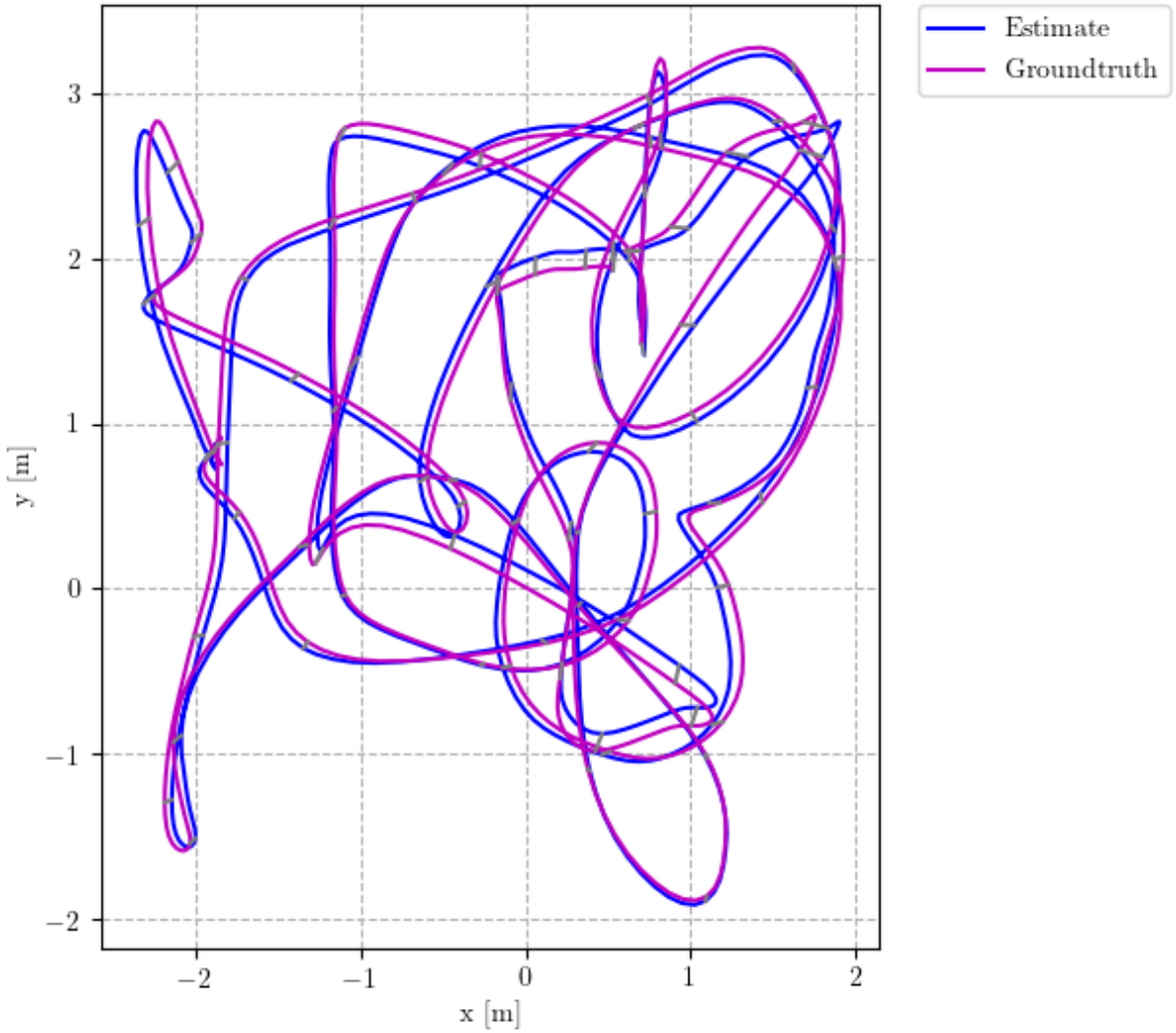

Figure C.7: Estimated Trajectory for EUROC V102

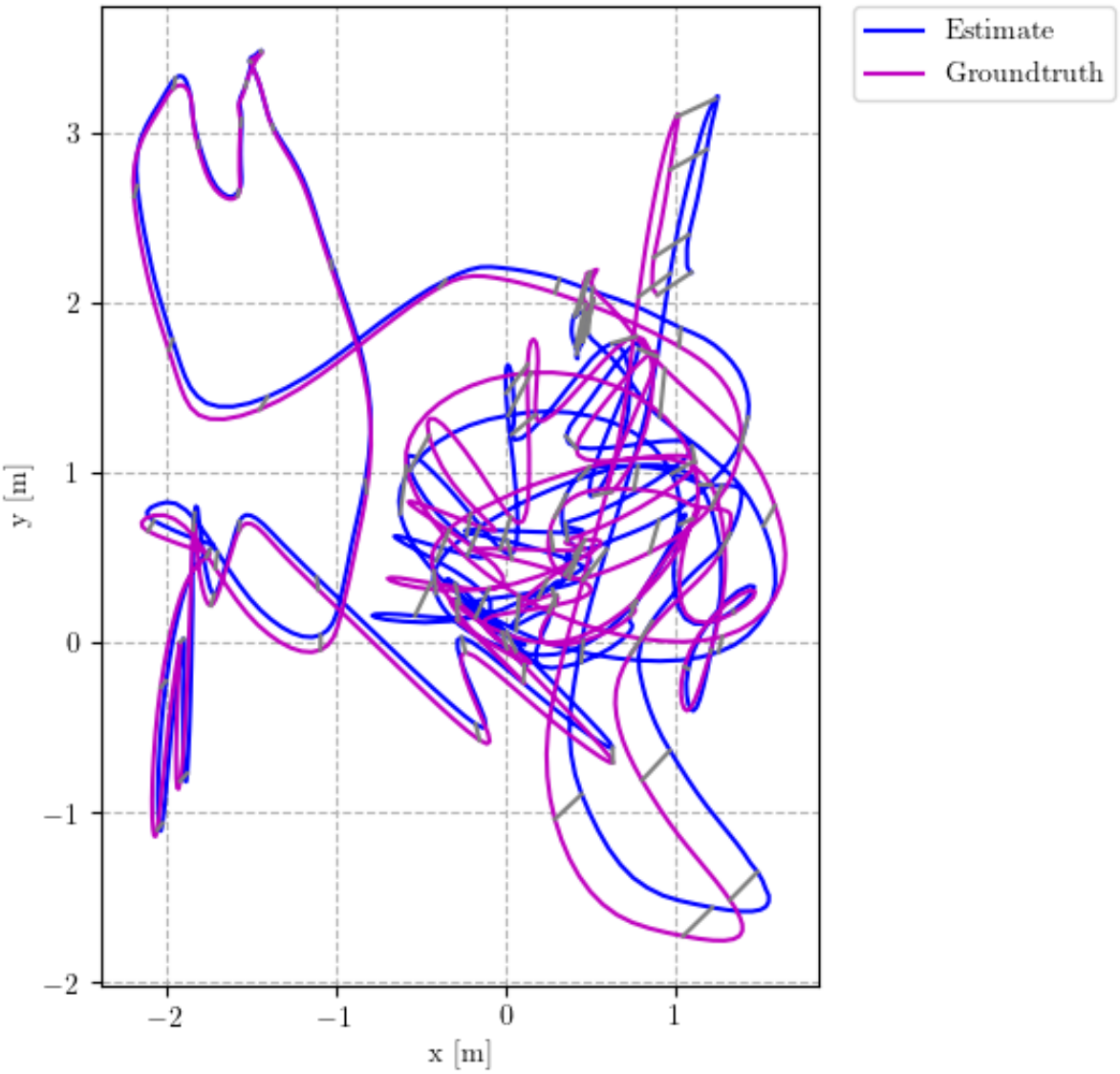

Figure C.8: Estimated Trajectory for EUROC V103

C.3 Vicon 2

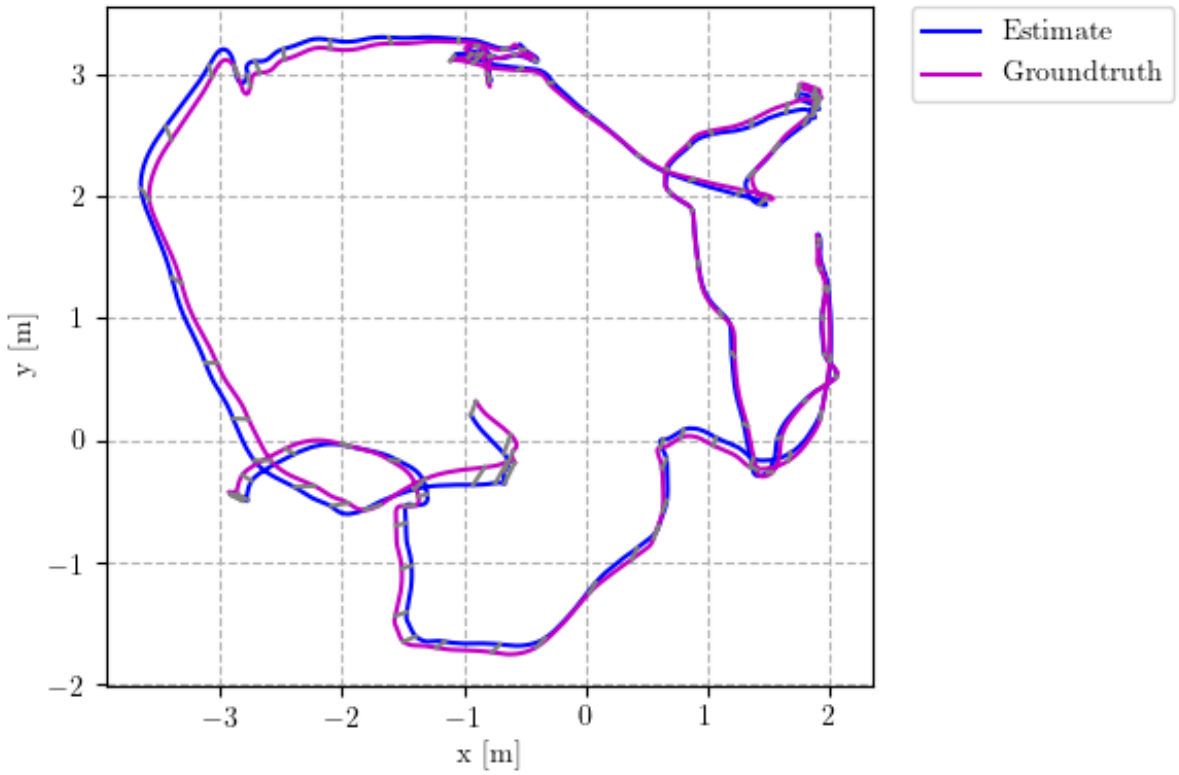

Figure C.9: Estimated Trajectory for EUROC V201

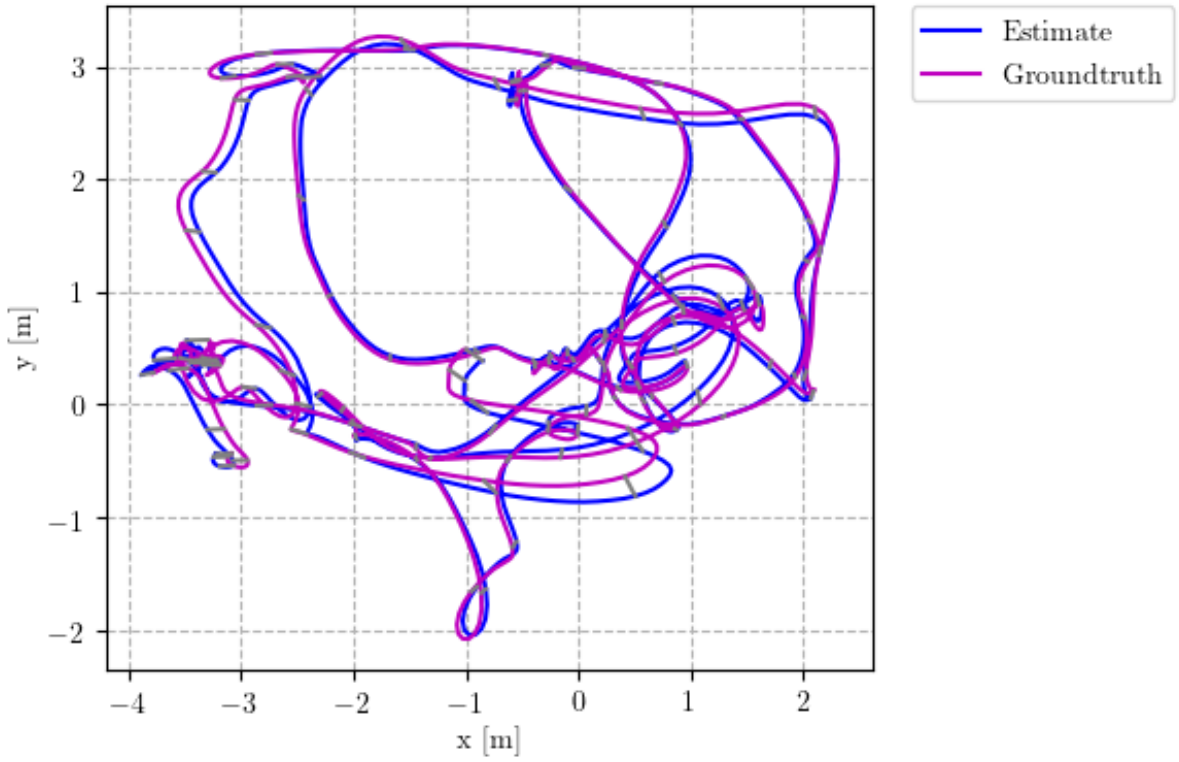

Figure C.10: Estimated Trajectory for EUROC V202

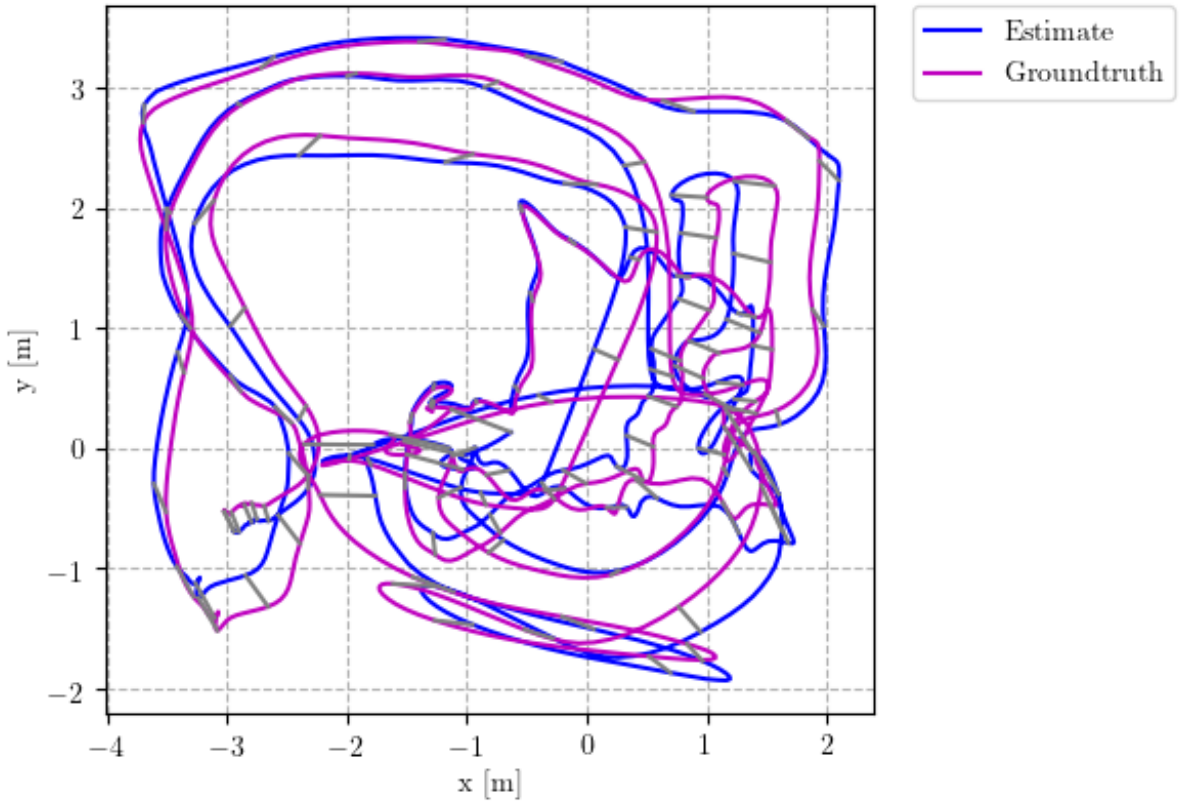

Figure C.11: Estimated Trajectory for EUROC V203